SYS-CON

IF ONLY... SOFTWARE MET ALL OUR NEEDS PAGE 4

JULY/AUGUST 2004

wldj™

THE LEADING INDEPENDENT MAGAZINE FOR WEBLOGIC™ PROFESSIONALS

WWW.WLDJ.COM

WebLogic on the Mac

A DEVELOPMENT ENVIRONMENT WORTHY OF SERIOUS CONSIDERATION...6

$9.99US $9.99CAN

TANGOSOL™

Spread the Load

With TANGOSOL COHERENCE™ many servers work as one. Partitioned data management ensures that each server carries its own load and contributes to the overall capacity and fault-tolerance of the cluster. Application capacity planning becomes a simple exercise in arithmetic.

www.tangosol.com

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

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

Managing your SOA apps can be slippery.

Remove the risks with Intersperse Manager.

Intersperse Manager™ is designed specifically to provide production management of applications deployed on BEA.

Leveraging the JMX standard, Intersperse Manager simultaneously manages both vertical apps and horizontal integrations—and market-moving capabilities enable applications to self-manage, self-heal, and self-protect.

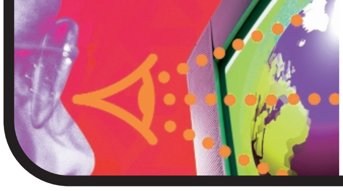
Intersperse Manager gives you the tools to reap the rewards and reduce the risks of J2EE-based SOA application deployments, and that puts you in control.

For a free demonstration, call Intersperse today at
866.499.2202
or visit www.intersperse.com.

intersperse™

Management solutions for service-oriented enterprise applications

626.535.2000
www.intersperse.com



If Only...

BY JOE MITCHKO

Rarely does a software product meet the expectations of each and every user. First of all, if it did, I guess there wouldn't be any need for further releases. We all have a wish list of sorts – if only this software program could do this or if only that could be better. Most of the time, you just grin and bear it, keep such thoughts to yourself, and accept the way it works until the next release. If only I had just a little clout to have the vendor design the software exactly the way I want it…

At this point, you can probably sense where I'm heading with all this, so without further ado, here are a few things that I think would be either nice additions or things to be improved upon in the BEA product arsenal.

If you are heavy on the presentation side of development, and are versed in using Java Page Flows (a.k.a., Struts), you may have come to appreciate the way WebLogic Workshop helps you graphically visualize page flows in your application. That's fine if the application just so happens to look like a page flow, but it doesn't. There isn't anything in Workshop (or any other IDE that I can see) that does a decent job of developing HTML and JSP pages using a WYSIWYG-type interface. It would be nice to see something where you can navigate through each page in the flow and actually see what the page will look like, and that allows you to develop and make updates on the fly. It would also be nice to be able to see and edit server- and client-side scripting along with the other client-side object components. This would save a lot of time and effort running through a "build, deploy, and run" cycle each time just to see the outcome of your changes.

And what's with the View Server Log feature in the WebLogic console? The viewer page allows you to customize the view – for instance, set how much of the log you would like to see according to time. But I'll be darned if I can figure out how to retain those settings for the next time I go to view the log. I find myself making the same adjustments each and every time.

It would also be nice to be able to view stack traces in a drill-down manner instead of having to sort through page upon page of stack trace data to look for the underlying culprit. The older I get, the more it makes me cross-eyed staring at that stuff. Any improvement that can be made that will help you quickly navigate through a stack trace and assist you in diagnosing the root cause of the problem would be a welcome addition.

Finally, for those of us who develop on the Windows-based operating systems (I would imagine that would be the majority of us), there must be a better way to run WebLogic Server other than having it run on a DOS command prompt (aside from running as a Windows service, which in certain situations, due to security, you don't have the liberty to use). Do you know how many times I've scratched my head wondering why an application page isn't refreshing, only to realize that the DOS command window stuck it in scroll-lock mode (from the last time I scrolled though it)? It gets you each and every time.

That's all I can think of for the moment; it's time to get back to work.

AUTHOR BIO...

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

CONTACT: joe@sys-con.com

wldj THE LEADING INDEPENDENT MAGAZINE FOR WEBLOGIC™ PROFESSIONALS

EDITORIAL ADVISORY BOARD
LEWIS CIRNE, TYLER JEWELL, WAYNE LESLEY LUND, CHRIS PELTZ, SEAN RHODY

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, JENNIFER VAN WINCKEL

ASSISTANT EDITOR
TORREY GAVER

RESEARCH EDITOR
BAHADIR KARUV, PhD

WRITERS IN THIS ISSUE
ANJALI ANAGOL-SUBBARAO, LABRO DIMITRIOU, PETER HOLDITCH, ANBARASU KRISHNASWAMY, THOMAS KRUSE, ALOIS LECHICKI, ALEX MACLINOVSKY, PRAKASH MALANI, JOE MITCHKO, KUNAL MITTAL, CHRIS PELTZ, CLAIRE ROGERS, PAT SHEPHERD, VIJAY MANDAVA

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

ADVERTISING SALES MANAGER
MEGAN MUSSA

ASSOCIATE SALES MANAGERS
KRISTIN KUHNLE • BETH JONES • DOROTHY GIL

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 • MATTHEW POLLOTA

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

WORLDWIDE NEWSSTAND DISTRIBUTION
Curtis Circulation Company, New Milford, NJ

NEWSSTAND DISTRIBUTION CONSULTANT
Brian J. Gregory /Gregory Associates/W.R.D.S
732-607-9941 - BJG Associates@cs.com

FOR LIST RENTAL INFORMATION:
Kevin Collopy: 845 731-2684, kevin.collopy@edithroman.com
Frank Cipolla: 845 731-3832, frank.cipolla@epostdirect.com

SYS-CON EVENTS

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.

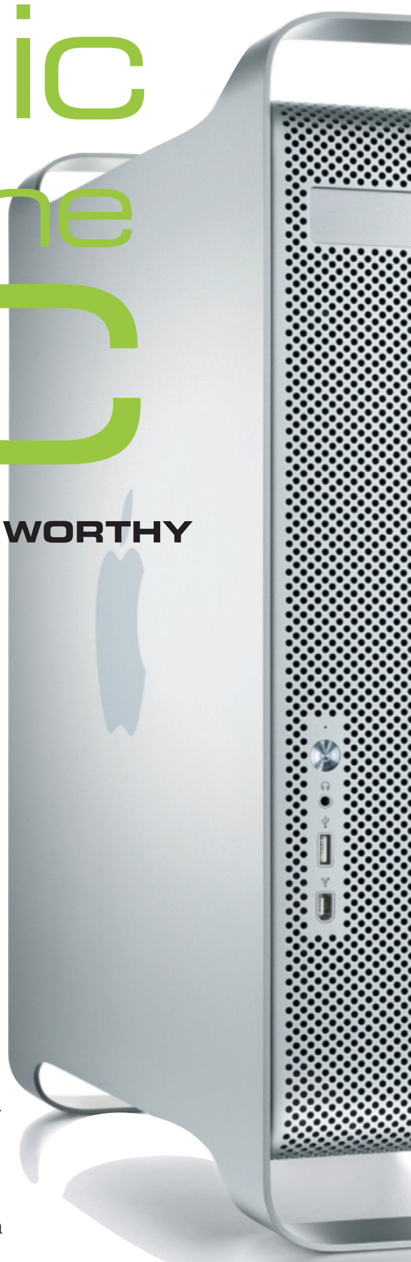
Get Wily.™

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

wily technology

WebLogic on the Mac

A DEVELOPMENT ENVIRONMENT WORTHY OF SERIOUS CONSIDERATION



You may not be aware of it yet, but Mac OS X – version 10.3, better known as Panther – is a great Java development environment.

BY **PAT SHEPHERD**

AUTHOR BIOS...

Pat Shepherd has been an active member of the Java revolution since its inception in 1995. He has written for several magazines and was a technical editor for the Sams book "*BEA WebLogic Platform 7*." He has worked at BEA for the last 4.5 years and is currently a global account architect, spreading the Java/SOA message everywhere he goes.

CONTACT...

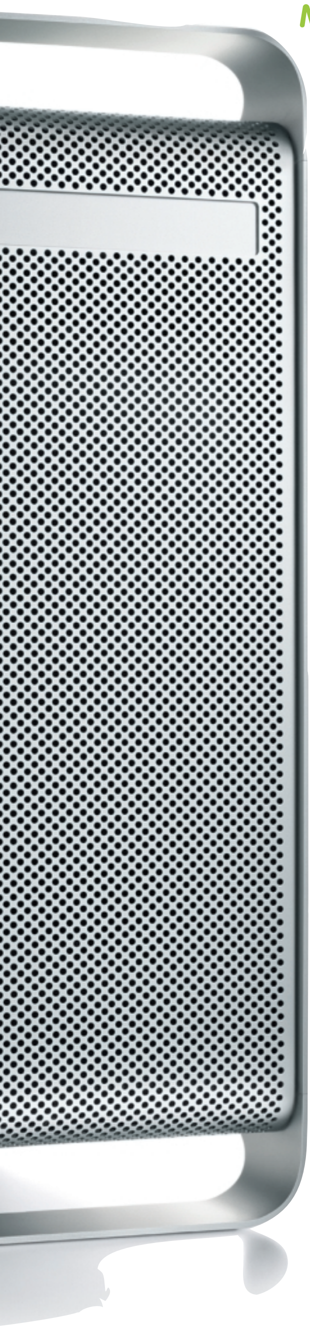
pat.shepherd@bea.com

I am a fairly recent Mac convert from the Windows and sometimes Unix/Linux world that I lived in. Maybe you are like me. Maybe aliens have abducted your friends and have turned them into Mac converts as well. Maybe you should join us. Speaking for myself, being able to use the Unix-based Mac OS X for WebLogic/J2EE development has given me yet another reason to be both a Java and a Mac Evangineer.

So what makes Mac OS X such a great Java development environment? Let's start with the best user interface on the planet. Add to that the core power of Unix and the ability to develop in an environment that more closely simulates the likely deployment environment for many enterprise applications (i.e., Unix-based Solaris, HPUX, and Linux). Top it all off with the use of BEA WebLogic Server and WebLogic Workshop for a surprising package of ease of use, power, and productivity.

This article is not a tutorial on WebLogic Server/WebLogic Workshop, or on Mac OS X, but it is an overview of where these technologies meet to create something that is unparalleled in the industry – a J2EE development environment on a Unix desktop done right. I will, however, provide you with basic instructions on how to set up your Mac on a productive WebLogic and WebLogic Workshop development environment. I used these instructions myself to set up WebLogic Server and Workshop successfully on my own Power Mac G5 and have used this environment to quickly and successfully create a small Web application. This application captures information, sends the information through a Page Flow (as a FormBean), writes that information to disk by using a Workshop FileControl, and finally exposes this functionality as a Web service. To do this, I created my own Workshop Domain using the Domain Configuration Wizard. All of this worked just fine, although there were some bumps along the way. I will share some of these with you.

It is important to note that at this time the Mac OS X and the Mac JVM are not official BEA-supported platforms. You must use this information within that context. So, it is up to the developer community to take it to the next level – take WebLogic Server and Workshop for a spin, report any bugs, make suggestions, and get involved.



Mac OS X Development Benefits

I wasn't going to spend much time justifying the Mac as a development platform, but I realized that a number of people may not have followed the astonishing progress of the Mac lately. You probably relate to the Mac as the "right brain" in the computing world, the creative side used by graphic artists and other nontechnical kin.

The release of Mac OS X offers the power of Unix as the foundational OS substrate with, arguably, the best GUI on the planet. The only other desktop OS to effectively combine Unix and GUI is Linux. We all love Linux, but from an ease-of-use and overall desktop capabilities standpoint, it can't hold a candle to the Mac's eadership in interface technology and implementation.

But don't just take my word for it; even the Father of Java, James Gosling, has fully embraced the Mac platform (see www.apple.com/pro/science/gosling/). According to the article, James and most of his development team love developing on a Mac because it blends the power of UNIX with the best MMI (man-machine interface) or user experience available. That is to say, it provides the power of a Unix server with the ease of operation that a Windows system aspires to.

An article in the March 2004 issue of ***Java Developer's Journal*** (Vol. 9, issue 3) talks through some additional benefits of Java on Mac OS X (www.sys-con.com/story/?storyid=43949&DE=1).

Here are some detailed reasons why Mac OS X is a great WebLogic/WebLogic Workshop development environment.

1. ***Java delivered as part of core system (i.e., JDK included):*** Java is well integrated into the OS, not an afterthought.
2. ***GUI:*** Unequaled ease of-use and history of excellence that other "alternative" desktops don't have.
3. ***Unix core:*** Utilities and productivity tools all at your fingertips.
   - Keep your valuable UnIX skills up while improving your productivity with WLW's 10 x productivity gains: You can issue all your favorite commands right from the Terminal window (available in the Applications/Utilities folder).
   - Solid UNIX distribution: Distribution of Unix on Mac OS X is BSD (Berkeley Software Distribution) that has been in existence since the '60s.
4. ***Integrates into Office Environment:*** Plays well with productivity tools like MS Office Suite for Mac (used in writing this article) so you will not be isolated from your team when it comes to sharing standard documents.
5. ***Simple networking:*** Hook your Mac up to your home network, allowing you to do cross-platform validation and clustering in the comfort of your home.

Now let's get on with the task at hand of installing, running, and developing with BEA WebLogic Server and WebLogic Workshop.

The installation process really is as easy as 1-2-3.

1. Download the correct distribution
2. Run the installer
3. Fix the Mac JVM distribution

With those simple steps completed you are now ready to run WebLogic Server and WebLogic Workshop, but you might want to refine the environment a little.

Installation

The following instructions assume that you are using Mac OS X 10.3 and have JDK level 1.4.2 installed. You can check this by (a) running "java –version", which should show you that the version is 1.4.2_03 (as of this writing); or (b) by navigating to /System/Library/Frameworks/JavaVM.framework/Versions, from which you should see folders for 1.2/1.3/1.3.1/1.4.2. If 1.4.2 is not the version you see as the most recent folder, make sure to upgrade before continuing.

Step 1: Download the Installation

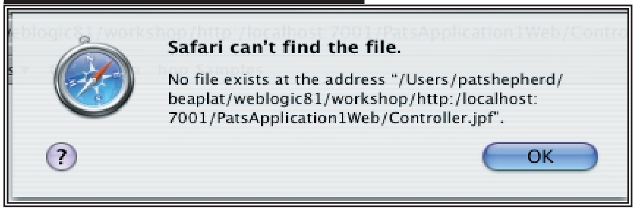
This part is easy, as long as you have a high-speed connection. The first thing to know is which distribution of WebLogic Server 8.1 SP2 to get from BEA's download site (http://commerce.bea.com/showproduct.jsp?family=WLS&major=8.1&minor=2). Your first choice is (a) WebLogic Platform Net Installer, (b) WebLogic Server Package Installer, or (c) Mail CD. I used option "b"and selected the IBM AIX (5.1,5.2) option, which will allow you to download a JAR file called pj_server812_generic.jar. Save this file to any folder that you use to store installation files. This file is 221.4 MB.

The file you download is the "generic" distribution that ends in ".jar" and is labeled as the AIX version on the download site. Note that this WebLogic Server/Workshop distribution, unlike, for example, the Windows distribution, does not include a JDK. We will use the JDK distribution that comes with Mac OS X 10.3 (Panther). More discussion on using the native JDK is in the "Fix the Mac Java Distribution"section.

Step 2: Run the GUI Installer

Now for installing the software. I will discuss using the GUI install wizard, though there is the option to install from the command line and even to do a silent installation. I will not bore you with the details, but you follow the directions listed in

FIGURE 1

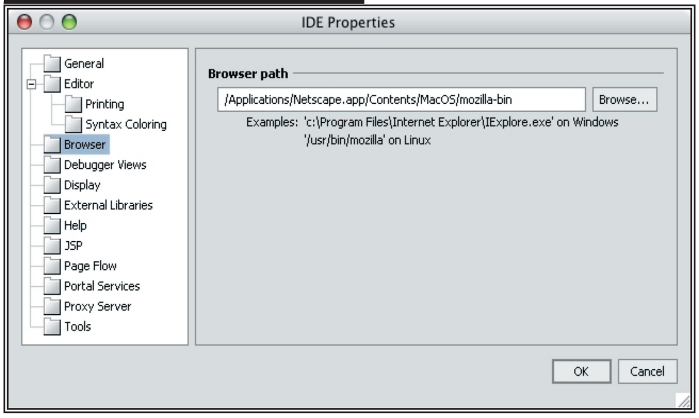


Safari can't find the file.

No file exists at the address "/Users/patshepherd/beaplat/weblogic81/workshop/http:/localhost:7001/PatsApplication1Web/Controller.jpf".

Safari is unable to launch

FIGURE 2


IDE Properties

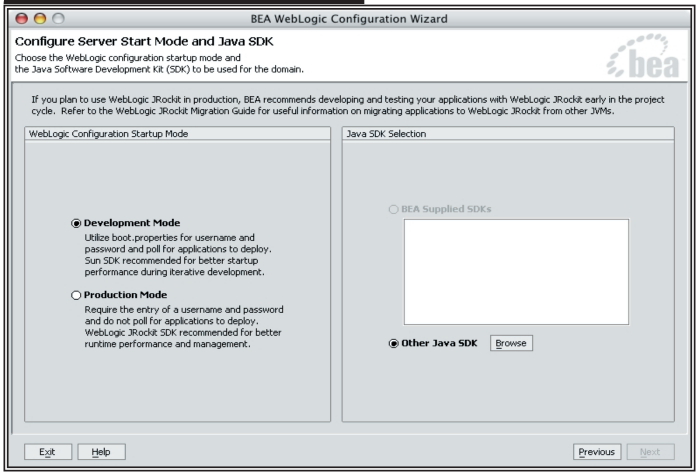
- General
- Editor
  - Printing
  - Syntax Coloring
- Browser
- Debugger Views
- Display
- External Libraries
- Help
- JSP
- Page Flow
- Portal Services
- Proxy Server
- Tools

Browser path: /Applications/Netscape.app/Contents/MacOS/mozilla-bin

Examples: 'c:\Program Files\Internet Explorer\IExplore.exe' on Windows
'/usr/bin/mozilla' on Linux

Trying to register the Netscape browser

FIGURE 3


BEA WebLogic Configuration Wizard

Configure Server Start Mode and Java SDK

Choose the WebLogic configuration startup mode and
the Java Software Development Kit (SDK) to be used for the domain.

If you plan to use WebLogic JRockit in production, BEA recommends developing and testing your applications with WebLogic JRockit early in the project cycle. Refer to the WebLogic JRockit Migration Guide for useful information on migrating applications to WebLogic JRockit from other JVMs.

WebLogic Configuration Startup Mode

[✓ Development Mode]
Utilize boot.properties for username and password and poll for applications to deploy. Sun SDK recommended for better startup performance during iterative development.

Production Mode
Require the entry of a username and password and do not poll for applications to deploy. WebLogic JRockit SDK recommended for better runtime performance and management.

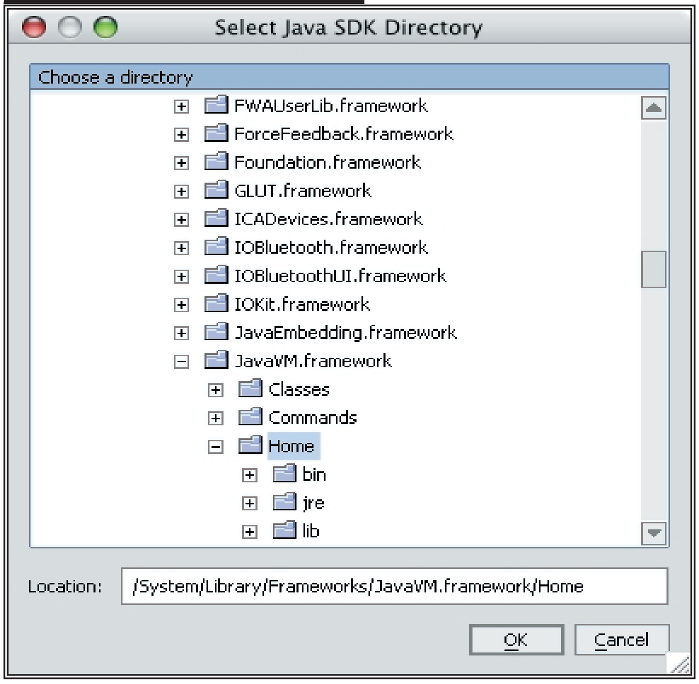
Java SDK Selection

BEA Supplied SDKs

[✓ Other Java SDK]

Specify the directory

FIGURE 4


Select Java SDK Directory

Choose a directory

- FWAUserLib.framework
- ForceFeedback.framework
- Foundation.framework
- GLUT.framework
- ICADevices.framework
- IOBluetooth.framework
- IOBluetoothUI.framework
- IOKit.framework
- JavaEmbedding.framework
- JavaVM.framework
  - Classes
  - Commands
  - Home
    - bin
    - jre
    - lib

Location: /System/Library/Frameworks/JavaVM.framework/Home

Browser tree

the installation guide at http://edocs.bea.com/wls/docs81/install/instprg.html#1054300.

Essentially, you run the following command and you are off and running/installing. The wizard walks you through the handful of steps to finish the installation.

```
java -jar pj_server811_generic.jar
```

Step 3: Fix the Mac Java Distribution

A word on the base install of your Mac JDK: the integration between the JDK and the OS is tighter than it is on other platforms. In fact Mac OS X has a layer of indirection that makes upgrading to new JDKs easy for the average user, but can be a little confusing at first glance to developers who need to know where, for example, the JAVA_HOME is.

If you try to run WebLogic Server or Workshop before you do this "fix," your servers will not start up because some directories are missing that Server and Workshop look for and, when missing, they will cause Server and Workshop to assume that you are not pointing to a valid JDK install. Further background information on the Mac JVM layout can be gleamed from an outstanding book, *Mac OS X for Java Geeks* by Will Iverson (O'Reilly; 2003).

Issue the following commands to allow WebLogic Server/Workshop to see your JDK as valid:

- ***sudo su:*** This is the substitute-user command allowing a permitted user to execute a command as the superuser or another user, as specified in the sudoers file.
- ***mkdir -p /Library/Java/Home/jre/bin:*** Create the missing bin directory.
- ***mkdir /Library/Java/Home/jre/lib:*** Create the missing lib directory.
- ***cd /Library/Java/Home/jre/bin***
- ***ln -s ../../bin/java:*** Create a symbolic directory link to existing Java executables directory. A link is useful for maintaining multiple copies of a file in many places at once without using up storage for the "copies'"; instead, a link "points" to the original copy.
- ***cd ../lib***
- ***ln -s ../../../Classes/classes.jar rt.jar:*** Create symbolic file links to existing Java library files.

Refining the Installation

After you open a terminal shell and you start the WebLogic Workshop IDE (or ISE as Gartner would call it), and attempt to run an application, Workshop tries to use Internet Explorer as the default browser. It is worth noting that you can theoretically set the browser to be anything you want. I was able to get Safari to partially work, meaning that it will auto-launch as the test harness from WebLogic Workshop when you hit the Run option. I was not, however, able to get the Mac version of Internet Explorer to auto-launch. The good news is that Netscape 7.1 for Mac OS X ran beautifully!

An issue that I ran into is that the HTML and Applet that is part of the WebLogic Server console completely and utterly refuses to load in the Safari browser. So, whereas I was using Safari for the Workshop browser, for administration in the WebLogic Server console, you must use another browser such as IE.

To specify the browser that Workshop will use, you need to set the IDE properties by going to the Tools>IDE Properties menu.

Initially, I selected "/Applications/Safari.app/Contents/MacOS/-Safari" to indicate Safari as the browser. It is interesting that the "Browse" button on this screen will let you navigate to the correct directory but this directory is not visible to Mac's Finder (analogous to Window

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

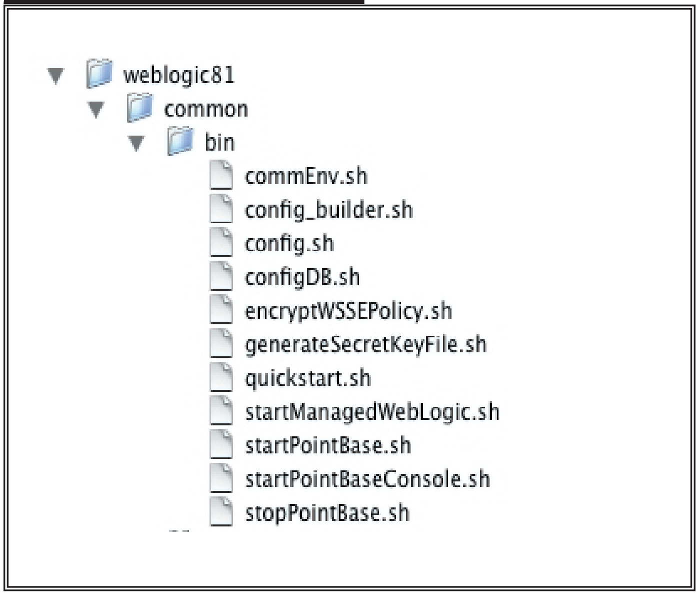
The wild blue yonder of operational monitoring and management is extremely unforgiving. Which means that going live with the monitoring software you used in development is a great way to go dead—quickly! You simply can't support operations if your staff is drowning in details provided by development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

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

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

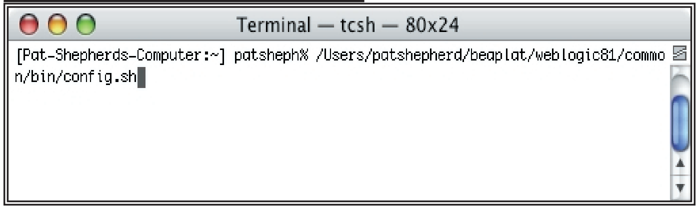
netiQ® Work Smarter®

FIGURE 5a



Specify the directory

FIGURE 5b



```
[Pat-Shepherds-Computer:~] patsheph% /Users/patshepherd/beaplat/weblogic81/commo
n/bin/config.sh
```

Using Terminal to launch the Domain Configuration wizard

Explorer) unless you CTRL-CLICK or right click the Safari icon and select "Show Package Contents" from the context menu and drill in from there.

I was able to get Safari auto-launched from Workshop's Run option, but a lot of extraneous information is pre-pended to the URL. If you clear this garbage (file:///Users/patshepherd/beaplat/weblogic81/-workshop/) and fix the single slash just after the port number (http:/localhost:7001/GettingStarted/Controller.jpf > http://localhost:7001/GettingStarted/Controller.jpf), then the application will run and you will even be able to debug your application. This, quite frankly, can be quite a speed bump in the standard build-test development cycle (see Figure 1).

When I found that Netscape 7.1 will auto-launch, I was overjoyed. That was until I tried to hit the WebLogic Server console and found that Netscape does not load the console's applet(s) either. Figure 2 shows the path I used to register the Netscape browser to auto-launch.

In summary, the Netscape browser is your best bet to use as Workshop's auto-launched browser, but none of the browsers will load the Server console applet(s). One quirk you will notice, though, is that Netscape will not automatically notify Workshop that the debug session is over when Netscape is closed – this is normal behavior on the Windows platform. The Netscape browser can be obtained at http://channels.netscape.com/ns/browsers/download.jsp.

Running the Domain Configuration Tool

The Domain Configuration Tool allows you to quickly create a domain from a predefined template.

While creating your domain, you will be asked to specify the directory where your JVM lives because, obviously, you are using your native Mac OS X JVM and none has been supplied as part of the generic distribution we have installed (see Figure 3).

I got my domain running by selecting the linked directory shown in the browser tree which is, as discussed earlier, a link to the /Library/-Java/Home directory we fixed earlier (see Figure 4).

Tips and Tricks

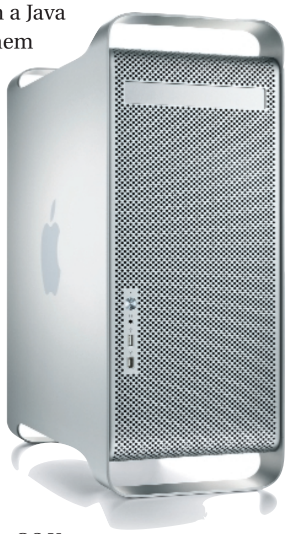
Cut and Paste

One joke I always tell is that "Copy, Paste" is the world's most prevalent form of code reuse. So, you might get frustrated using the standard Apple key combination of CMD-C for "copy" or CMD-V for paste. It is worth noting that the shortcut keystrokes in Workshop are exactly the same as they would be in the Windows world where, for example, CTRL-C is the standard "copy" combination. In fact, all key combinations are the same as their Windows brethren (with the exception of things like CMD-Q for "quit").

Terminal Velocity

Now is a great time to mention a very useful feature – the ability to drag a script file from finder and drop it right into the terminal shell. No more need to type in long path and file information to execute a start script, for example. Figures 5a and 5b show the result of dragging the file "config.sh"onto the terminal shell to start the Domain Configuration Tool, but the process works equally well for launching the start scripts for WebLogic Workshop or WebLogic Server.

Summary



For developers who are interested in a Java development environment that gives them "the best of both worlds" – the best user interface coupled with the power of a very mature UNIX distribution – you should seriously consider Mac OS X. While Apple was slow to keep the JVM versions in OS 9 up to date (never made it past 1.1.8), with Mac OS X Apple is now shipping a native 1.4.2 JVM that establishes Java as a first-class citizen in the Mac world.

Adding BEA's J2EE-based Application Platform Suite (APS), developers can leverage the power of Mac OS X to begin to develp world-class, service-oriented applications.

There are parts and pieces of Workshop that do not fully work on Mac OS X, but keep in mind that, as I stated earlier, Mac OS X is not currently a BEA-supported platform and has not gone through rigorous certification. With the BEA and Mac communities working together, Mac OS X could become a premiere development environment for truly serious developers. If you want to register with BEA your interest in making Mac OS X a BEA-supported platform, please e-mail Weblogic-on-osx@bea.com and Put "OS.X interest" in the subject.

Acknowledgements

I want to acknowledge some very useful postings by various users on O'Reilly Developer WebLogs hosted by BEA's Rod Chavez. There is a steady stream of postings there and interested readers should feel free to contribute to the momentum with their own postings. (www.oreillynet.com/pub/wlg/4091).

Businesses spend more than $80 billion annually on integration.

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

Automating the Exchange of Data Across an SOA

PANTERO

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

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

Contact us today!

781-890-2890

www.pantero.com

Managing the WebLogic Platform with HP OpenView

THE IT OPERATIONS PERSPECTIVE

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

In our earlier article (**WLDJ**, Vol. 3, issue 5), we discussed the importance of designing for manageability. Using a case study of an on-line shopping application, DizzyWorld, we showed the developer's perspective around application manageability. This included both the development of a JMX MBean and the instrumentation of a Java Page Flow on the BEA WebLogic Platform.

In this article, we look at manageability from the operator's perspective. We will show how management metrics can be defined and mapped to JMX MBeans, and how those metrics can be collected and monitored with HP OpenView. The end goal is to better illustrate how manageability can be designed into your application, and how this data can then be leveraged in building a truly adaptive IT environment. We will begin with background on how the HP OpenView architecture supports the management of the BEA WebLogic platform.

Background

Let's go through a quick introduction of HP OpenView, which consists of a suite of management products. For this example, we will use HP OpenView Operations (OVO) and the Smart Plug-in (SPI) for the BEA WebLogic Server. IT operators can use OVO and various SPIs to manage the entire IT infrastructure. This includes managing clusters of J2EE servers, databases, operating systems and other IT resources. In some cases OVO allows the capability of correlating management data between all of these systems and components. In this scenario, we will leverage OVO in combination with the SPI to provide a management solution that allows an operator to proactively monitor and manage the DizzyWorld application running on the WebLogic platform.

There are three basic components in the OVO architecture: agents, a management server, and

display stations (see Figure 1). The agents are responsible for interacting with each managed resource. A management server consists of several managers to proactively send and display messages based on events captured by the agents. The display stations then provide a single view into the managed environment.

OVO monitors, controls, and reports on the health of the IT environment by using different management technologies, including SNMP, ARM, WBEM, CIM, and JMX. OVO and the WebLogic SPI leverage managed resources that are exposed with JMX on the WebLogic Platform. Operators can manage and monitor any WebLogic instance in addition to any JMX-enabled application running on those servers. While performing diagnostics, operators can start and stop WebLogic or turn on tracing and monitoring for a particular WebLogic Server instance. In addition, they can easily discover a newly deployed WebLogic instance running on machines in the network and add it to their management view. This can save valuable time for the network or system administrator.

In addition, the SPI provides access to over 50 different predefined metrics that can be used for management purposes. Developers can use these metrics to get information such as average execution time for a servlet, the number of times per minute that EJB beans were not available from the free pool, or the percentage of the JMS queue that has been filled. These metrics are defined through the JMX MBeans provided by the WebLogic platform.

Operators use these metrics to create reports and graphs or to monitor certain aspects of the WebLogic platform. These monitoring capabilities are based on the concept of OVO policies. The settings within these policies define various conditions and thresholds to monitor within the WebLogic Server. Once in use, the WebLogic SPI policies allow information to be sent back to the OpenView Operations management server to help operators proactively address application problems and avoid serious disruptions to their business.

As Figure 2 shows, the WebLogic SPI performs a number of steps to collect the JMX data for each metric:

1. The OVO Monitor Agent starts based on the scheduling policies defined in OVO.
2. The JMX collector:
   - Accesses a User-defined Metrics (UDM) file that contains references to JMX objects and attributes,
   - Communicates with the MBean Server running within the WebLogic platform and retrieves the data,
   - Performs any calculations and submits the values to the OVO Monitor Agent, where they are compared against defined thresholds.
3. Based on the comparison results, an alarm message may be sent to the OVO console, or an e-mail or other type of notification can be sent to the person responsible for monitoring the application.

In the following sections, we'll take an in-depth look at defining the UDM, collector, and threshold policies in order to manage a JMX managed resource for our DizzyWorld application. Let's begin by taking a closer look at the Metric Definition file, which is key to accessing JMX data from the WebLogic platform.

Configuring a User-Defined Metric in OpenView

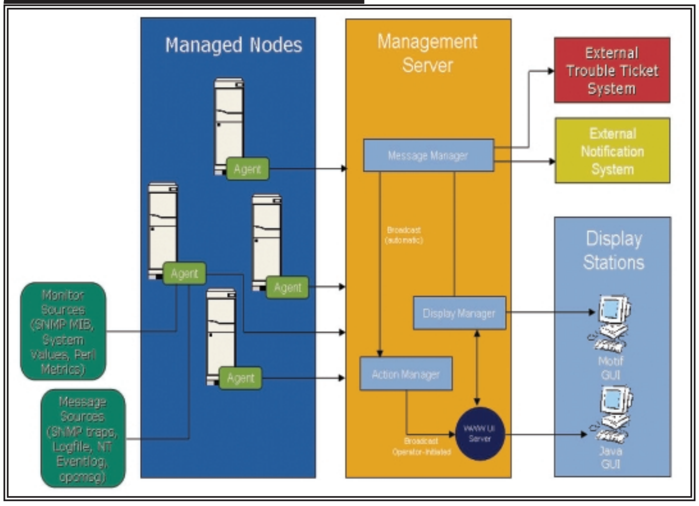
In our previous article, we developed a custom MBean for tracking the state of a shopping cart process and we want to use OVO to monitor it. Assuming this MBean has already been registered with the JMX Server, we will need to define this metric within OVO. The metric can then be used in OVO to generate alarms, create graphs, etc.

The first step is to create a UDM file. The UDM is an XML configuration file that defines what custom JMX MBeans will be collected by OVO. Multiple metrics can be defined in a single UDM file, and each metric can be either a reference to an MBean attribute or a calculated metric that combines other metrics together. Our DizzyWorld application has defined a very simple JMX MBean (see Figure 3).

This MBean defines two attributes, CartsCreatedCount and CartsCompletedCount, which refer to the total number of shopping carts that have been created and completed. We first want to define the mapping of each of these attributes within the UDM file. Listing 1 outlines the specific entry that can be added in the UDM file to retrieve the CartsCreatedCount.

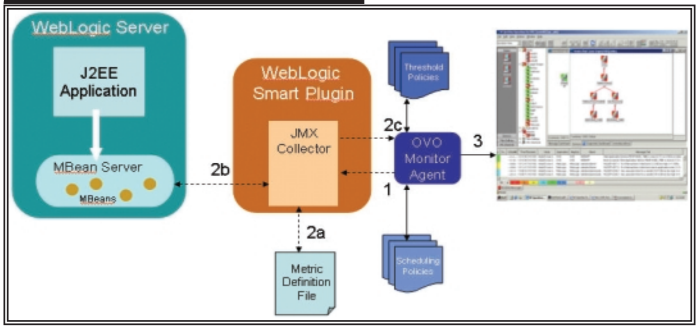
This metric is given the name ShoppingCart_Completed. The FromVersion element within the MBean mapping indicates what versions of WebLogic are supported by this metric. The JMX object and attribute name are specified via the <ObjectName> and <Attribute> elements. Note that the object name supports JMX pattern matching. We specify the pattern string, "*:",Name=MyShoppingCartManager" to locate the MBean for our shopping cart manager.

FIGURE 1


Managed Nodes
Agent
Agent
Agent
Agent
Monitor Sources (SNMP MIB, System Values, Perf Metrics)
Message Sources (SNMP traps, Logfile, NT Eventlog, opcmsg)
Management Server
Message Manager
Broadcast (automatic)
Display Manager
Action Manager
Broadcast Operator-initiated
WWW UI Server
External Trouble Ticket System
External Notification System
Display Stations
Motif GUI
Java GUI

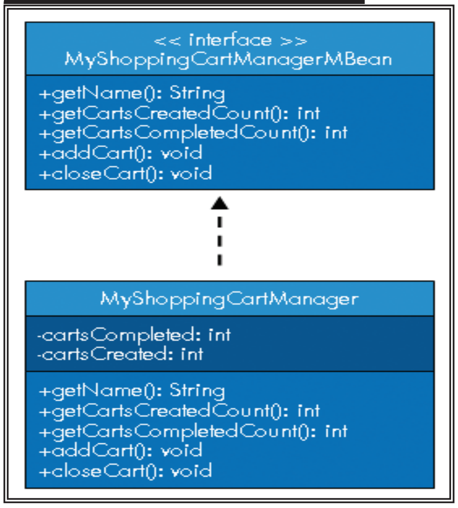
High-level OVO architecture

FIGURE 2



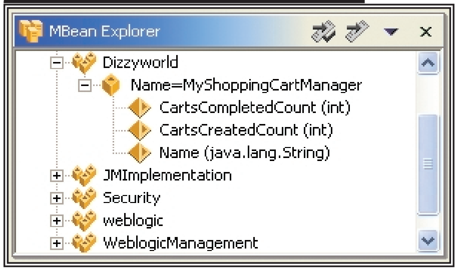
How the SPI works

FIGURE 3



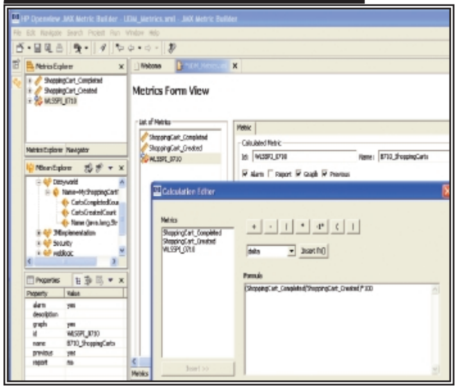
JMX Interface

FIGURE 4



Browsing MBeans in the JMX Metric Builder

FIGURE 5



Developing metrics in the JMX Metric Builder

You can take these individual JMX attribute values and combine them to create more useful, coarse-grained metrics. To do this, you have to construct a calculated metric that references the individual MBean values. In our example, we created two metrics, ShoppingCart_Completed and ShoppingCart_Created. These correspond to the CartsCreatedCount and CartsCompletedCount JMX attribute values. Listing 2 shows how a calculated metric can then be defined in the UDM file, corresponding to the percentage of all shopping carts that have been abandoned by users.

In this example, we have given this metric an ID of "WLSSPI_0710", a naming convention used by the WebLogic Server SPI to collect metrics. Our metric will be referenced as "710" during the configuration of the alarms and thresholds in the next section.

Once we've thought about the specific metrics we want to expose in the UDM file, the next step is to actually create the file. While it can be written using any text or XML editor, we recommend using the OpenView JMX Metric Builder, a tool supplied by HP to aid in UDM construction. This Eclipse-based tool provides facilities to connect and browse the JMX server and build metrics using a forms-based editor. Using the tool's MBean Explorer, you can browse the available JMX MBeans that you wish to use in the UDM file. Figure 4 shows a snapshot of our JMX Server with the MyShoppingCartManager MBean and related attributes highlighted.

Finally, using the Metrics Forms View, you can construct both the MBean metrics and the calculated metrics. Figure 5 provides an example of how our calculated metric for the shopping cart metric can be created using the JMX Metric tool.

Configuring OpenView Operations

With the UDM file created, the next step is to set specific threshold values for sending alarms to OpenView, and then to set the frequency with which we wish to collect this metric.

First, we will configure the generation of alarms through threshold settings. For our example, we want to generate an alarm if the percentage of shopping carts closed is at or below 50%. We must create a monitor policy to set the threshold settings for the metric. As shown in Figure 6, we configured the threshold to be a minimum threshold since we are trying to determine when the rate is less than or equal to 50%. Next, we determined what action we want performed when the threshold value is crossed. We could have configured a number of actions when a threshold is crossed, including sending an alarm to OVO, running a command or script, or generating a report. This flexibility is useful if you are working towards becoming an adaptive enterprise in which management data can be used to adapt the infrastructure to meet business demand.

For this example, we configured a message to be sent to the OVO console. As you can see from the Outgoing Message dialog box in Figure 6, we can set the Severity to a number of levels, including Major, Minor, or Critical. This helps the operator determine the importance of the message. In addition, we added instructions to help the operator diagnose and take appropriate actions (see Figure 7). The instructions include possible causes for the error and solutions for resolving it. This level of detail is important when trying to diagnose the problem.

Figure 8 shows a summary of the threshold and actions information. OVO evaluates the metric and when the threshold limit of 50% is reached, the appropriate message is generated in OVO.

Once the threshold settings have been made, the final step is to determine how often we want OVO to collect this metric. We configured the collector to run every minute. Additionally, we customized the collector program to collect data for our ShoppingCart metric.

Let's now turn back to the DizzyWorld application and see how we can manage and monitor the application.

Managing and Monitoring the Application

Our scenario consists of customers using the DizzyWorld application to do online shopping. Each customer would create a shopping cart during their shopping experience. However, not all of these customers complete their shopping at the Web site. Line-of-business managers want to know when the percentage of "completed carts" out of the "created carts" is 50% or lower. We have already created a threshold setting for 50% and we have the OVO data collector running every minute checking the metric against this threshold.

Within OVO, service maps are available that provide a view of your managed environment. Figure 9 displays a service map for WebLogic. Notice that there is no graphical indication of any event that needs attention at the moment in the "Before" picture. At this point, we will assume that a user has completed one shopping experience, so the carts completed and carts created counts are both 1 in the JMX MBean. We will begin a shopping experience, but instead of completing the purchasing process, we will exit the Web site before completing the purchase. That will leave the counts at 1 cart completed and 2 carts created in the JMX MBean. This will automatically fire an OVO event that indicates

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

diagnoSys

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

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

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

You need DiagnoSys.

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

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

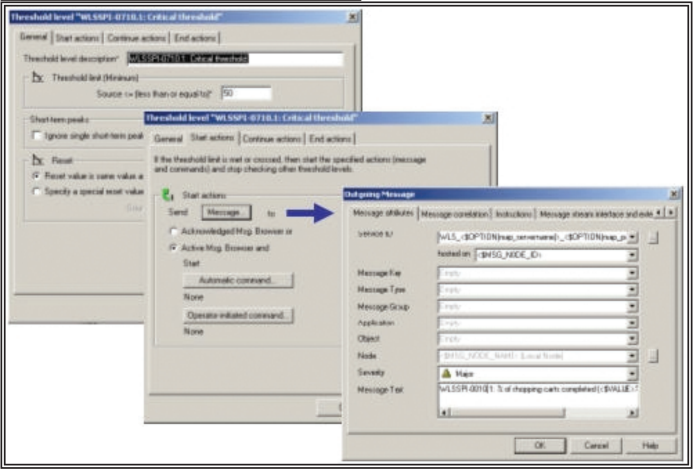
SILVER ANNIVERSARY 25 1979 - 2004

H&W

www.hwcs.com | 1.800.338.6692

Bridging the Enterprise Computing Gap For 25 Years

FIGURE 6


Threshold level "WLSSPI-0710.1: Critical threshold"
General | Start actions | Continue actions | End actions
Short-term peaks
Ignore single short-term peak
Reset
Reset value is same value a
Specify a special reset value
Start actions
Send | Message... | to
Acknowledged Msg. Browser or
Active Msg. Browser and
Start
Automatic command...
None
Operator-initiated command...
None
Outgoing Message
Message attributes | Message correlation | Instructions | Message stream interface and exte
OK | Cancel | Help

Threshold level description: WLSSPI-0710.1: Critical threshold

Threshold limit (Minimum): Source <= (less than or equal to): 50

If the threshold limit is met or crossed, then start the specified actions (message and commands) and stop checking other threshold levels.

Service ID: WLS_<$OPTION(map_servername)>_<$OPTION(map_p

Hosted on: <$MSG_NODE_ID>

Message Key: Empty

Message Type: Empty

Message Group: Empty

Application: Empty

Object: Empty

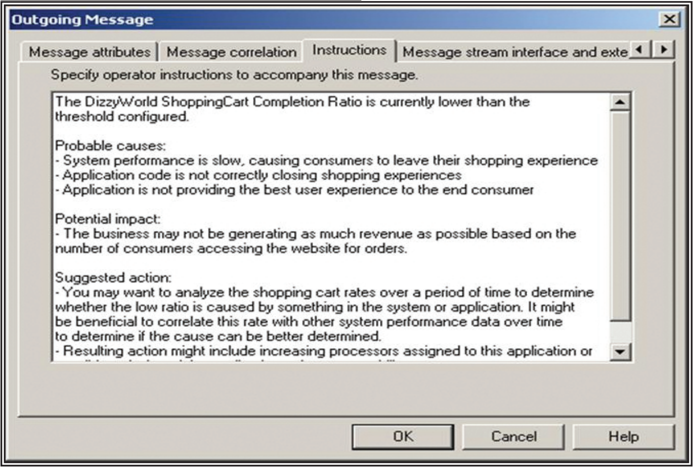
Node: <$MSG_NODE_NAME> (Local Node)

Severity: Major

Message Text: WLSSPI-0010.1: % of shopping carts completed (<$VALUE>

Setting thresholds, actions, and messages in OVO

FIGURE 7


Outgoing Message
Message attributes | Message correlation | Instructions | Message stream interface and exte
OK | Cancel | Help

Specify operator instructions to accompany this message.

The DizzyWorld ShoppingCart Completion Ratio is currently lower than the threshold configured.

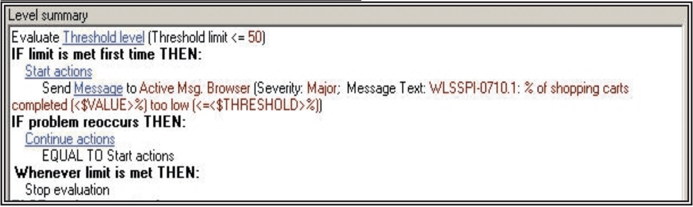
Probable causes:
- System performance is slow, causing consumers to leave their shopping experience
- Application code is not correctly closing shopping experiences
- Application is not providing the best user experience to the end consumer

Potential impact:
- The business may not be generating as much revenue as possible based on the number of consumers accessing the website for orders.

Suggested action:
- You may want to analyze the shopping cart rates over a period of time to determine whether the low ratio is caused by something in the system or application. It might be beneficial to correlate this rate with other system performance data over time to determine if the cause can be better determined.
- Resulting action might include increasing processors assigned to this application or

Instructions

FIGURE 8

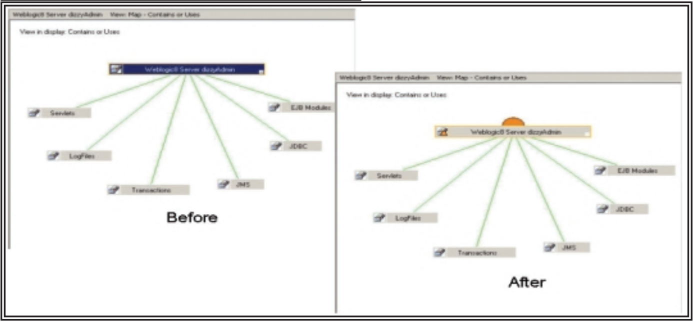


Level summary

Evaluate Threshold level (Threshold limit <= 50)
IF limit is met first time THEN:
Start actions
Send Message to Active Msg. Browser (Severity: Major; Message Text: WLSSPI-0710.1: % of shopping carts completed (<$VALUE>%) too low (<=<$THRESHOLD>%))
IF problem reoccurs THEN:
Continue actions
EQUAL TO Start actions
Whenever limit is met THEN:
Stop evaluation

Summary of threshold settings

FIGURE 9


Weblogic8 Server dizzyAdmin View: Map - Contains or Uses
View in display: Contains or Uses
Weblogic8 Server dizzyAdmin
Servlets
LogFiles
Transactions
JMS
JDBC
EJB Modules
Before
Weblogic8 Server dizzyAdmin View: Map - Contains or Uses
View in display: Contains or Uses
Weblogic8 Server dizzyAdmin
Servlets
LogFiles
Transactions
JMS
JDBC
EJB Modules
After

Service map of WebLogic

too few carts are being completed and the service map in OVO will indicate a problem (as shown in the "After" picture in Figure 9).

Since our application has been instrumented with JMX, certain shopping events update the MBean attribute values each time a user proceeds to checkout or places an order. We have configured OVO to collect on a metric that performs a calculation of the completed shopping rate and checks it against a threshold setting. If the threshold is crossed, an alarm is automatically sent to the OVO console.

We can switch to the message view and see a message in OVO indicating the threshold level has been crossed (see Figure 10). This information is important to the line of business manager and helps him determine how his business is doing. He could correlate this information with WebLogic performance data to help him determine why so many shoppers are leaving the site without completing a purchase. In addition, he could select the "Instructions" tab to get further instructions on how to correct the problem.

Now that we have integrated the JMX metrics with OVO, an operator may want to show those results in a graph. Using OVO's Graph Configuration tool, the operator can configure and view a graph of the business metric. OVO gives an operator the option of displaying many types of graphs from the information that has been collected through the metrics. In this example, we chose a bar graph using our Shopping Cart metric (see Figure 11).

We configured and graphed information on the business metric to give ourselves a graphical view of the number of completed shopping carts, allowing us to display that information for a given period. The graph shows data for a fairly simple metric but operators can correlate the Shopping Carts business metric with JVM Memory Heap, Time of Day, or number of users on the Web site, and take proactive steps to improve response time for the customers using the site.

Conclusion

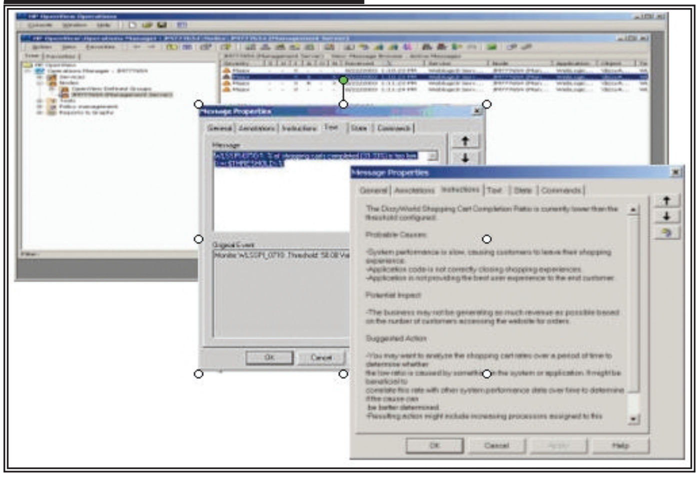
IT organizations are continually looking for opportunities to reduce costs and to improve operational efficiencies. It is clear that deployment and maintenance issues can be a big factor. In the past, developers have typically looked at management as the responsibility of IT Operations. There are clear benefits to designing and developing for manageability, ensuring that IT has the information they need to quickly analyze and respond to both operational and business needs.

In this article, we took a closer look at application manageability from the perspective of the IT operations staff. Specifically, we looked at J2EE manageability using JMX technology. This article showcased how HP OpenView Operations and the WebLogic SPI allow operators to manage the BEA WebLogic Platform more efficiently. With the SPI, IT operations can quickly view, analyze, and report on operational metrics such as EJB, Servlet, and JMS metrics.

There are situations where you may need to look at more than just the J2EE containers or platform components. To truly build an adaptive enterprise, where IT can better reflect the needs of the business, it's critical that operational and business data is available for monitoring and control. Gathering business metrics often requires the development of custom managed objects and artifacts.

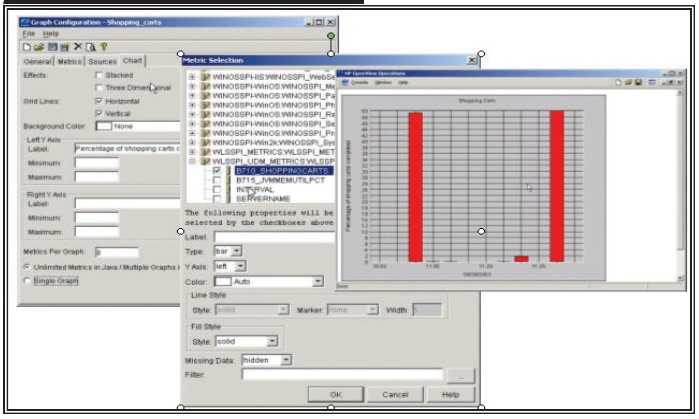
We showed how JMX technology can be used to collect these business metrics. Then, using the user-defined metrics capability within OVO, we showed how these JMX objects could be collected from the BEA WebLogic platform. We further showed how to configure OVO policies to monitor these business metrics for certain threshold conditions.

FIGURE 10



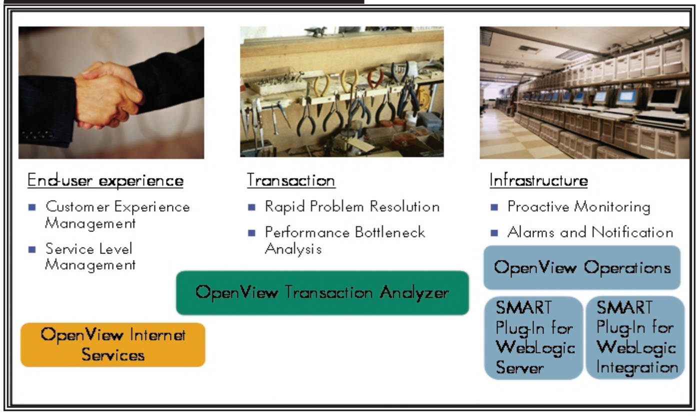
Monitoring alarms in OpenView

FIGURE 11



Graphing metrics

FIGURE 12


End-user experience
Customer Experience Management
Service Level Management
Transaction
Rapid Problem Resolution
Performance Bottleneck Analysis
Infrastructure
Proactive Monitoring
Alarms and Notification
OpenView Operations
OpenView Transaction Analyzer
OpenView Internet Services
SMART Plug-In for WebLogic Server
SMART Plug-In for WebLogic Integration

End-to-end management of the WebLogic Platform

Clearly, diagnosing applications for performance and SLA conformance requires a more end-to-end view of the environment. Management must take into account the client's perspective, all middleware platforms in deployment, as well as any back-end systems and databases. This analysis often requires a "single pane of glass" into the IT infrastructure, where IT operations and staff can correlate management data and perform root cause analysis.

To address these needs, HP provides a suite of Web application management products that support the BEA WebLogic platform (see Figure 12). From an end-user perspective, the HP OpenView Internet Services (OVIS) reports on the availability and response time of synthetic or real user transactions located as close to the user as possible. HP also offers OpenView Transaction Analyzer (OVTA) for tracing transactions from the client to the application server to backend databases. OVTA also provides a passive application monitoring capability, including application server response time and volume metrics. Combined with OVIS, this provides the user with a complete J2EE application server-level management solution.

Looking Ahead

Looking to the future, we see an increased use of Web services, service-oriented architectures, and grid-based networks. Use of these technologies further increases the demand for robust management software. But, management is not simply about reacting to problems in the systems and applications in development. Organizations need more adaptive management solutions where IT is better aligned with the business. One such solution offered by HP is Life Cycle Management for Web Services (LCM4WS). LCM4WS assists enterprises in better managing life-cycle issues such as deployment, versioning, and security of Web services.

There are a number of technical resources currently available to help get you started in application management and adaptive management. If you are a developer, the best place to start is HP's Dev Resource Central (http://devresource.hp.com). Here you will find a variety of technical papers, articles, software downloads, and forums focused around JMX and application manageability. If you want to learn more about HP OpenView and the products available for managing web applications, check out http://openview.hp.com/solutions/appm.

Reference

• Peltz, Chris and Rogers, Claire. "Instrumenting a Java Page Flow using JMX Technology". ***BEA WebLogic Developer's Journal***, Volume 4, issue 5.

Listing 1

```
<Metric id="ShoppingCart_Created" alarm="no">
  <MBean instanceType="single">
        <FromVersion server="6.0" update="1"/>
        <ObjectName>*:Name=MyShoppingCartManager</ObjectName>
        <Attribute>CartsCreatedCount</Attribute>
  </MBean>
</Metric>
```

Listing 2

```
<Metric id="WLSSPI_0710" name="B710_ShoppingCarts" alarm="yes"
graph="yes">
  <Calculation>
        <FromVersion server="6.0" update="1"/>
        <Formula>
                    (ShoppingCart_Completed/ShoppingCart_Created)*100
        </Formula>
  </Calculation>
</Metric>
```

PUBLISHING BUSINESS OBJECTS IN PORTALS

A BENCHMARK FOR THE IDEAL SOLUTION

BY **ALEX MACLINOVSKY**

AUTHOR BIOS...

Alex Maclinovsky is a solutions architect with Roundarch, Inc. For the last 15 years, he has focused on developing and architecting large distributed object systems on an enterprise, national and global scale. His professional interests include solution-oriented architectures, adaptive frameworks and OO methodologies.

CONTACT...

maclinovsky@yahoo.com

Current Web applications, especially portals, have become increasingly content driven. It led to development of a plethora of sophisticated and powerful Web Content Management Systems, or WCMS. They help to automate creation, management, reviewing, tagging, rendering, publication, maintenance, and deprecation of Web content. Usually, these systems support a wide variety of content types and formats; however, most of them stop short of supporting one crucial type – application data.

Using real-life examples, this article introduces the notion of business objects as a distinct category of highly structured publishable content. From this perspective, they have many of the same publishability requirements that call for use of a WCMS. Unfortunately, business objects also have a number of unique characteristics that prevent most WCMSs from being able to publish them out of the box. I'll establish a set of criteria as the benchmark for the ideal solution.

The article describes several approaches that can be used to facilitate the creation, management, and publication of business objects from existing WCMSs, highlighting their strengths and weaknesses. It then describes the implementation of the replication approach, which leverages the power of BEA's XML Beans to offer a convenient, reliable, and scalable solution without relying on features unique to any particular WCMS. This approach is realized in the form of a Business Object Publishing Service that was originally developed for Documentum and WebLogic but can be easily extended to work with different WCMSs and Web application platforms.

Defining Highly Structured Content

Structured content is one of those concepts that is used every day but lacks a concise and universally accepted definition. Searching for such a definition on the Web does not produce any meaningful results; surprisingly neither does searching for it on the sites of four leading CMS vendors.

CMS View

Most CMS vendors try to define structured content by juxtaposing it against unstructured content, or plain files containing text, images, and so on, with no clearly defined and universally accessible metadata. In contrast, they see structured content as the very same files tagged with metadata, such as Author, Topic, Geography, Content Type, etc. Most enterprise-grade CMS packages allow information architects to define rich systems of content types, each with its own set of attributes, as well as workflows for capturing, maintaining, and managing this metadata.

Ultimately CMS publishes all the content along with available metadata to a portal, which for the purpose of this article I will define as a Web application that presents dynamically published content to the users. Portals use metadata to classify, link, find, personalize, and deliver tagged content at the right time to the right audience.

Not Structured Enough

From the portal's point of view the situation is somewhat different: the structured content differs greatly according to how it is represented and used when rendering pages. It makes sense to distinguish between semi-structured content, or files with certain associated, loosely typed metadata, and highly structured content, or business objects, such as Navigation Nodes, Products, Special Offers, etc., with a number of specific and strongly typed attributes. The former is in essence what is considered structured content in the CMS world.

The main difference between these types is that in the case of semi-structured content metadata simply adorns it – it might have an effect on when certain items are found or, perhaps, how they are rendered. If some or even all of the meta-attributes are missing, or have invalid values, the content is still valid, it just might not show up in all the right places. While business objects must have all of their attributes present and often have stringent constraints on their values (e.g., Coupon.Discount must be a positive number between 1 and 99, SpecialOffer.EndDate must be a valid date in the future, etc.). Also, business objects can relate to other business objects – e.g,. Coupon extends SpecialOffer, which in turn can feature Products, which belong to a Brand, etc. Some attributes can be complex types or dependent objects, e.g., Recipe includes a list of RecipeIngredients, which consists of Product, Quantity, and UnitOfMeasure. When presenting such objects through a portal it is essential to be able to traverse these relationships and maintain their integrity as dependents, and parents are published or unpublished.

In other words, highly structured content represents entities that will become objects in the application code and one or more tables in the database schema. It is important to stress that the business objects discussed in this article are strictly content – they can only be created and modified by the content providers and are read-only for all portal users. The metaphor of publishing does not apply to user-modifiable objects, such as User Profiles, Shopping Carts, Blogs, etc. These types of objects cannot be considered content and are outside of the scope of this article.

Examples of Business Objects

This section illustrates the concept of highly structured content and its role in the portal applications, with two examples based on recent implementations.

FOOD & NUTRITION PORTAL

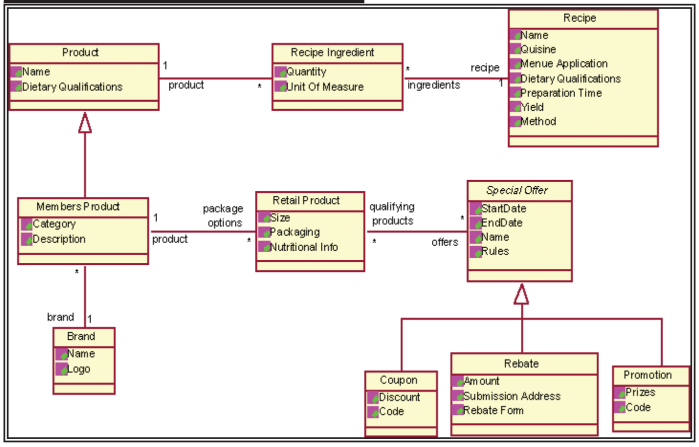
The portal is targeted to food enthusiasts. It features a wide range of content about food, various cuisines, and related matters. It receives content from a broad base of contributors as well as member food manufacturers, which use it to promote their products. It uses a sophisticated CMS implementation to manage all this content and coordinate distributed submission and publishing. In addition to semi-structured content, such as articles, ads, features, campaigns, etc., food manufacturers require capability to publish a number of business objects. This includes their products, brands, and packaging options, which can be added to the product catalog; recipes featuring these products, which go into the recipe box; special offers promoting these products, etc. A simplified domain object model is shown in Figure 1.

The system supports various modes of browsing and searching the product catalog, recipe box, and special offers section. It has rich navigation capabilities between recipes and featured products, and between offers and qualifying products. Business object display pages don't simply present the information, but process it in a number of ways, including:

- Building and validating inter-object traversal links
- Providing users with unit of measure conversion capability when displaying recipes
- Allowing users to calculate yield and price for a recipe
- Allowing users to calculate dollar savings on coupon pages

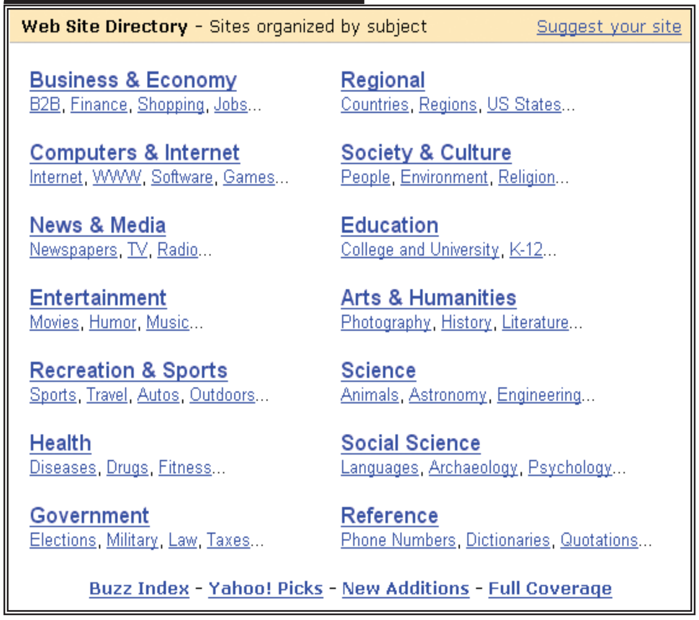
When publishing, updating, and retiring business objects, the system, in addition to following the required workflows, should val-

FIGURE 1



Business objects from the Food Portal

FIGURE 2


Web Site Directory - Sites organized by subject Suggest your site

Business & Economy
B2B, Finance, Shopping, Jobs...

Regional
Countries, Regions, US States...

Computers & Internet
Internet, WWW, Software, Games...

Society & Culture
People, Environment, Religion...

News & Media
Newspapers, TV, Radio...

Education
College and University, K-12...

Entertainment
Movies, Humor, Music...

Arts & Humanities
Photography, History, Literature...

Recreation & Sports
Sports, Travel, Autos, Outdoors...

Science
Animals, Astronomy, Engineering...

Health
Diseases, Drugs, Fitness...

Social Science
Languages, Archaeology, Psychology...

Government
Elections, Military, Law, Taxes...

Reference
Phone Numbers, Dictionaries, Quotations...

Buzz Index - Yahoo! Picks - New Additions - Full Coverage

Yahoo Web Site Directory

idate and maintain complex referential integrity rules, such as:

- Do not publish a Special Offer if it refers to unpublished Product.
- Do not publish a Recipe if it refers to unpublished Retail Product.
- Do not unpublish a Members Product if it has any published Retail Products.
- Do not unpublish a Product if it is used as an ingredient in any Recipe.
- When republishing any Business Object, preserve all relationships that target it.

The system should delegate to members complete control over their content, including business objects. In addition, it should be able to target relevant semi-structured content, such as ads, to the pages that display Business Objects.

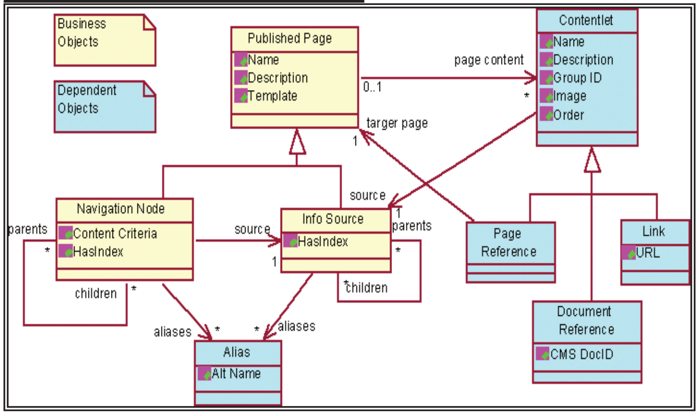
Global Knowledge Repository

This repository is an information portal that presents systematized information in health and life sciences. It accumulates a broad range of content from diverse contributors, or information sources, and presents it to audiences ranging from the general public to researchers and health professionals. The accuracy and relevance of all content is of utmost importance. Information sources are administrative units within a number of organizations that are deemed authorities over certain information fields within the portal's information domain. They should be able to publish and manage content within that field, as well as determine the part of portal's taxonomy that covers the field. All information on the site should be attributed and traceable to a specific source. The administration over both content and navigation should support hierarchical delegation, e.g., if a Cancer Research Institute has authority over all cancer-related pages and content, it should be able to delegate authority over everything specific to lung cancer to its department, specializing in that area.

The portal is organized along the taxonomy, similar to the one found in Yahoo's Web Site Directory shown in Figure 2. This taxonomy was implemented as a multihierarchy of Navigation Nodes. To facilitate navigation, the nodes should be traversable to their immediate children and parents (breadcrumbs), and should be able to build indexes of all descendants. All information on the site was tied back to its source in the form of a link to the Info Source page. We maintained these pages by the respective administrative entities, and formed another traversable hierarchy.

All informational pages on the site, including navigation nodes,

FIGURE 3



Business objects from the Global Knowledge Repository

information sources, and stand-alone pages were publishable by the contributors without any involvement of developers or site administrators. These pages had featured content published in the form of various types of contentlets, and in the case of navigational nodes content criteria, allowing the portal to bind them at run time to the published semi-structured content based on its metadata (see Figure 3).

Publishing Challenges

If a portal uses highly structured content it requires a publishing mechanism that has the degree of sophistication that modern CMS packages provide to publishers of semi-structured content. In particular it should allow you to:

- Define content states and publishing workflows
- Provide versioning mechanism with rollback support
- Provide a robust and flexible security model, with role-based ACLs and delegation capabilities
- Validate publishable content and overall content integrity and provide publishers with accurate and timely feedback
- Refresh the entire content if Portal's version gets corrupted

It is highly preferable that publishing solutions for highly structured and semi-structured content have a consistent look and feel, workflows, and security. The most obvious way to achieve this is to publish business objects through the same CMS package as the rest of the site's content.

This is not the only option; in fact the Food & Nutrition Portal described here was initially implemented with a custom Web-based business object publishing system. Implementation of the publishing part took more than half of the overall portal development time and resources and resulted in a system that had less flexibility in workflows and authorization than any CMS product on the market. In addition, the publishing system had to be changed every time there was a change to any of the business objects. And finally, the content providers had to use completely different mechanisms for publishing business objects and the rest of the content.

In the Global Knowledge Repository business object publishing was implemented through the same Documentum CMS as the rest of the portal's content. This allowed content publishers to leverage the full power of Documentum and enjoy a unified environment. Portal developers spent less than 10% of the overall development time on the business object publishing, despite the fact that the full domain object model was more complex than that of the Food & Nutrition Portal.

I believe this proves that the best way to publish highly structured content is through a CMS package. The remainder of this article will concentrate on this solution.

Requirements

Let's establish a set of core requirements for an optimal business object publishing solution. Naturally such requirements depend on the nature of the portal applications that will make use of the published objects; however, experience shows that many key requirements are applicable to the majority of cases.

1. ***Fast access:*** Portal applications should be able to read published objects on the fly, while rendering pages for the end users.
2. ***Transactionality:*** Objects should be published, updated, or unpublished in transactional manner maintaining consistency at all times.
3. ***Referenceable:*** It should be possible to reference published objects from other entities.

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.

4. ***Verifiable:*** Publishing solution should be able to validate objects and maintain referential integrity as it publishes, updates, or unpublishes them.
5. ***Reliable:*** There should be a mechanism to notify object publishers if an operation failed or was rejected.

CMS Capabilities

The solution was originally developed as a workaround for Documentum's inability to publish highly structured content in a way that satisfied the above requirements. However, we believe that the problem is universal and the approach outlined below can be utilized with most commercial CMS packages. Based on an analysis of a number of CMS packages, including Documentum, Open Text's Livelink, and Interwoven, we have identified the following assumptions about common CMS capabilities and limitations in the area of business object publishing.

1. CMS packages support authoring of highly structured content, usually in the form of XML documents that conform to a specified DTD or XSD Schema. With minimal package-specific development a set of input forms and validation rules can be implemented for each business object class.
2. CMS packages support creating and maintaining links between documents. Linking objects can take into account their state, i.e., rules can permit linking only to objects that are in a published state.
3. CMS packages cannot publish highly structured content directly into a relational database. Of the products we have surveyed only Interwoven had a capability to map and deploy XML data directly into a database schema. However, according to their documentation this capability was intended for synchronizing the production database with content created during the development process. Thus it is unlikely to be suitable for ongoing management of highly structured content.
4. CMS packages can publish metadata associated with structured content directly into a relational database. Usually the package dictates the schema for the metadata, which is often volatile, weakly typed, and highly denormalized.

Approaches

Facing the problem of publishing highly structured content from a CMS package, we considered a number of approaches. Each had strengths and weaknesses, and they differed widely in the amount of work required to implement them. In this section we will outline the four most viable approaches along with their pros and cons (see Table 1).

RELATIONAL APPROACH

The first thing to consider is publishing business objects directly into a database, where they would be immediately accessible by the portal application. This seems by far the most natural approach, satisfying all the requirements and requiring no custom development. Unfortunately, it did not work in Documentum, which as subsequent research has shown is a common case for most CMS packages.

METADATA APPROACH

Alternatively, business objects could be published in the same way as semi-structured content with all the attributes stored as metadata. Technically, it would mean that objects of all types would be stored in multiple rows in the same metadata tables. Although it might be possible to define some recursive views that would present the data in the desirable format, it might not work for more complex schemas and is likely to be resource intensive. In any case, reading business objects from name-value format requires additional computation, including attribute and constraint validation, especially when dealing with inter-object relationships. Additionally, a well-defined rejection mechanism would be required to deal with dangling links and malformed objects, since metadata representation would not allow you to enforce such constraints on the database level. To further complicate things, most standard object persistence mechanisms and tools (such as CMP, DAO generators, etc.) would not work with objects stored as metadata, and a custom solution would be required. Finally, adding new object types or changing existing ones would require changing the CMS meta-model. In most CMS packages this is a time-consuming and error-prone operation (e.g., in Documentum any change to the metadata schema causes flushing of Web Cache and republishing of the entire content). On the positive side this approach still allows transactional access, and if certain strict naming conventions are followed, it might be possible to develop a reflection-based universal DAO, to automate development of the persistence layer.

XML APPROACH

Each business object could also be published as an XML document, with the attribute values encoded as XML tags. This makes validation relatively easy, although a custom-built rejection mechanism would have to be implemented. However, parsing XML into objects would be much more resource intensive than reading from a database; and more importantly, this extra load would occur in the application server at the time of rendering client pages. This approach also lacks transactionality.

REPLICATION APPROACH

This is a hybrid approach. First, business objects are published via either the Metadata or XML Approach; then a staging process reads them, performs the necessary validation, and places them into the appropriate object tables in the application-defined schema. The same process should monitor unpublishing activity and remove deleted records from object tables. This ensures that the data conversion overhead is incurred only once per lifetime of an object; however, it might result in some data consistency issues. The staging process would have to keep a transaction log to know which records have already been migrated, deleted, etc.

TABLE 1

APPROACH	PROS	CONS
Relational	1. Meets all the requirements 2. Requires least development	1. Not supported by most CMS packages
Metadata	1. Meets requirements II, III and V 2. Supported out of the box in most CMS	1. Inconsistent with requirement I 2. Does not address requirement IV 3. Requires significant development 4. Makes it difficult to implement
XML	1. Meets requirement IV 2. Supported out of the box in all CMS 3. Requires little development	1. Inconsistent with requirements I and II 2. Does not address requirements III and V
Replication	1. Meets all the requirements 2. Can work with any CMS 3. Can publish into the application's native database schema	1. Requires significant development 2. Could result in a time lag between object publication from CMS and their availability to the portal.

Business object publishing approaches

Business Object Publishing Service

Based on this analysis, the Replication Approach is the best solution for performance-sensitive applications and CMS that do not support publishing directly into relational databases. This section describes its implementation in the form of Business Object Publishing Service, which supported authoring of business objects in Documentum and their publication into a portal application running on WebLogic Application Server. This service was successfully used in the implementation of a knowledge portal similar to the one described previously.

Architecture

A number of application-specific business objects were identified during domain analysis and captured in the form of UML models. These models were used to derive the database schema for object persistence and XML schemas (XSD files), describing presentation of publishable objects in the form of XML documents. These XSD files were then imported into Documentum and used to define object input forms. They were also compiled into XML Beans using BEA's WebLogic Workshop 8.1.

A special content type, Business Object, was defined in Documentum, with subtypes for every specific object class. After a new instance of business object was authored in Documentum, and went through all the necessary steps defined in the associated workflow, it was published into Web Cache. There it was picked up by the Publishing Service, which determined the appropriate publisher and invoked the required action. Publisher then checked the integrity and other object-specific business constraints and updated the appropriate portal's database tables or rejected invalid objects back to Documentum. The publishing service used similar steps for publishing updates and unpublishing objects. All publishing transactions were recorded in a transaction log, which was used to reconcile published objects with XML documents. The same log was used by the portal's caching services to determine when to refresh objects from the database.

The publishing service was implemented as a J2EE application that could either be run manually by the content administrators via a Web interface, or deployed as a service to facilitate near real-time publishing.

Implementation

The service was developed as an active object, which implemented interface Runnable, and encapsulated a daemon thread. Since the rest of the project used Struts, it was configured as a plugin for automatic life cycle management, when run as a service. Service object had a number of control parameters, such as timeouts, priority, etc., which could be set from a configuration file or from the Publisher Console. Some object publishing operations are non-idempotent (i.e., they cannot be performed more than once without causing harm). The easiest way to prevent such problems is to ensure that only a single instance of the service runs at any time. If that is not feasible due to load balancing or high availability considerations, additional safeguards are required to ensure that every instance is able to detect duplicate operations, and each non-idempotent operation is performed only once. This could be achieved through mutual exclusion mechanisms, such as database locking, or through JNDI.

Every time the service runs it executes three queries against Web Cache and the transaction log to determine lists of new publications, updates, and deletions. The entries are then forwarded to the appropriate operations of corresponding publishers. Publishers are implemented as stateless session beans to provide automatic transaction handling during validation and database updates. Each publisher handles objects of a specific type. The publisher type is determined from the configuration file based on the content sub-type of the XML document published from Documentum. This mechanism allows you to extend the service to handle new object types without any modification to the service code itself. All that is required is to compile the XSD schema into a new XML Bean, write a new publisher, and register them with the service.

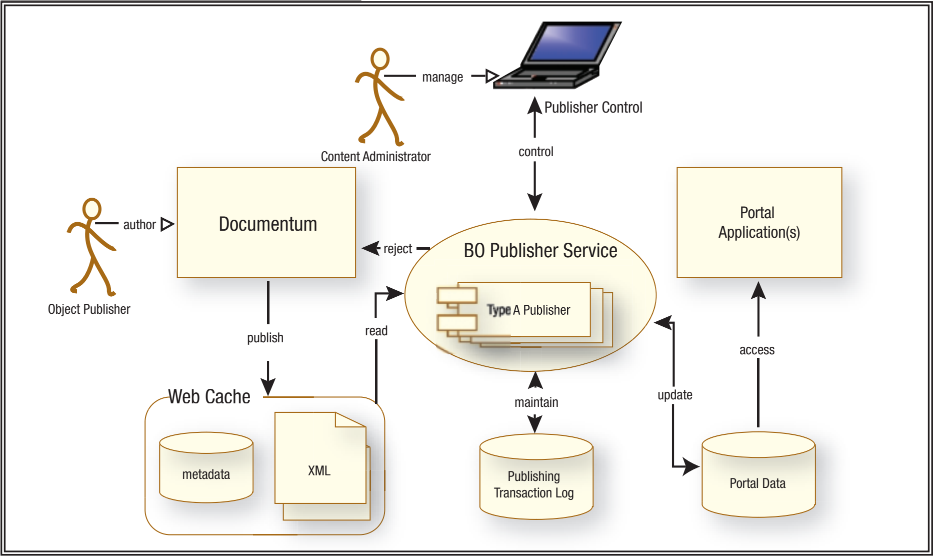
Integration with WebLogic and Documentum

Some additional steps were required to integrate the Object Publishing Service with Documentum.

1. Documentum uses character strings as document IDs. These strings are not suitable as application Object IDs (OIDs), so Publishing Service was responsible for translating between the two types using data from the transaction log.
2. Documentum's Web Cache schema is highly denormalized, can change frequently, and its keys are not unique. We found that defining filtering views and running the Publishing Service against them helps to insulate against many unnecessary complexities.
3. Sometimes Documentum, rather than publishing a new version of a document, unpublishes it altogether and republishes the new version later. To preserve interobject relationships in such cases original OIDs for unpublished objects were kept in the transaction log so they could be recovered if the object was republished later.
4. Out of the box Documentum does not have a mechanism to report publication failures back into the workflow, so all failed transactions were logged into the database and e-mailed to the object publisher.

–continued on page 28

FIGURE 4



High-level architecture of Business Object Publishing Service

Portal Tips and Tricks

EASY FIXES TO IMPROVE YOUR PORTLETS

BY **KUNAL MITTAL**

AUTHOR BIO

Kunal Mittal is a WebLogic architect and consultant for implementation and strategy of Web services and service-oriented architectures. He has coauthored and contributed to several books on Java, WebLogic, and Web services. Kunal has worked on numerous projects using several BEA products. His projects have ranged in verticals such as finance, real-estate, supply chain, broadband, entertainment, and ISVs in the Web services space.

CONTACT...

kunal@kunalmittal.com

WebLogic Portal 8.1 Service Pack 2 has been out for several months. By the time this article is published, Service Pack 3 may also be out. Having worked on a couple of WebLogic Portal projects with this version, I have come across several small and large issues.

This article will provide several tips and tricks to solve these problems, with appropriate code snippets to help Portal developers. Please note that several of these snippets can be found on the BEA newsgroup and/or as part of the BEA WebLogic Portal samples that come with the WebLogic Platform download. However, I have taken most of those snippets and modified them to suit the use cases that I will apply them to in this article. So let's jump right in…

Tip 1: Log in to Your Portal

The BEA samples kit provides several login examples. There are a few things to bear in mind when choosing the login type that you want: Do you want to support automatic login using cookies, login using a controller or a backing file, etc.? One thing to remember is that you always want the portal to refresh after login, so that entitlements can kick in. The two samples worth looking at are the AutoLoginBacking.java and the LoginBacking.java file under the tutorial portal directory. I like using a backing file for the login process so that I can perform an automatic login, and other login checks regardless of the exact page URL that the user bookmarks. Listing 1 shows AutoLoginBacking.java. I have taken the original AutoLoginBacking.java provided by BEA and merged it with the LoginBacking.java. I had some issues with deleting the cookies on logout. You will see a complete working implementation in this example (the code for this article is online at www.sys-con.com/weblogic/sourcec.cfm).

Tip 2: Working with Portlet Maximize and Minimize

You will often feel the need to control the maximizing and minimizing behavior of your portlets. I'll show you a single snippet of code that can be tweaked for almost all scenarios where you need to do this. Let's look at the code and then I will talk about when and where to use this.

The first thing you need to do is get a PortletBackingContext:

```
PortletBackingContext context =
PortletBackingContext.getPortletBackingContext(request);
```

On the context object you can do things like change the mode (View Mode or Edit Mode), or change the state (Normal, Minimized, or Maximized). You will need code such as

```
context.setupModeChangeEvent(WindowCapabilities.VIEW.getName());
context.setupStateChangeEvent(WindowCapabilities.MAXIMIZED.getName());
```

You can also hide the buttons programmatically. For example, to hide the maximize button, you can use the following code:

```
context.setCapabilityVisible(WindowCapabilities.MAXIMIZED, getName(),
false);
```

The lines of code shown above are all you need to work with the state, mode, and visibility of your portlets. I have used them from backing files as well as controllers. For example, consider the following use cases.

1. You have four pages, each with some portlets. Let's say the user is on Page 1, and they maximize a Portlet. Then they go to Page 2. When they come back to Page 1, by default the Portlet that was maximized will remain maximized. If you want it to be restored, you will need to use a backing file at the page or Portlet level.
2. You have enabled the EDIT button on the Portlet title bar. If you want the edit page to always show in a maximized view, again you will need a backing file. In this case, you will need to check the mode and if the mode is EDIT, you will need to maximize the portlet. Similarly, if the mode is VIEW, you will need to restore the portlet to its original state.
3. You don't want to show the maximize button to the user, but you still support use case 2 above. In this case, you cannot turn off the maximize capability, but you can hide the icon using the code above.
4. You can also use this code in the beforeAction() and afterAction() methods of your controller to control the behavior of your portlets.

Tip 3: Refresh Action for Controllers

In Service Pack 2, a patch was released that forced the controller to re-execute each time the page was refreshed. Without this patch, a controller would not get executed on a refresh (patch number CR129301 from BEA). I recommend getting this patch, as you will almost surely need it. Hopefully this will be part of Service Pack 3.

Tip 4: IconUrl in JSR 168 Portlets

You cannot use the iconUrl property in WebLogic Workshop to set a graphic in the titlebar for a JSR 168 portlet. You need to do it by editing the weblogic-portlet.xml file. See the code snippet below.

```
<portlet>
     <portlet-name>myPortlet</portlet-name>
    <supports>
         <mime-type>text/html</mime-type>
        <titlebar-presentation>
                <id>jsr-titlebar</id>
               <icon-url>window-icon.gif</icon-url>
       </titlebar-presentation>
   </portlet>
```

Tip 5: DotPortal Versus Streaming Mode

The basic difference between these modes is whether you read the portal definitions from the XML files that are bundled in the EAR or whether you read them from the database. It is a good idea to deploy in streaming mode so that you can define entitlements for your portlets. Also, in streaming mode you have dynamic control of the layout of the portlets, look and feel, and other such user attributes.

From a development standpoint, the real tip here is to keep all your URLs relative. In streaming mode all the URLs will change so you need to watch out for that. Here is a code snippet you can use to check whether you are in the DotPortal or Streaming mode.

```
<%@ page import="com.bea.netuix.servlets.manager.AppContext"%>

        if (session.getAttribute("DOT_PORTAL") == null)
        {
            AppContext appContext = AppContext.getAppContext(request);
            if (appContext.isDotPortal())
            {
                session.setAttribute("DOT_PORTAL", new Boolean(true));
            }
            else
            {
                session.setAttribute("DOT_PORTAL", new Boolean(false));
            }
    }
```

Summary

This article provides some simple tips and tricks for common issues that I faced when building a portal application. Many of these are related to patches and bugs that should be a part of Service Pack 3. Others are just simple tweaks based on your specific use cases.

A Real-World Business Process Model

PART I

USING THE WEBLOGIC PLATFORM FOR ORDER MANAGEMENT

BY **ANJALI ANAGOL-SUBBARAO**

AUTHOR BIO

Anjali Anagol-Subbarao works in HP's IT organization as an IT architect. She has 12 years of experience, the last 5 in Web services. Her book on J2EE Web services on BEA WebLogic will be published this year.

CONTACT...

anjali.anagol-subbarao@hp.com

When processes are modeled well and don't change, existing IT systems work well. However, real business changes all the time and processes are becoming more complex especially as the Internet is able to easily link internal and external systems.

Business process management (BPM) can help in managing this complex and ever changing process. The concept of manipulating data, which IT systems do efficiently today, can be extended to business processes. Businesses can use BPM systems, using workflow-type techniques, to control existing applications, Web services, and human processes together, or to construct or deconstruct processes or sub-processes.

Using business process modeling, we can model and also measure the effectiveness of existing business processes. This helps us understand how the business currently operates and allows the business to identify the improvements needed. It also gives us the ability to test the suggested improvements before implementing them to see if they will provide the desired results. Figure 1 shows how you can implement BPM in your business. First you analyze and define the process model. These business processes can be built and deployed through a process engine. Then, you can monitor the business process.

In this series we will see how WebLogic Integration provides a BPM solution. We will go through a real world example involving order management to show how Integration can be used to model, execute, and monitor a business process

PD4J and WebLogic Integration

WebLogic Integration provides a robust and integrated BPM solution. Process modeling in WebLogic Workshop is based on Process Definition for Java (PD4J). Different organizations are working towards different specifications in BPM – the Java Community Process (JCP) is working on Process Definition for Java (PD4J), OASIS on Business Process Execution Language for Web Service (WSBPEL or BPEL). PD4J has been proposed in JSR 207 and builds on the Java Language Metadata technology in JSR 175. The latter standard provides an easy-to-use syntax for describing a business process at the source-code level for the J2EE platform. The intention of JSR 207 is to explore and standardize the relationship between process languages like BPEL and the Java language and J2EE platform.

As a major next step to defining this Java process standard, BEA and IBM have closely collaborated to create a new specification called BPELJ, which has also been submitted to the JSR 207 working group. BPELJ is a combination of BPEL and Java. It allows these two languages to be used together to build complete business process applications. By enabling BPEL and Java to work together, BPELJ leverages what each does best. BPELJ is implemented via extensions to the BPEL language; therefore any BPEL process can be executed through BPELJ. By standardizing these extensions, business processes will be truly portable and interoperable across the J2EE platform.

BEA is the lead for JSR 207 and a coauthor of both BPEL and BPELJ. In addition to BPEL capabilities, BEA will also provide full support for BPELJ in the next major release of WebLogic Integration. It will provide automatic and seamless migration experience from processes written in PD4J to BPELJ.

Key Components to Build BPM in WebLogic Integration

WebLogic Integration builds the BPM solution in three stages:

1. Modeling of the process
2. Execution or automation of the process, which is basically the build and deploy process
3. Analysis of the process which includes monitoring of the process

For process modeling, WebLogic Integration uses WebLogic Workshop, which has graphical tools to build, view, and change business process models in both the design and source views in the WebLogic Workshop environment. Features of the modeling tool provide synchronous and asynchronous Web services, branching, nesting, looping, parallelism, grouping, and exception handling.

The business process also integrates with the WebLogic Control Framework, which supports database, file, messaging, service broker, and human interaction. The Control Framework, which WebLogic Workshop provides, is a way to encapsulate business logic and to access enterprise resources such as databases, message queues, and timers, and to expose them as Web services. Using the Control Framework you can combine data and business logic from different resources and use them in a business process.

For XML-to-Java transformation, you'll use XMLBeans. XMLBeans are technology used throughout WebLogic Workshop; it provides very easy translation between XML data and Java types. XMLBeans have many advantages,without losing access to original XML structure, they provide a Java object-based view of XML data. The XML's integrity as a document is not lost with XMLBeans. They handle the entire XML document instance as a whole instead of taking XML apart like other APIs. We will see in Workshop how when an xsd file is imported to a schemas folder, XMLBeans are created and are available to your business process.

For XML2XML, Java2XML, and non-XML2non-XML mapping you'll use the XQuery Transformation Mapper. XQuery Maps provide a way for you to reshape the XML messages that Web services send or receive.

WebLogic Integration takes care of process automation for the process designed in WebLogic Workshop. Process building is accomplished through auto-generation of J2EE code. Source code is created automatically when you design business processes using the graphical tools in WebLogic Workshop, which stores the process definition in accordance with PD4J. It is called a JPD (Process Definition for Java) file. You can also edit Java code in the source view. WebLogic Server is used to build and deploy the business process. WebLogic Workshop has a test browser that helps to test the process. You can execute stateful processes, which are processes that maintain state information and stateless processes, which are processes without state information as well as synchronous and asynchronous processes. A WSDL can be generated for a business process and therefore enable it to be invoked as a Web service.

Process analysis provides ongoing monitoring and collects statistical data in real time. This is very important in BPM. Service-level agreement (SLA) status monitoring and generating reports of historical process information are provided by the BPM tool. You can use the WebLogic Integration console to monitor the process.

A Real World Example

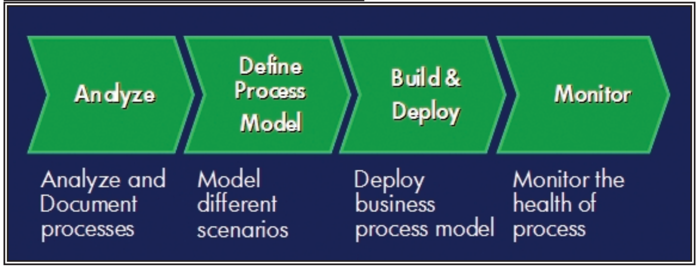
WebLogic Workshop can be used to model a real-world business process. It has a design view that has process nodes and control palettes. Through these you can add Web services, client requests, decision nodes, and Java controls to mimic a real world process. As you add these components, Workshop generates, in a separate tab, the source code and annotations to reflect the process logic using the PD4J specification. After modeling this business process, you can test it by executing it within Workshop.

In this series of articles, we will look at a real business process – a Change Order Request within the Order Management business process for a vendor. We will see how we can build this "real world business process" in WebLogic Workshop using the graphical interface.

The scenario we are looking at here is a PC reseller selling configure-to-order PCs. A consumer who buys through this reseller wants to upgrade the disk drive in the PC he has ordered. This results in a change request made by the reseller on the manufacturer. The reseller places a request to change the product configuration information, which is to upgrade a disk drive for the PC of a previously placed order.

The order change process has many process steps and decision points. The first process step is the receipt of the order change request from the reseller in the form of an XML. This XML uses the RosettaNet Partner Interface Processes (PIPs), for an order change. The RosettaNet PIP is part of the RosettaNet Implementation Framework and is a standardized electronic business transaction between trading partners. Consistency in transaction formats is extremely important to decrease time to implement transactions between trading partners, therefore each PIP comes with a message guideline and XML document-type definition (DTD).

FIGURE 1

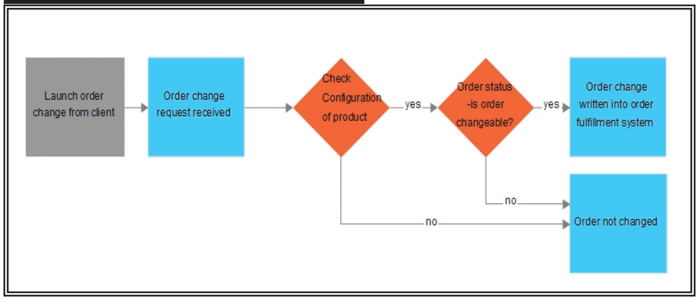

Analyze
Define Process Model
Build & Deploy
Monitor
Analyze and Document processes
Model different scenarios
Deploy business process model
Monitor the health of process

Implementing BPM

As the Change Order has a new configuration for an upgraded disk drive, we need to check whether this configuration is valid for the PC. The business process uses a Web service to check the validity of the configuration. If this upgraded disk drive cannot be ordered with this PC, the configuration is not valid and the process ends. If the configuration is valid, then the business process goes to the next step.

The next step is to check with the factory floor the status of the original order. This will help to track where the PC is in the assembly process. The status will let you know if it is at a point in the assembly where a disk drive can be added, which means the order is changeable, or it may be at the shipping dock, where the order is not changeable.The status is stored in a database. In the process step a database control needs to be invoked to check the Order Status. If this order cannot be "changed" the process ends. If the order can be "changed" the change is performed in the ERP-based "system" or it can be written out to a file that can be uploaded to an ERP system. Figure 2 shows the change order request process steps. We will use these steps in WebLogic Workshop to create the business process.

FIGURE 3


Launch order change from client
Order change request received
Check Configuration of product
yes
Order status -is order changeable?
yes
Order change written into order fulfillment system
no
no
Order not changed

Change order request process

Steps to Create a Business Process

First, you need to create an application. We can call this application orderChange. In this application we need to create a new process called orderChange.jpd. To start the process we need to add a ClientRequest received. Next we will add the Web service validate. Then we need to decide whether or not the configuration is valid; therefore, we will add a decision node. To check the order status, add the order status database control. Again, we have to decide if the order status allows us to proceed, therefore, we will add another decision node. The last part of the process is to either write this change to a file control, which can then be uploaded to an SAP order fulfillment system, or written directly to SAP. The last step is to conclude the process.

Summary

We will go through the details of each step in the next articles in this series. The second article will cover the creation of the process, the client request and addition of the Web service. The third article will look at the addition of the decision points and database control. The fourth article will cover writing to a file control and the ending of process. We will see how we can test the business process in the test browser and how the process can be monitored in the last article.

References

- *BPEL* and *BPELJ:* http://dev2dev.bea.com
- Keen, Peter. "Business Process Management". www.infocono-my.com

PUBLISHING BUSINESS OBJECTS IN PORTALS

–continued from page 23

WebLogic did not require many customizations; we just had to ensure that we always had just a single instance of the service running when deploying application in a clustered environment.

When the publishing service is used in a deployment architecture that includes multiple environments, e.g., QA, staging, production, etc., the best approach is to deploy a single service instance into each of the environments and then rely on the multi-environment publishing capabilities of the CMS package to ensure controlled propagation of published objects.

Conclusion

This article identified business objects, which encapsulate application data, as a distinct category of highly structured publishable content, which has enough similarities with other content types to warrant the use of a Web Content Management System, yet are different to such an extent that they are not supported by the majority of existing WCMSs. Having outlined several approaches to solving this problem, it describes the architecture and implementation of a Business Object Publishing Service that was successfully used on a project similar to the Global Knowledge Repository. That required less than 10% of the effort that went into development of the display capabilities of that portal. In contrast, on a project similar to the Food & Nutrition Portal a decision was made to implement a custom Object publishing solution. This required the same amount of effort that went into development of the portal's display capabilities, and still resulted in a solution which was less efficient and flexible then any commercial CMS package.

Although the Replication Approach and Object Publishing Service were initially developed for Documentum they do not depend on any features not commonly found in other WCMSs, so the same techniques can be used with any commercial CMS, allowing companies to leverage their investment while significantly reducing development of complex Web applications.

References

- "Vitrage: a Framework for Compartment-Oriented Application Development." Whitepaper by Roundarch, Inc. describes a framework developed for rendering Business Objects in Portal Applications. www.roundarch.com/features/vitrage.html
- Siemens, George. "Content Management: Our Organized Future." A good introduction to CMS concepts and features. www.elearnspace.org/Articles/contentmanagement.htm
- "Automatic Content Categorization and Tagging with Content Intelligence Services." A whitepaper from Documentum provides an idea of how CMS vendors typically view structured content. www.documentum.com/products/collateral/platform/wp_tech_cis.pdf
- Bau, David. "XMLBeans." BEA Senior Staff Engineer and XML Beans Architect. http://dev2dev.bea.com/technologies/xmlbeans/articles/Bau.jsp

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

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

AppDev™ VXML for BEA's WebLogic™ Workshop

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

For additional information:

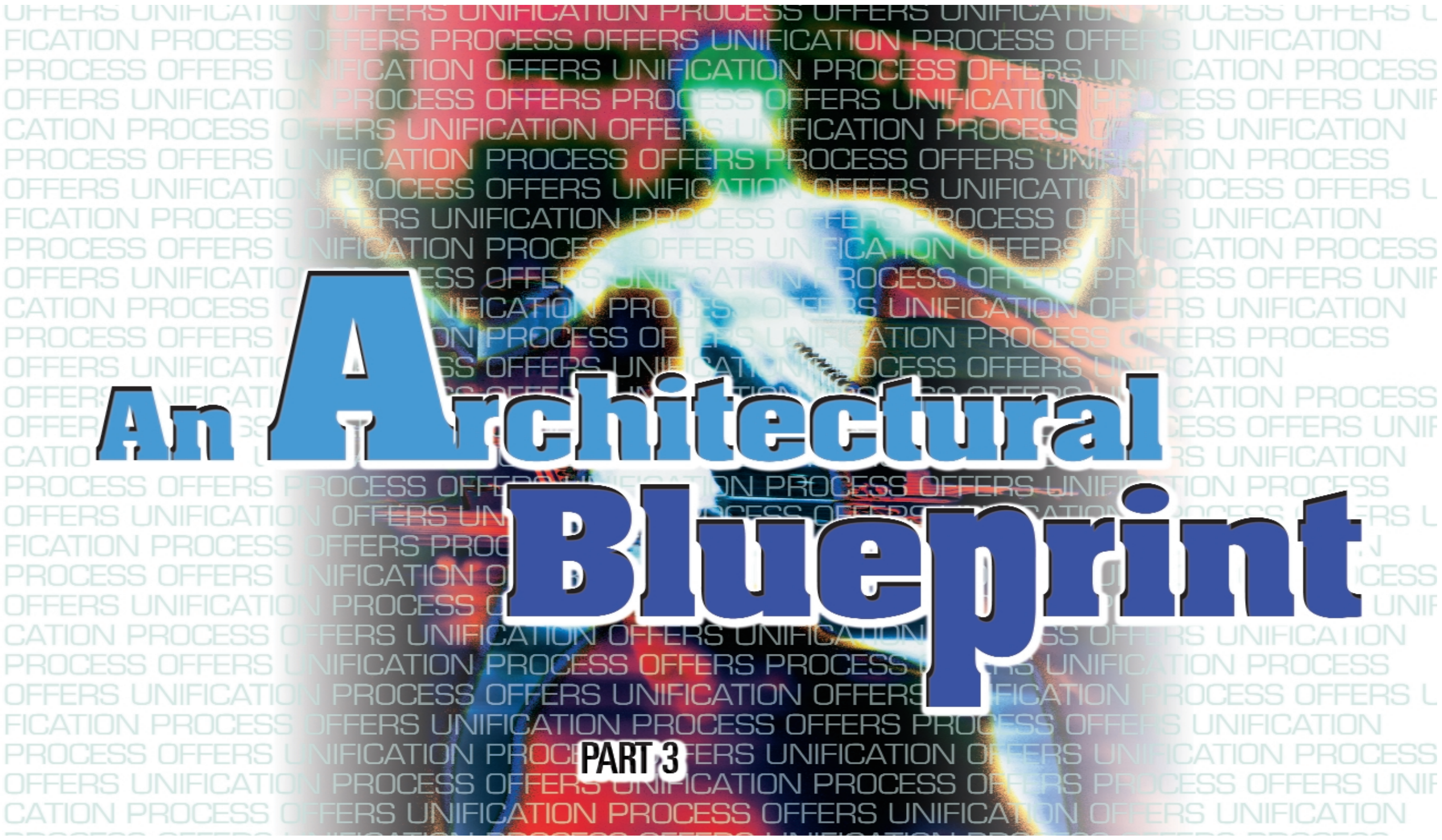
SandCherry, Inc.
1715 38th Street
Boulder, CO 80301

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

Info@sandcherry.com
www.sandcherry.com

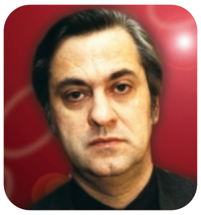
Download 30 Day Free Trial
www.sandcherry.com

SandCherry™



An Architectural Blueprint

PROCESS OFFERS UNIFICATION



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

As we've discussed over the past few issues, JTA-style transactions provide a way for multiple data updates to be tied together so application logic can operate safely in the assumption that it will succeed or fail consistently, even in the face of technical failures along the road.

In this last article in this series, I will apply some of the BPM techniques covered in the first two articles (*WLDJ*, Vol. 3, issues 4–5) to a business case, in order to design a robust SOA-based solution. In particular, I will (1) define the business case as a set of high-level policies, and (2) apply business process improvement techniques to derive an efficient business process model that encapsulates the real-world problem. The business model will contain business events that start processes, the high-level process threads and how they decompose to low-level enterprise business services. Then I will discuss various architectural design options towards building a proof of concept (POC). But first, I'll talk about BEA WebLogic Workshop 8.1's new IDE programming paradigm and in particular how the process integration framework provides a solid programming environment for building SOA Web services integration applications.

From Database Client/Server Tools to a Distributed Application Integration Platform

BEA's new IDE provides a powerful, seamless, and all-encompassing tool for designing and deploying J2EE applications. In the same fashion that MS Access and PowerBuilder propagated the development of mission-critical applications using the client/server model in the early '90s without requiring deep expertise in RDBMS or TCP-IP communication, Workshop hides the OO and J2EE complexities by using simple concepts such as controls, events, methods, and properties without compromising the expressiveness and quality of applications. Visual controls act as the interface layer between developer and application infrastructure. Workshop transforms visual controls into J2EE components and the developer can always write Java code when necessary.

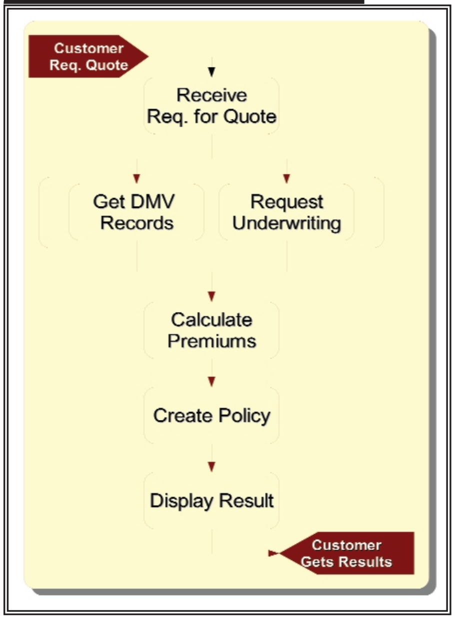
Developers can build complex Web-based user interfaces using Java Page Flow (JPF). Workshop leverages Struts, an open source implementation of Model-View-Controller architecture for binding data elements with interface widgets. This is now called the MVC1 architectural pattern; as opposed to MVC2, which more closely resembles the initial Smalltalk pattern that decouples data presentation and user control from business model. It is worth mentioning here that Barracuda is another open source implementation, not implemented out of the box, of the MVC1 pattern that comes closer to the simplicity of event-based UI that the Visual Studio .NET IDE provides, but I will leave this comparison for another time! Web services connections are built as Java Web Services (JWS) files – standard Java files with a simple Javadoc annotation for accessing Web services functionality. The developer can worry about SOAP protocol, WSDL, and XML bindings only if he wants to!

A Java Control is the single most important concept in WebLogic Workshop 8.1. Pretty much any programmatic resource can be handled as a visual Java Control. A legacy system or a Web service, a custom Java object or an EJB, an external resource or a workflow implementing a piece of proprietary business logic, an asynchronous or two-way messaging state full or stateless conversation, can all be represented as controls. Developers interact with controls by handling events and setting properties, and extend the built-in controls by writing custom controls.

The Business Case

Our fictitious, but virtually real, Car Insurance Company (CIC) is faced with competitive pressures, lower margins, and

FIGURE 1



Process map

a higher cost of operating and maintaining legacy systems. Some of the problems are customers demanding shorter decision-making time; client applications for insurance are misplaced when moving from front office to middle office; the complexity in getting information from the legacy systems and vast data repositories; requests for credit information and motor vehicle records is time consuming and hard to track; and finally, underwriters come and go and keep changing communication requirements.

In light of these challenges, the board of directors voted on a number of new policies: (1) establish an e-commerce presence by allowing potential clients to get a 60-second response via the Web; (2) streamline the credit check process and take advantage of the new secure Web services interface that the Department of Motor Vehicle (DMV) just deployed; and (3) become a leader within the underwriting space by establishing a B2B collaborative community. If the project accomplishes its objectives, the board will then make decisions on retiring some of the aging legacy systems. But in this phase, the board strongly recommends optimal utilization of any legacy systems and existing data.

Business Rules

Users visiting the Web site enter their car type, year, miles per year; name, sex, age, and Social Security number. For simplicity, car types are compact, sedan, and SUV with base premiums of 200, 250, and 400 respectively. The mileage per year produces a mileage factor (MF): if less than 8,000 the MF is 1; if between 8,000 and 20,000, the MF is 2; if greater than 20,000, the MF is 3. The gender and age also produce a sex-age factor (SAF): the SAF is 4 for males under 24, 3 for females under 25, 2 for males older than 24 and females between 24 and 34, and 1 for females older than 34. The insurance company uses the DMV's state-of-the-art secure Web service to pull the driver's record. Using proprietary methods based on data collected throughout many years, a legacy back-office application scores the DMV record and produces a driver profile factor (DPF), a number between one and four. The final premium calculation is equal to base premium + weighted factor * 50, where the weighted factor (WF) is equal to (3*MF+4*SAF+4*DPF)/10.

If the WF is less than 20, the agent can produce a quote and send it back to the user; otherwise, the user request for a quote needs to be sent for underwriting.

The underwriters want to use the same premium calculation techniques if possible. The user can then decide if he wants to apply for the coverage, in which case he submits payment information, etc. The rest of the process is out of scope for this POC.

Clearly all of these rules are volatile and subject to constant changes and adjustments. Most of them are currently implemented in various systems with varying degree of complexity, from spreadsheets to COBOL books. Human intervention and coordination are also required to complete a request for quote.

Developing a Business Model

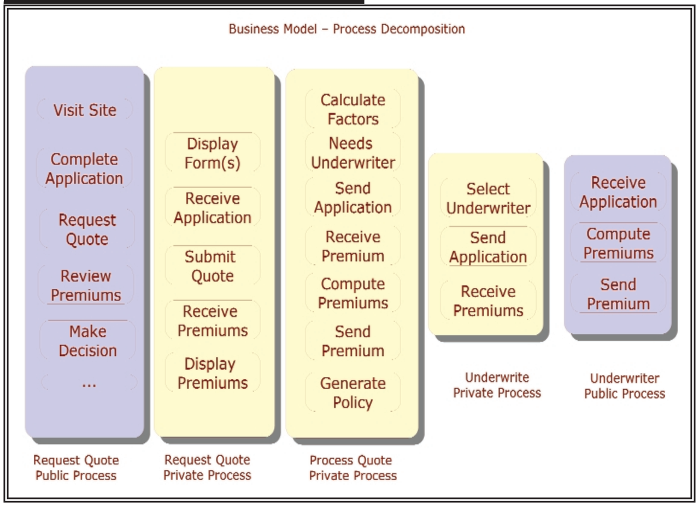
Think processes not functions. A process starts with a predefined business event and is what a business does – not how – to produce results consistent with the business policies and corporate vision set by the executive office and mission statement. An event can trigger one or more process threads and can be internal or external to the business. Money-making events are usually external and associated in some fashion with the supply chain. Events can be actual or temporal. Results may be felt inside or outside the firewall, the imaginary protective boundaries of the enterprise. A good and consistent naming convention is always important: a verb plus an object for a business process, i.e., Get DMV records and Calculate Premiums; an actor plus a verb plus an object for an actual event, i.e., Customer Requests Quote; and a time snapshot definition for a temporal event, i.e., Sixty Second Response or 9:00 AM Consolidation.

Clearly, one of the merits of a BPM design and development approach is the visual workflow aspect. However, WebLogic Workshop has no facilities for developing swim-lane type diagrams. The processing design phase has to happen on paper, or using Visio-like drawing tools or BPM tools with design capabilities. The latter approach would create two substantial issues:

1. Burden of importing/exporting processes
2. Traceability issues when making changes. Workshop does not even allow a simple view of two or more processes side-by-side in the design area of the IDE. Such a simple solution would make an interactive design session much easier.

For this article, I used the OpenOffice drawing package for the first one of two levels of process decomposition. As the processes gain more depth and get closer to EBS (Enterprise Business

FIGURE 2



Processes in "swim lane" diagram

Services), I'll start designing in Workshop Studio. I have the ability to group together activities in a box that I can name descriptively and collapse at will. The first step is to develop a high-level process map (see Figure 1) containing actors or participants initiating business events and coarse grain decomposition to sub-processes and activities. This diagram serves a good purpose. It is a baseline communication and serves as a straw man; however, it has a weak point. It misses the fact that processes among two parties have two views. Processes are just like objects that have internal and private implementation and an external public interface. Of course it is all relative and depends on which side of the process you are. In this case, the classification is from the CIC point of view. Figure 2 contains the processes distributed in "swim lanes." There are five processes from left to right: request quote public process, request quote private process, process Quote private process, underwrite private process, and Underwrite public process.

Design Decisions and Architectural Choices

You would think that building UI should be well understood and a simple matter by now. Think again. There are as many options as problem types and each option has its own challenges. For now, I will discuss two issues: event communication and building widgets.

Event Communication

User events need to either start a process and then interact with it or perhaps attach to a long-running monitoring process that in response spawns an instance of a process thread dedicated to a specific client; something like a TCP/IP server listening to a port for client requests and spawning a new process thread to facilitate the interaction with a new client via a dedicated socket. But how do you do that with a request and reply protocol? The answer is you don't; at least not in a trivial and efficient way.

Business events, the heart of a business process–driven enterprise, are at odds with the Web. No surprise here, HTTP was never meant to be a substitute for a full-blown programming paradigm, just a mere protocol for exchanging information, pictures, text and the likes. So what now? In the case of a stateless interaction no real issues exist. Web services can encapsulate and facilitate communication in a synchronous fashion. In the case of stateful interaction conversational Web services, asynchronous in nature, they can maintain state across multiple calls. The programmer equips a conversational Web service with a callback. When the Web service completes its service, the callback in the client code gets activated. That is a nice clean solution; however, some clients cannot have a callback. For example, a Web page interaction using HTTP cannot have a callback installed. So we are back where we started, about three decades ago: polling. Yes, polling. The Web service has to implement a method something like: areYouDoneYet () returning a Boolean true or false. When the service is done, it updates a state variable from false to true. The client code keeps polling or executing the areYouDoneYet () method until it gets a true. The client can then call a getResults () method of the Web service to get the result.

Building Widgets

At first look, JPF sounds similar to process flow but they have nothing in common other than the word process. JPF manages the navigation amongst http pages. Workshop provides a widget-rich environment for building complex UIs. What is particularly powerful is the use of Form Beans, which manage the binding, storage, intel.com

Can a microscopic piece of technology solve enterprise-sized problems?

88% of the world's servers* are powered by Intel® processors.

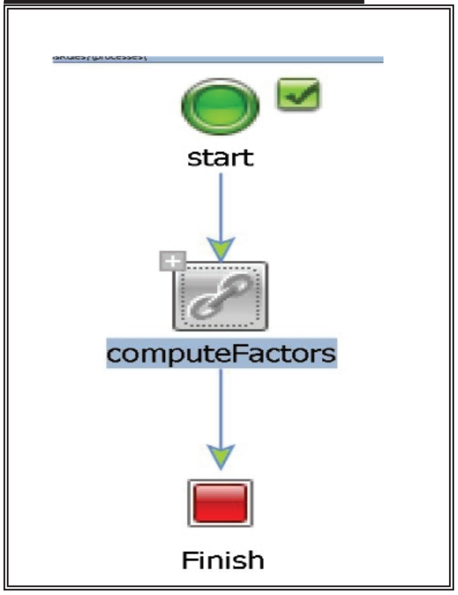
An Intel engineer holds a batch of silicon wafers, each one containing up to hundreds of processors and up to 12 billion transistors.

That's because Intel-based servers deliver higher performance, lower costs and maximum flexibility — which can deliver savings that are anything but microscopic. So when you choose the servers that drive your company look for Intel Inside®.

www.intel.com/ids/bea for technical information

*Based on total server shipments 1996-CY Q1 2002 according to IDC's Worldwide Quarterly Server Tracker, May 2002.

FIGURE 3



Timer to monitor response

and validation of form data. JPF communicates with the rest of the system via a conversational Web service. The Web service uses a process control to execute the private process quote process. Since the JPF cannot have a callback mechanism, the Web service implements the polling technique described earlier. In addition, the Web service implants the sixty seconds timer to monitor response from the rest of the systems (see Figure 3).

Worklist Client API

BEA WebLogic Integrator provides a programmatic interface for applications to interact with the process container. As part of the process, an activity may require human interaction. Worklist is a simple HTTP-based testing tool that enables human interaction. It is a work queue. Clearly, a better design is to define your own client interface and hook up to the process engine using the Worklist Client API, but the details of building such an interface would be out of the scope of this article. Worklist provides a simple solution for testing the validity of the Underwriting process. Gregor Hohpe and Bobby Woolf in *Enterprise Integration Patterns* provide a whole chapter describing a Loan Broker example that is very similar to the concept of Underwriters B2B ecosystem. In particular, they describe three different models: Fixed addressing, Distribution, and Auction. Using a combination of Web services, a list of URIs, and messaging channels, a fully extendable solution can be build.

BPM as Programming Tool

In my previous articles, I said that most process engines are implemented as a kind of state machine, and that Petri nets are Turing complete. You can then prove that any program written in any language can also be implemented in BPM. Clearly, some languages are better than other languages for some specific problem types. The BPM language is the natural choice for implementing the integration layer or volatile and fast changing aspects of the problem.

Business rules make a perfect candidate for a BPM implementation. To make a clarification here, I don't mean more business rules in the sense of Artificial Intelligence expert systems with inference engines and the like. I mean business rules in the sense of what I stated previously with factor calculations and premiums. However, I must admit it would be an interesting exercise to build a generalized expert shell using BPM workflows. To illustrate the point, I have implemented the business rules of our business as a process: computeFactors. I implement a parallel split for the three different factors. In reality, parallel branches are logically concurrent. WebLogic Server serializes the execution of the branches. When grid computing and high-performance computing become a common practice, true parallelization may be just an additional option. For now, such an option is available only through grid-enabling software from specialized companies like Powerllel, or Distributed Resource Manager (DRM) systems providers like Platform and Sun. The third branch executes the DMV Web service that returns the DMV factor. Here is how: assuming the Web service is available on the Internet, I point the browser to it and download the WSDL file on my local disk. Then, in Workshop I browse to the WSDL location, right-click, and select Generate JCX from WSDL. The resulting JCX file is a Web service control that I can now bind to a process element.

Conclusion

WebLogic Platform provides a robust product stack for deploying SOA-based solutions. It offers a highly productive environment that enables the development, deployment, and operations of enterprise-wide distributed solutions. The visual paradigm does a nice job in hiding J2EE and OO complexities. I find the concept of turning almost every resource available into a control – a computing component with a consistent interface – its most powerful feature. In a very consistent and repeatable way, controls harness the power of object reuse and make solutions easily extendible in response to the ever changing agile enterprise.

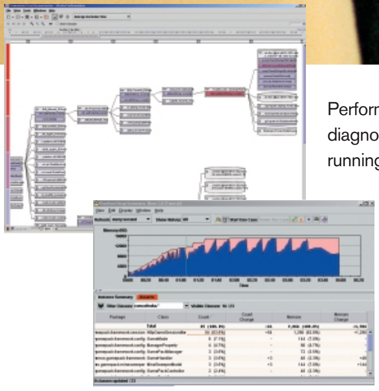
Process is the unifying construct that is applicable across all development phases. Process bridges the modeling tower of Babel. Business and technology can now talk the same language. At the macro level, process is the natural choice for implementing integration tasks. At the micro level, process is the perfect place to implement volatile aspects of a solution, like business rules. Indeed, BPM is a first-class computing citizen.

The WebLogic Platform implements a comprehensive BPM environment. However, BEA does not provide a critical component of a BPM solution: a pure business process design and modeling environment. In the same way that few business analysts were object modelers, few business analysts will become fluent in BEA's business integration environment. But considering the activities in the modeling space with the acquisitions of Rational Rose and TogetherSoft, Microsoft's renewed interest in UML modeling, and the continued success of the pure-play BPM players, BEA better be working hard on the next all-encompassing release of BPM for Business Analysts, which will probably be UML compliant!

Finally, service-oriented architecture has emerged as the unambiguous fabric of modern distributed systems. The universal acceptance of Web services as the connectivity technology makes collaboration more attainable than ever. Clearly there is more to come in this area. After all, we still accumulate questions faster than we can give answers to existing ones. Some of the questions are in key areas such as security and networking. The very foundation of our technology, TCP/IP, is in doubt, as it is described by Paul A. Watson, in his recent paper "Slipping In The Window: TCP Reset Attacks." And if there is any doubt, we have lost the war on spam, at least for now. Inevitably we can stand still; we never have. The time of IT commoditization has yet to come. I predict that in the next two to three years new technologies and innovations that better answer some of these questions will deliver yet another generation in distributed systems, with more mature, dynamic, and adaptable architectural paradigms.

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

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 trouble-shooting 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.**

QUEST SOFTWARE®

Handling Large Database **Result Sets**

THE ONE-WRITE APPROACH IN LARGE-SCALE APPLICATIONS

BY **THOMAS KRUSE & ALOIS LECHICKI**

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

Many J2EE applications include the requirement to display lists of database records as a result of the user search. Such search operations can involve processing large data sets on the server side. For example, think of the typical search in an online product catalog containing several thousand products.

Applications processing large result sets can face scalability and performance issues when the number of concurrent clients increases. When a query is executed according to user-supplied search criteria there may be no way to anticipate how much data will be returned. The result set might include a huge amount of data, so you cannot blindly send the query result back to the client without causing undue performance problems.

For managing large database result sets the design pattern Value List Handler is frequently used. The J2EE Blueprint recommends using this pattern to control the search, cache the results, and provide the results to the client. In this solution, when the client wants to search data the ValueListHandler is called which intercepts the client search request and returns data iteratively in chunks of pages.

The ValueListHandler provides query execution functionality and results caching. It implements an Iterator interface (see Figure 1).

The ValueList is a collection class that holds a set of Transfer Objects representing database records. In this design pattern the handler does not return the entire result list but only a subset of data. In a production application the ValueListHandler is usually hidden behind a façade according to the Business Service Façade pattern.

The J2EE Blueprint describes some general implementation strategies for the Value List Handler pattern. For example, it recommends using a Data Access Object (DAO) instead of entity beans if performance is an issue. However, when implementing a particular application a developer needs to consider further design trade-offs in order to meet specific functional and performance requirements.

In this article we present an implementation of the Value List Handler pattern proven in a large-scale WebLogic project. You can apply this approach when you develop an application that:

- Is to be used by several thousand concurrent users
- Has to display some part of the user search results in chunks of pages
- Assumes that the paging is consistent across requests, i.e. the underlying data is not expected to change when scrolling the result set

A Case Study

A travel retailer decided to develop a Web-based system offering travel agents direct access to information and reservation systems of a wide range of travel providers. The system was to be used by thousands of travel agents worldwide. Agents could select from over 100 million package holidays and over 5 million last-minute travel

AUTHOR BIO

Thomas Kruse has been working in the IT industry for 13 years. He has experience with J2EE technology and has completed two-large scale J2EE applications during this time. His professional background ranges from developer to system designer to system architect.

Alois Lechicki is a principal consultant at Softlab, Munich. He works with customers to help design and implement J2EE architectures. Alois has more than 15 years of experience in software engineering.

CONTACT...

thomas.kruse@kruse-it.de
alois.lechicki@softlab.de

bargains, updated daily. The system enabled users to search for a desired holiday package, check room and flight availability, then book and track customer reservations.

The site had to support the following use cases:

- A travel agent enters a customer's requirements, such as holiday destination, hotel facilities, room furnishings, and flight time.
- The site generates travel options based on requirements provided by the agent.
- The customer selects a travel option and the agent makes reservations.
- The agent completes the transaction by supplying a credit card number.

The second use case required running complex queries in the database containing package holidays. Searching in this database could produce result sets of extremely different lengths. Depending on the holiday destination, hotel category, room furnishings, and flight time, one could find a small or a huge number of options. However, the travel agent was not expected to browse through long result lists. She or he was rather expected to repeat the search operation after narrowing the search criteria.

Consequently, one requirement was to sort the result list according to a customer's preferences and then to cut it at a given position. The retailer estimated that it was enough to show only the first few hundred holiday packages.

Another requirement was to avoid a heavy load on online reservation systems when checking for availability. Thus the result list should not be checked at once but rather in chunks of, say, 20 maximum package holidays. Consequently, one design choice was to implement a page-by-page iterator based on the Value List Handler pattern. In this case, we could assume that the paging was consistent across requests. In other words, the underlying data in the database was not expected to change when scrolling the returned result set.

Since the application was required to have a sophisticated GUI with a complex workflow, another design choice was to develop a rich client using the applet technology. As a result, the client-side approach was used to manage session information (a session identifier) across multiple requests.

The server-side application design followed the principle of a service-oriented architecture (SOA). Groups of methods are clustered together in the services layer and exposed to the client via a service object – an EJB session bean. For example, the session bean HolidayPackageServices provides methods for searching and booking holiday packages.

For transferring data between components the design pattern Data Transfer Object (DTO) was applied. However, in order to minimize the impact of frequent changes in the domain layer, we decided to use dynamic data transfer objects (DDTO; see sidebar).

The system was implemented using BEA WebLogic Server 8.1 SP1 and an Oracle database for storing data related to package holidays. The application accessed online reservation systems of tour operators via SOAP over HTTP(S). The client (applet) accessed the server-side services via the Web services infrastructure provided by WebLogic Server. This was easy to develop using sever tools that generate Web services interfaces and require little or no coding. Figure 2 shows an overview of the application architecture.

Problems

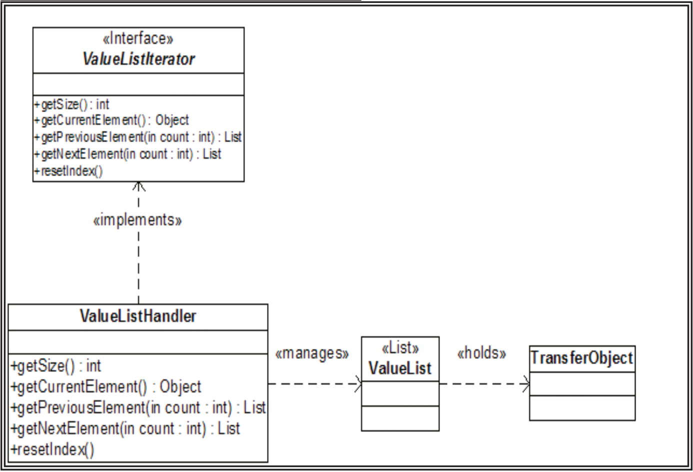
Since the application had to manage several thousand concurrent clients, we decided to use the stateless approach. However, search operations in the database turned out to be too resource intensive to be repeated for each paging request. Thus one design decision was to store search results in the database whenever a query result was too large to be returned to the client in one chunk.

A holiday package was represented by a graph of objects, called package items. In the first implementation packages were stored in the database as full-blown object graphs using an object-to-relational mapping (O/R-mapping). Unfortunately, load and performance tests soon revealed a poor response time primarily due to extensive interactions between the application server and the database server. Each "store query result" request led to several hundred "writes" of the package item records.

Improved Solution

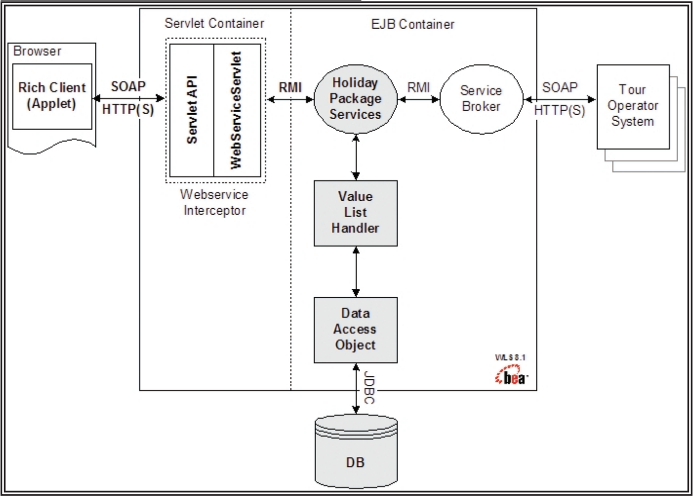
Due to the performance issues described above, the original implementation of the Value List Handler pattern had to be changed. The key idea was to minimize the number of "write" operations. We started by storing all items of a holiday package in one Oracle BLOB field. The next improvement was to write the result set in chunks of pages, each chunk as one BLOB. Finally we decided to store the entire result set (i.e., the actually fetched part

FIGURE 1


«Interface»
ValueListIterator
+getSize() : int
+getCurrentElement() : Object
+getPreviousElement(in count : int) : List
+getNextElement(in count : int) : List
+resetIndex()
«implements»
ValueListHandler
+getSize() : int
+getCurrentElement() : Object
+getPreviousElement(in count : int) : List
+getNextElement(in count : int) : List
+resetIndex()
«manages»
«List»
ValueList
«holds»
TransferObject

Class diagram for the Value List Handler pattern

FIGURE 2



Application architecture overview

of it) in a BLOB field in a single "write" operation.

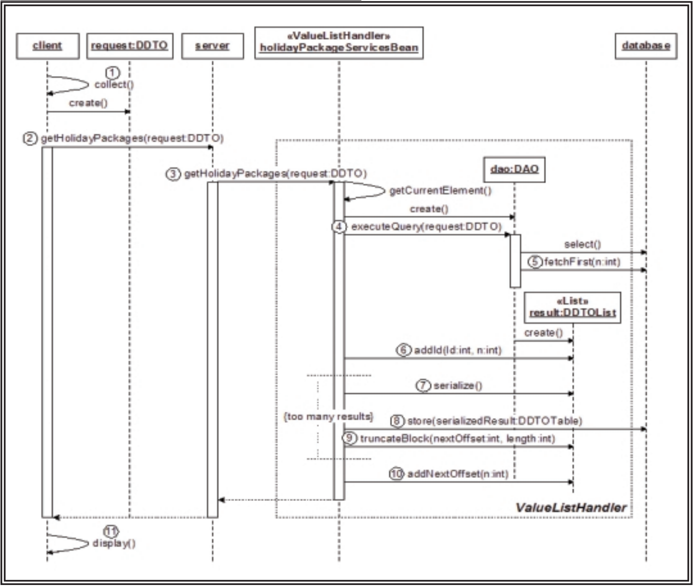
After executing a query according to user-supplied search criteria, the first N records were fetched (if available) and stored in a list of dynamic data transfer objects (DDTOList). The number N was configurable and could be changed at run time. When the actual number of returned records was greater than the page size, the DDTOList was serialized and stored as a single record in the HOLIDAY-_PACKAGE_BAG table (see Table 1). Of course, a clean-up mechanism was implemented as well.

TABLE 1

COLUMN NAME	DATA TYPE	REMARK
ID	NUMBER(16)	Primary key
RESULT_SET	BLOB	Serialized DDTOList object containing query result
CREATION_TIMESTAMP	DATE	Timestamp of record creation

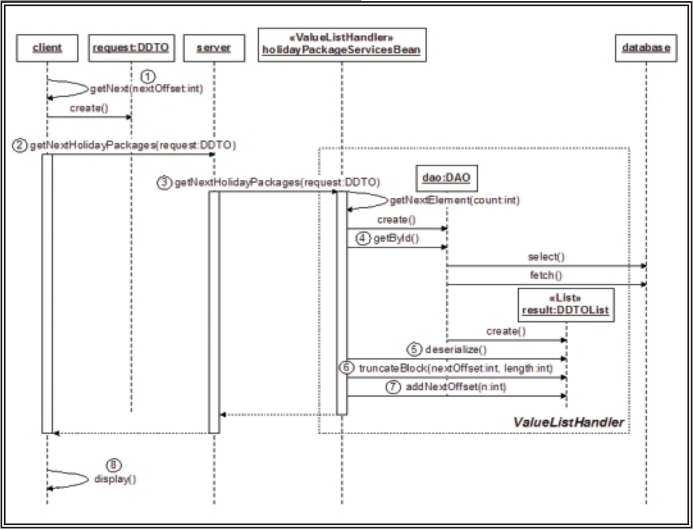
Database table for storing result sets

FIGURE 3



Sequence diagram: get the first chunk of holiday packages

FIGURE 4


client
request:DDTO
server
«ValueListHandler» holidayPackageServicesBean
database
1 getNext(nextOffset:int)
create()
2 getNextHolidayPackages(request:DDTO)
3 getNextHolidayPackages(request:DDTO)
dao:DAO
getNextElement(count:int)
create()
4 getById()
select()
fetch()
«List» result:DDTOList
create()
5 deserialize()
6 truncateBlock(nextOffset:int, length:int)
7 addNextOffset(n:int)
ValueListHandler
8 display()

Sequence diagram: get next block of holiday packages

In this solution, the record containing the result set was read each time the client required the next page. It means that more data was actually read than needed. However, we found that in this trade-off "read" operations were less expensive than "write" operations.

Finally, further load and performance tests confirmed that the "one-write" solution improved the performance of the system by an order of magnitude compared to the initial implementation.

Under the Hood

Let's look a little more closely at the implementation – and consider a typical use case.

When selling holidays the travel agent asks the customer about his wishes for a vacation destination, hotel facilities, required room furnishings, and so on. After supplying all mandatory information the travel agent sends a request to the system and waits for the first page of holiday packages. He then presents the result list to the customer, asking him to make a choice. However, it may happen that none of the offers on the list appeals to the customer. In this case the travel agent either asks the system for the next page or he changes the search criteria and repeats the search request.

From a technical viewpoint, this use case can be split into two sequence diagrams, which are discussed in the next sections.

Scenario 1: Get First Block of Holiday Packages

This scenario (see Figure 3) shows the flow of control when the customer's request is defined and the search operation is started.

- ***Message (1):*** At first, all information required for the getHolidayPackages request is collected by the applet. The client-side workflow supports the user in entering only a valid set of search criteria. This data is stored in a dynamic data transfer object (DDTO).
- ***Message (2):*** After filling the DDTO, the applet calls the Web services method getHoliday-Packages using a local stub object. The stub serializes the DDTO to a SOAP message and sends it via HTTPS to the server.
- ***Message (3):*** When the server receives the SOAP message it deserializes its payload and puts the data into a DDTO. Then it calls the method getHolidayPackages of the stateless session bean HolidayPackageServicesBean (for better readability we have skipped aspects of session tracking and security and authorization issues).
- ***Message (4):*** Now a DAO object builds a select statement using the search criteria contained in the DDTO and executes the query. In this process, the result set is sorted according to user preferences and business rules.
- ***Message (5):*** The DAO object gets the first N records from the database (the number N is configurable), if available, and stores them in a DDTOList object. When the DAO completes reading records, it closes the database connection.
- ***Message (6):*** A unique holidayPackageBagId is created and stored in the DDTOList object. This id has to be supplied by the client among other items in subsequent requests to identify the result set.
- ***Messages (7) – (9):*** These messages are sent only if the actual number of retrieved records exceeds the (configurable) page size. The DDTOList is serialized using the standard Java mechanism and stored as one record (as BLOB) in the HOLIDAY_PACKAGE_BAG table. The holidayPackageBagId is

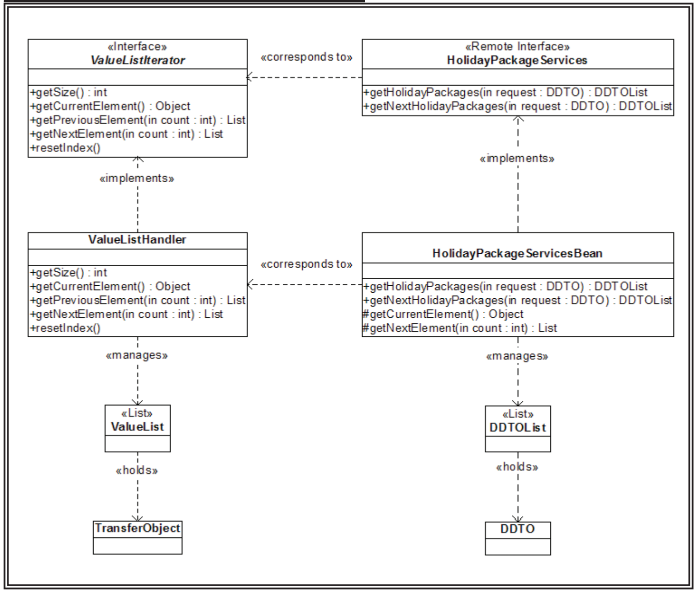
used here as key. After that, the DDTOList is so truncated that it contains only the first chunk of the result list and the length of the original list.

- ***Message (10):*** The position of the next page nextOffset is added to the DDTOList and the list is sent back to the client using the WLS WebServices infrastructure.
- ***Message (11):*** The client receives the response and displays the first page of the result list. The total number of records available on the server is displayed as well.

Scenario 2: Get the Next Block of Holiday Packages

The second scenario (see Figure 4) shows the flow of control when the client asks for the next chunk of the result list. It means that the initial search operation returned a record set having more records than one page could contain.

FIGURE 5


«Interface»
ValueListIterator
+getSize() : int
+getCurrentElement() : Object
+getPreviousElement(in count : int) : List
+getNextElement(in count : int) : List
+resetIndex()
«corresponds to»
«Remote Interface»
HolidayPackageServices
+getHolidayPackages(in request : DDTO) : DDTOList
+getNextHolidayPackages(in request : DDTO) : DDTOList
«implements»
«implements»
ValueListHandler
+getSize() : int
+getCurrentElement() : Object
+getPreviousElement(in count : int) : List
+getNextElement(in count : int) : List
+resetIndex()
«corresponds to»
HolidayPackageServicesBean
+getHolidayPackages(in request : DDTO) : DDTOList
+getNextHolidayPackages(in request : DDTO) : DDTOList
#getCurrentElement() : Object
#getNextElement(in count : int) : List
«manages»
«manages»
«List»
ValueList
«List»
DDTOList
«holds»
«holds»
TransferObject
DDTO

Class diagram: the implemented Value List Handler pattern

DYNAMIC DATA TRANSFER OBJECTS (DDTO)

The concept of dynamic data transfer object (DDTO) is based on the Generic Attribute Access pattern. This approach is used to minimize the impact of frequent changes in the domain layer on the rest of the system. A DDTO is a generic data container that can hold an arbitrary set of data. It provides a generic attribute access interface using HashMaps and key-value notation.

DDTOs implemented in the case study could contain both simple data types (String, Integer, etc.) and complex ones (structures). Moreover, nested DDTOs and lists of DDTOs (DDTOList) were supported as well.

The implemented framework provided a type-safe access to the content of a DDTO. It also contained superclasses for DAOs and entity beans implementing completely generic interface methods. Thus DAOs and entity beans could expose a unified interface to their attributes, with almost no extra coding required.

- ***Message (1):*** The applet creates a dynamic data transfer object (DDTO) and puts both holidayPackageBagId and nextOffset into it. Note that it is up to the client to manage session information.
- ***Message (2):*** The applet calls the Web services method getNextHolidayPackages using a local stub object. The stub serializes the DDTO to a SOAP message and sends it via HTTPS to the server.
- ***Message (3):*** The server deserializes the message payload and puts the data into a DDTO. It then calls the method getNextHolidayPackages of the stateless session bean HolidayPackageServicesBean.
- ***Message (4):*** A DAO object is used to retrieve the record corresponding to holidayPackageBagId from the table HOLIDAY_PACKAGE_BAG.
- ***Messages (5)–(7):*** The BLOB field RESULT_SET is deserialized into a DDTOList object using the standard Java mechanism. Let us recall (see Scenario 1) that the DDTOList contains all holiday packages retrieved in the initial search operation. Now using the parameter nextOffset the next chunk of the result list is created and sent as DDTOList back to the client. Before that, the value of nextOffset is updated so that it points at the subsequent page.
- ***Message (8):*** The client receives the response and displays the next page of the result list.

Back to the Pattern

In the solution presented above the HolidayPackageServices Bean implemented only a part of the interface ValueListIterator. For example, the method getPreviousElement was not needed because the client (applet) cached read records. From the client point of view it was enough to provide only two methods of the ValueListIterator: getHolidayPackages and getNextHoliday Packages. Figure 5 shows the Value List Handler pattern as implemented in the case study in relation to Sun's archetype.

Conclusion

In the case study presented here we used the Value List Handler pattern to provide a Web service for searching holiday packages. In this pattern long result sets are sent to the client iteratively in chunks of pages.

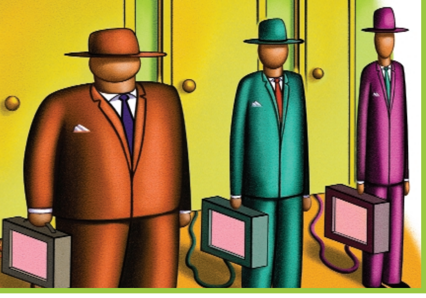
While this approach reduces network overhead, caching of result sets can still be an issue. Stateful session beans are not always an option if you have several thousand concurrent users and the ValueList object has to hold long result lists.

The key design decision in the case study was to use stateless session beans and to store result sets, longer than one page, in the database. The chosen implementation improved the performance of database operations by reducing the number of write operations. This was done by storing result sets in a compact form as Oracle BLOBs.

An additional advantage of this approach was easy deployment of the load balancing mechanism.

References

- *Sun Microsystems:* http://java.sun.com/blueprints/corej2eepatterns/Patterns/ValueListHandler.html
- SOA: webservices.xml.com/pub/a/ws/2003/09/30/soa.html
- Marinescu, Floyd. (2002) *EJB Design Patterns: Advanced Patterns, Processes, and Idioms*. Wiley Computer Publishing.



Notes from a Small Place

CAN YOU CHANGE THE WEATHER BY TALKING TO BUTTERFLIES?

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

As we've discussed over the past few issues, JTA-style transactions provide a way for multiple data updates to be tied together so application logic can operate safely in the assumption that it will succeed or fail consistently, even in the face of technical failures along the road.

I have often mused, in this column and outside, on how bizarre it seems that I have filled over two years worth of space in ***WLDJ*** talking about JTA, a tiny corner of the tiny universe that is J2EE, itself a tiny part of what a development shop needs in order to deliver something of value to its lords and paymasters as fast as possible. Let's face it, when was the last time you saw a requirements document that said "we need 25 entity beans, 7 message-driven beans, and an XML interface, all of which must be transactional, because that will save us 10% per year over 20 years on our operational costs"?

What you are more likely to hear is something like "we need to automate the on-boarding process; every time we hire a new employee, it takes a full half day, say, of an HR person and a senior manager's time to complete the necessary procedures. We need to cut the overhead of recruitment down to no more than one hour: go and do it in six months with a budget of $150k."

It would be easy to leap from that reality to the view that these articles are of no value, that J2EE is of no value, and that the world should be filled with "solution providers" who do not sell or deliver technology, but solutions to business problems. The sspirin for every business headache, just add water!

The Aspirin for Every Business Headache

However, unlike headaches, business problems don't exist in isolation. A single pain point is just that – a point. Lose sight of the bigger picture in your solution (however seductive that solution might seem on paper) and you will build nothing more than a piece of tomorrow's migraine from today's palliative.

These realities have led the industry to a broader perspective as to what an application server should be, and what an infrastructure vendor (BEA still being the only independent one) should be providing. Gartner talks about the application platform suite (APS) as being the next-generation application server, having formulated that an APS can save an organization up to 22% in time to market across the application life cycle and 40% in post-deployment maintenance (a recurring cost), and give 50% better utilization of expensive developers and architects. Not surprising then that they predict that the APS will become the major unit of purchase within the first 10 years of this century.

So what is an APS? Well, if you think about a normal project's requirements – you typically need a front-end interface, some integration to existing business systems incorporating some automation of process flow between them, and somewhere to put custom logic. The "Do It Yourself" brigade might see that as an invitation to buy an app server, a portal UI framework, and an EAI bus and get coding. However, the savings

Gartner identified from the APS approach to the solution flows from the fact that in a true APS Portal, integration and development infrastructures are unified with a single programming model and infrastructure, whereas in the DIY case effort must be spent understanding the infrastructural moving parts in isolation, and all that before the development effort of tying them together (with custom code that you need to maintain and understand) can start. Oh, and after all that the three sub-teams that understand the three component technolologies can start building their pieces of the overall solution that was required before the fooling around started.

You can see from this that the most effective way to provide a solution for a given business problem is to have a team of business-level analysts and developers specify and implement an end-to-end solution using APS technology. I am clearly somewhat partisan in this view (unlike Gartner, and the other analysts, who see it too), but I am proud to work for BEA because we are widely acknowledged in the industry as having a two-year technology lead in this important and growing approach to application development, being the only company that has a production-quality APS product shipping today (and that's said before you start thinking about SOA, SODA, ISE, ESB, and the other technologies that are coming along apace, promising new levels of flexibility to the tired, stovepiped world).

It is against that backdrop that I am currently reading lots of noise about who has the biggest market share in the "J2EE application server wars" – the market share figures are notoriously subjective at the best of times because many vendors don't release audited sales figures at the granularity of individual products. Some analysts say BEA is still in the lead, some say it has dropped to number two, and either way, a good puff of invective on the subject is a great consumer of column inches – there's nothing like a shock headline, whatever the reality. When you add the whole industry's march to the APS approach to the opaque nature of the figures being compared in this debate, what you are left with is the feeling that a change may or may not have taken place in a marketplace, which is rapidly being redefined anyway. Hold the front page! I think it's too early for victory celebration, or obituaries anywhere just yet.

So back to application servers, J2EE, and transactions, not to mention these articles: are they too small to be important? Well, no. The most comprehensive technical solution to any business problem must be built on a solid foundation, so the base-level underpinnings are still important, although hopefully the APS approach will mean fewer people on each project team will have to be intimately familiar with the ins and outs of J2EE. And besides, as chaos theory lovers will know, the wing beats of a butterfly in the rainforest today can cause a storm in the city tomorrow.

But as for the J2EE app server market being the barometer of infrastructure industry success, I don't know anybody who goes about trying to change the weather by talking to butterflies.

Reference

- Gartner Group. (2003). "Application Platform Suite Architectural Costs Analysis".

TROUBLE SEEING BEA PORTAL DATA?

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, IntelliJIDEA, and other Java development environments.

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

[illegible]

Easy Java Portlets

DEVELOP AND DEPLOY BASED ON JSR 168

BY **PRAKASH MALANI**

AUTHOR BIO

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

CONTACT...

pmalani@malani.org

A portlet is a Web component that generates fragments – pieces of markup (e.g., HTML, XML) adhering to certain specifications. Fragments are aggregated to form a complete document.

This article introduces the Java Specification Request (JSR) 168 on Java Portlets. It illustrates the creation of Java Portlets using BEA WebLogic Workshop 8.1 SP2 and the deployment of these portlets on BEA WebLogic Portal 8.1 SP2. I'll look at essential concepts such as portal, desktop, and portlets and describe in detail the various portlet modes and window states. I'll also look at designing, implementing, configuring, and executing portlets with Workshop.

JSR 168 defines the specification for Java Portlets. A portal is a Web application and an aggregation of portlets. A portlet container runs portlets and manages their life cycle. JSR 168 defines the contract between a portlet and portlet container. It does not define the contract between a portlet container and a portal. The implementation of the portal is left up to the portal vendors.

BEA WebLogic Portal

The current version of BEA WebLogic Portal (8.1 SP2) supports different types of portlets: JSP/HTML portlets, Java PageFlow portlets, Struts portlets, and Java portlets. In the future, other portlets, such as Web Services for Remote Portlets (WSRP) will be supported. Our focus will be on Java portlets.

WebLogic Portal provides portal capabilities above and beyond those described in JSR 168 including, but not limited to, the organization of portlets in books and pages; multichannel support; and customization using skins, skeletons, and shells.

In order to follow along here, before proceeding to the next section please complete the following:

- Using the WebLogic Domain Configuration Wizard, create a portal domain (e.g., JSR168PortalDomain).
- Using WebLogic Workshop create a portal application (e.g., JSR168PortalApp) that uses the above-created domain.
- Create a portal Web project (e.g., JSR168Portal-WebProject) inside the portal application.
- Create a WebLogic Portal .portal file (e.g., JSR168.portal) inside the portal Web project.
- Start the server instance.

Creating Your First Java Portlet

The following steps describe creating your first JSR 168 portlet.

- Using WebLogic Workshop, create a new folder for the portlet (e.g., FirstPortlet) inside a portal Web project (e.g., JSR168PortalWeb-Project).
- Create a new portlet by creating the corresponding .portlet file (e.g., First.portlet) using the Wizard inside the new folder.
- Choose Java Portlet as the portlet type.
- Specify the title (e.g., First).
- Specify the definition label (e.g., first).
- Specify the class name (e.g., com.malani.examples.portlets.jsr168.FirstPortlet).

- Open the portal (e.g., JSR168.portal).
- Drag-and-drop the portlet (e.g., First.portlet) onto a page in the portal (e.g,. JSR168.portal).
- Run the .portal file to test it.

Your first JSR 168 portlet is running successfully! But what does the wizard do underneath the covers?

- It creates a WebLogic Workshop and WebLogic Portal–specific .portlet file. The .portlet file forms the contract with the Workshop and WebLogic Portal–specific .portal file.
- The wizard creates a .java file (e.g,. com.malani.examples.-portlets.jsr168.FirstPortlet.java) that is placed in the WEB-INF/src directory.
- The wizard creates a WEB-INF/portlet.xml configuration file and inserts an entry for the portlet into the file. The entry for the portlet looks something like:

```
  <portlet>
    <description>Description goes here</description>
    <portlet-name>first</portlet-name>
    <portlet-class>com.malani.examples.portlets.jsr168.FirstPortlet</port-
let-class>
    <portlet-info>
      <title>First</title>
    </portlet-info>
  </portlet>
```

Java Portlet Class

The Portlet Java file generated by the wizard in this example extends the javax.portlet.GenericPortlet class. The GenericPortlet class implements the javax.portlet.Portlet interface. Figure 1 is a Unified Modeling Language (UML) class diagram depicting these relationships. A portlet can be written by directly implementing the portlet interface. However, GenericPortlet is a more convenient way of creating a portlet. First, let's look at the portlet life cycle, portlet modes, and window states.

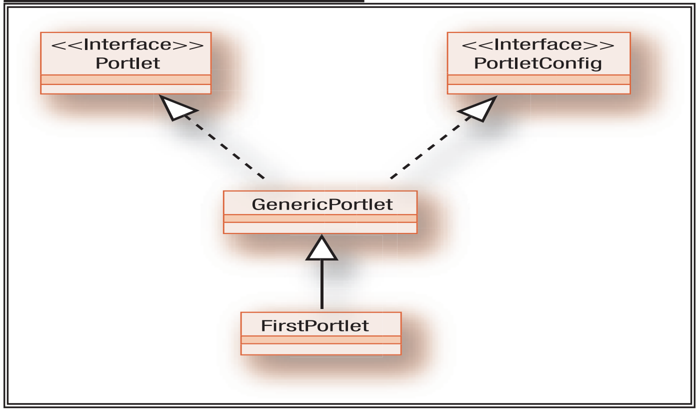
Portlet Life Cycle

To successfully create portlets, you have to follow the portlet life cycle. The methods in the javax.portlet.Portlet interface define that life cycle. The life-cycle methods are init(), render(), processAction(), and destroy(). The init() method is called when an instance of the portlet is deployed. It is used to obtain any needed expensive resources, such as back-end connections, and perform other one-time activities. The destroy() method is used to release those resources when an instance of the portlet is undeployed.

The portlet specification makes a clear distinction between render requests and action requests. Figure 2 depicts a UML class diagram of portlet requests and responses. A render request on the portal page results in the calling of the render() method on each portlet on the displayed page. When a user invokes an action on a particular portlet, typically an HTML form submission, the processAction() method of that portlet is invoked. The render() methods of all portlets on the same page are also invoked. Thus, an action request by the user translates into one invocation of a processAction() method and multiple invocations of render() methods. Figure 3 is a sequence diagram depicting the effect of invoking processAction() method and subsequent invocations of render() methods for the portlets on the same page. For further information, refer to the section on Processing Actions.

There are two overloaded init() methods, one with no-arguments and the other with an instance of javax.portlet.PortletConfig class. *Note:* There is a peculiar caveat about the init(PortletConfig) method. Failure to invoke super.init(aPortletConfig) results in a NullPointerException. The Init portlet in the included source code example illustrates this behavior (the source code is online at www.sys-con.com/weblogic/source.cfm).

FIGURE 1



A UML class diagram depicting relationships between Portlet, GenericPortlet, and FirstPortlet.

Portlet Mode

JSR 168 defines three Portlet Modes: VIEW, EDIT, and HELP. A portlet instance can be in exactly one portlet mode at any time. Other custom portlet modes, such as config and source, are possible. The VIEW mode is the default mode. The portlet specification recommends that the EDIT mode allow the user of the portlet to customize the portlet instance and the HELP mode display the usage information about the portlet. A portlet must support the VIEW mode but the support of the EDIT mode and HELP mode in a portlet is optional. For example, the First portlet example does not support the EDIT mode or HELP mode.

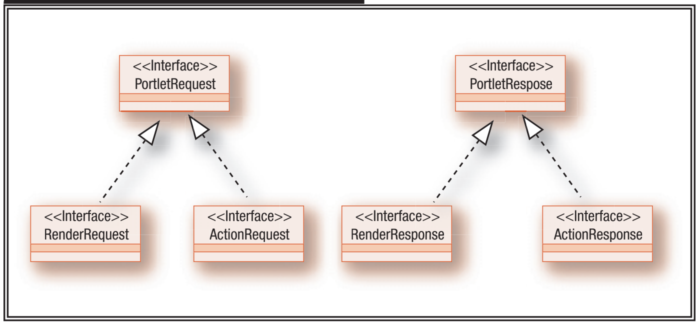
Window State

JSR 168 defines three Window States: NORMAL, MINIMIZED, and MAXIMIZED. The portlet instance can be in exactly one window state at any time. Other custom window states, such as half-page, are also possible. In the NORMAL state, the portlet occupies a small part of the screen real estate. The screen real estate is shared with other portlets. In the MINIMIZED state, the contents of the portlet are hidden. In the MAXIMIZED state, the contents of the portlet occupy most of the screen real estate. Other portlets sharing the same page are hidden in the MAXIMIZED state. For example, the First portlet example supports all three Window States.

GenericPortlet Class

Most of the portlets you create will extend the javax.portlet.GenericPortlet class instead of directly implementing the javax.portlet.Portlet interface. The GenericPortlet class implements the render() method. If the portlet's window state is minimized, then the render() method doesn't do anything. If the portlet's window state is other than minimized, then the render() method sets the title specified in the portlet.xml file and invokes the doDispatch() method. Depending upon the Portlet Mode, the doDispatch() method invokes doView(), doEdit(), and doHelp() methods appropriately. Thus, the GenericPortlet class is more convenient to extend than the Portlet interface is to implement because the GenericPortlet

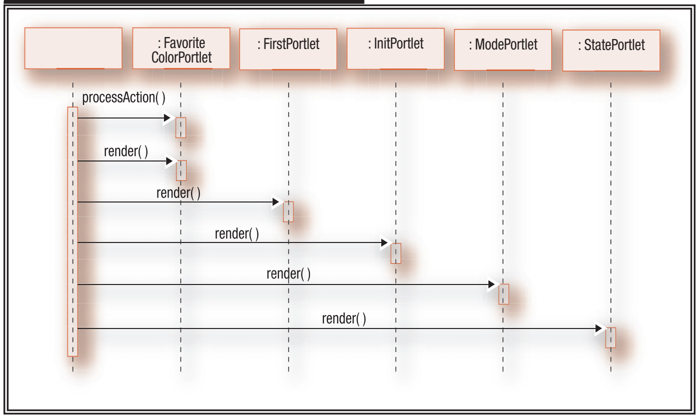
FIGURE 2



<<Interface>> PortletRequest
<<Interface>> PortletResponse
<<Interface>> RenderRequest
<<Interface>> ActionRequest
<<Interface>> RenderResponse
<<Interface>> ActionResponse

A UML class diagram depicting portlet requests and responses

FIGURE 3



A UML sequence diagram depicting an action invocation

class helpfully implements the render() method and provides easy doView(), doEdit(), and doHelp() methods to override.

Consider the First portlet example. The FirstPortlet class extends GenericPortlet. FirstPortlet overrides the doView() method:

```
public void doView(RenderRequest request, RenderResponse response)
  throws PortletException, IOException
{
  response.setContentType("text/html");
  response.getWriter().write("<p>Hello World</p>");
}
```

Note: Calling the getWriter() method before calling setContent-Type() method results in a java.lang.IllegalStateException.

Implementing Portlet Modes

The VIEW mode is mandatory but the EDIT and HELP modes are optional. In order to implement the EDIT and HELP portlet modes, implement the appropriate doEdit() and doHelp() methods in the portlet class. Refer to Mode portlet included in the source code example (source code for this article is online at www.sys-con.com/wldj/sourcec.cfm). In addition, you must configure the modes in the portlet.xml as follows:

```
<supports>
  <mime-type>text/html</mime-type>
  <portlet-mode>edit</portlet-mode>
  <portlet-mode>help</portlet-mode>
</supports>
```

Note: Changing the portlet.xml configuration file, but not implementing the corresponding methods in the portlet class, results in a javax.portlet.PortletException.

Implementing Window States

JSR 168 does not describe a way to disable support for Window States. However, WebLogic Portal implements the disabling of them. To disable a portlet's support for Window states, exclude the Window states in the weblogic-portlet.xml:

```
<portlet>
  <portlet-name>state</portlet-name>
  <supports>
    <mime-type>text/html</mime-type>
    <excluded-window-state>minimized</excluded-window-state>
    <excluded-window-state>maximized</excluded-window-state>
  </supports>
</portlet>
```

Refer to the State portlet included in the source code example.

Including JavaServer Pages (JSPs)

Consider the doView() method of the First portlet. This method gets the instance of the Writer and directly outputs HTML fragments. Outputting direct HTML fragments is usually not recommended for various reasons, such as trying to achieve a separation between Java logic and HTML view presentation. The recommended way is to display the view using a JSP. The methods in the portlet class execute business logic, set the render parameters, and include the JSP. To include a particular JSP, first get the PortletContext. From the PortletContext instance, get an instance of PortletRequestDispatcher by calling the getRequestDispatcher() method. Include the JSP by invoking the include() method. For example:

```
    // execute the necessary logic here...
    PortletRequestDispatcher aDispatcher =
getPortletContext().getRequestDispatcher(
      "/IncludePortlet/includeView.jsp"
    );
    aDispatcher.include(aRequest, aResponse);
```

Note: A portlet may use a PortletRequestDispatcher object only when executing the render() method.

Refer to the Include portlet in the source code. A JSP page, such as includeView.jsp, does not contain root HTML tags such as <html>, <title>, and <body> since the tags are provided by the portal framework. The JSP page contains only the HTML fragments necessary to display the portlet.

Processing Actions

In a standard Web application an HTML form submission results in executing some business logic. The results of business processing are either set in the request or session as attributes and forwarded or included to the next JSP.

In a JSR 168 portlet, what should the action URL for an HTML form be? JSR 168 defines a JSP tag library, known as portlet taglib. The action URL for an HTML form is generated using the actionURL portlet tag. For example (refer to favoriteColorEdit.jsp file):

```
<form action="<portlet:actionURL/>" method="post">
  …
</form>
```

Submitting the HTML form results in invoking the processAction(ActionRequest aRequest, ActionResponse aResponse) method of the portlet. As usual, form parameters can be obtained by invoking getParameter() method of the request object. *Note:* Invoking the action by submitting the form, but not having the processAction() method in the portlet, results in a javax.portlet.PortletException.

The processAction() method sets the values in the response object. Do not use the setAttribute() method of the ActionRequest or ActionResponse object. The values will not be propagated from the processAction() to the render() method and will not be available in the JSP. Instead, use the setRenderParameter() method of the ActionResponse object. These render parameters will be available for all subsequent render requests and are quite different from typical Web application request attributes. The typical Web application request attributes are only valid for a request. On the other hand, the render parameters are valid for many subsequent render requests. The render parameters remain valid until the value is explicitly changed or removed by re-execution of the action.

Consider the FavoriteColor portlet. It displays a user's favorite color in the VIEW mode, but can be changed in the EDIT mode. Submitting the favorite color choice in the EDIT mode invokes the processAction() method. This method gets the favorite color request parameter and sets it as the render parameter. Thus, the favorite color render parameter is available in all subsequent render requests.

How are the rendered parameters displayed on the JSP? Use the defineObjects tag from the portlet taglib to define portlet objects. This tag makes renderRequest, renderResponse, and portletConfig portlet objects available in the page. A parameter is displayed by invoking the getParameter() method of the renderRequest object. Refer to favoriteColorView.jsp in the included source code example.

The FavoriteColor portlet demonstrates other concepts as well. The first is how to programmatically change the portlet mode in the processAction() method. Invoke the setPortletMode() method of the ActionResponse object to change the portlet mode. The second concept is how to change the portlet mode using an HTML link. The link URL is generated using the renderURL tag from the portlet taglib. The value for the portletMode attribute is specified as the desired portlet mode. Refer to FavoriteColorPortlet class and favoriteColorView.jsp page included in the source code example.

Portlet Preferences

Portlet preferences are the basic configuration data for portlets. A preference is a name and value pair. The type of name is a string, whereas the type of value is either a string or an array of strings. A portlet preference is not suited for storing arbitrary data. The portlet container provides persistence for portlet preferences. In WebLogic Portal, the persistence for preferences works only when both of the following conditions are true:

- The portal is running in the desktop instead of DOT portal mode.
- A user is logged in.

Desktop Versus DOT Portal Mode

When a .portal file is created in WebLogic Workshop, items such as books, pages, and portlets are dragged-and-dropped into the .portal file and the .portal file can be run directly from within Workshop. However, certain features, like storing of preferences, are not available when running in this DOT portal mode. (DOT portal mode is also known as Single File Mode.)

The other mode is known as desktop mode. Using the Portal Administrator a portal is created. Within the portal, a desktop is created. Items such as books, pages, and portlets are created and placed within the desktop. In this mode, certain features, like storing of preferences, are available. (The desktop mode is also known as Streamed Mode.)

Before we go on, create a desktop:

- Launch Portal Administration (e.g., http://localhost:7001/-JSR168PortalAppAdmin/). One way to launch Portal Administration is directly from Workshop. Select the Portal menu and Portal Administration menu item.
- Log in to the Portal Administration.
- Create a new portal (e.g., JSR168).
- Inside the portal, create a new desktop (e.g., d1).
- Add the LoginPortlet to one of the pages of the desktop.
- Add the ContactPortlet to one of the pages of the desktop.

Portlet Preferences Example

The Contact portlet demonstrates Portlet Preferences. Portlet Preferences can be static or dynamic. Static preferences are specified in the portlet.xml file together with the portlet. For example, the ContactPortlet has a preference named contact-preference. The default value of the contact-preference is also specified:

```
<portlet-preferences>
  <preference>
    <name>contact-preference</name>
    <value>Email</value>
  </preference>
</portlet-preferences>
```

Dynamic preferences are not predefined in the portlet.xml configuration file. These preferences are stored and retrieved as the portlet is running. At runtime, an instance of the javax.portlet.PortletPreferences interface contains the preferences. The instance is obtained by invoking the getPreferences() method on the PortletRequest object. The value of a particular preference is obtained by invoking the getValue() method on the preferences instance.

Invoking the setValue() method of the preferences instance updates a preference value. However, an additional step is required to commit the changes. The store() method of the preferences instance is invoked to persist the preferences. Preferences can only be modified in the processAction() method. Any changes made to the preferences instance are discarded if the store() method is not invoked in the processAction() method. *Note:* As I mentioned earlier, if the user is not logged in or the portal is in the DOT portal mode, calling the store() method results in a runtime exception.

There are quite a few similarities between portlets and servlets. However, there are crucial differences as well. The portlet specification builds upon the servlet specification. The portlet container resides within the servlet container. Just as servlets are deployed within a Web application, so are portlets. Servlets and Web applications are configured using the web.xml. Portlets are configured using the portlet.xml file. A servlet has an explicit life cycle: init(), doGet(), doPost(), etc. Similarly, a portlet has an explicit life cycle: doView(), doEdit(), processAction(), etc. Methods of servlet and portlet classes must be coded in a thread-safe manner.

However, there are crucial differences as well. Servlets are allowed to do include, forward, and redirect, whereas portlets are only allowed to include. Servlets can render a complete page, whereas portlets render only page fragments. Portlets have well-defined portlet modes and Window states, unlike servlets. Portlets have more formal request handling with render requests and action requests and they have preferences. A portlet is not a servlet!

Conclusion

I started this article by describing the creation of portlets with a simple wizard. I illustrated the life cycle of the portlet and the inner workings of the portlet class implementation and described the structure and semantics of the portlet.xml configuration file and the corresponding weblogic-portlet.xml configuration file. I explained various concepts such as portlet modes and window states. I demonstrated the usage of portlet taglib and form processing in the portlet. Finally, I explained how to leverage and work with portlet preferences. Armed with the knowledge and concepts described in this article you're on your way to creating and deploying your own powerful portlets.

Acknowledgments

I want to thank Subbu Allamaraju, Max Cooper, Steve Ditlinger, David Lu, Roshni Malani and Alex Toussaint for reviewing this article and providing invaluable feedback.

References

- *To discuss the article and ask questions start here:* www.bartssandbox.com. Free membership is required.
- *Download and read JSR 168:* www.jcp.org/en/jsr/detail?id=168
- *Starting point for WebLogic Portal documentation*: http://e-docs.bea.com/wlp/docs81/index.html
- *Building Java Portlets section of the Workshop Help:* http://e-docs.bea.com/workshop/docs81/doc/en/core/index.html
- *Developing JSR 168 Portlets with WebLogic Portal 8.1:* http://dev2dev.bea.com/products/wlportal81/articles/JSR168.jsp
- *Web Services for Remote Portlets (WSRP) specification:* www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsrp
- *Take WSRP for a test drive:* http://ev2dev.bea.com/codelibrary/code/wsrp_supportkit.jsp
- *Single File Mode versus Streamed Rendering Mode:* http://e-docs.bea.com/workshop/docs81/doc/en/portal/buildportals/fwPortal4.html?skipReload=true#singlestreamed
- *Articles on Portlet Specification:*
  – *Introducing Portlet Specification, Part 1:* www.javaworld.com/javaworld/jw-08-2003/jw-0801-portlet_p.html
  – *Introducing Portlet Specification, Part 2:* www.javaworld.com/javaworld/jw-09-2003/jw-0905-portlet2_p.html
- *Introduction to JSR 168 whitepaper:* http://developers.sun.com/prodtech/portalserver/reference/techart/jsr168/pb_whitepaper.pdf
- *Java Passion Portlet Lecture Notes:* www.javapassion.com/j2eeadvanced/Portlet4.pdf

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

hp®
invent

Building Business Processes

PART 1

WEBLOGIC INTEGRATION DEVELOPMENT BEST PRACTICES

BY **VIJAY MANDAVA** & **ANBARASU KRISHNASWAMY**

AUTHOR BIO

Vijay Mandava joined BEA as a technical manager in the Professional Services organization in 1999. He now works as a principal systems engineer in the Systems Engineering organization. Vijay is a Sun certified Java programmer and a BEA certified Weblogic Server developer.

Anbarasu Krishnaswamy is a senior principal consultant with BEA professional services. He has been working with BEA Systems for over 5 years and BEA technology for about 10 years. He is a Sun certified Java programmer and BEA certified WebLogic Server developer.

CONTACT...

vijay.mandava@bea.com
anbarasu@bea.com

A business process in the real world typically is never done end-to-end by a single employee. It usually involves multiple employees/back end systems handing over work, similar to a 4x100 track relay where batons are passed between the athletes. The employees/back end should be passively notified of their tasks rather than actively waiting. BEA WebLogic Workshop provides a great framework to build these business processes for deployment on the WebLogic Platform.

Applications and Business Processes – What's the Difference?

Typically, programmers think in terms of applications (e.g., service-oriented architecture) and non-technical people think in terms of business processes (e.g., the supplier should be notified if inventory levels reach a certain threshold). A business process orchestrates applications to go from start to finish. In other words, business process is a higher abstraction, or a controller or manager for applications.

A business process orchestrates applications but shields itself from the complexities inside them. Every software architect understands the importance of layering and separation of responsibilities. One way to accomplish this pattern is to build all the domain logic/data in the application and place all the decision logic in the business process.

No specification in the J2EE space addresses modeling business processes – entity beans address business domain objects, stateless session beans address synchronous services, and message-driven beans address asynchronous services. A business process spans applications and systems.

From a different perspective, a business process involves multiple business transactions, where each business transaction can be a combination of multiple system transactions. To build an application that keeps track of what transactions are complete, does compensating transactions when necessary, and deals with retry conditions could be a tedious and complex task. This is where BPM tools can really shine and make developers very productive.

Instead of building an application that coordinates other applications or services based on data passed to it, you can build a business process that coordinates applications. A business process can be defined as something that can be modeled by a GUI tool and can be easily modified and extended without writing any code or little code.

The WebLogic platform provides tools and an environment where business processes can be defined, executed, and monitored. Using Workshop, you can define a business process through a GUI tool and this definition is then compiled into EJBs (stateless session beans for stateless business processes and entity beans for stateful workflows).

WebLogic Integration JPD Best Practices

Having set the motivation for building business processes on the WebLogic 8.1 platform, we will cover some best practices that we identified in working with WebLogic Integration over the last year. This article lists the best practices and briefly discusses how to apply the concepts in development.

Use of Perform Node

Best Practice

Do not abuse perform node. Use it only when required. Use the out-of-the-box controls as much as possible and build new logic as reusable custom controls instead of embedding Java code into perform node.

Reason

Placing business logic in perform nodes hides the business logic from the process and makes the code not reusable. By building modular custom controls, the logic becomes reusable and pluggable.

Details

There are 10–15 bugs per 100 lines of code written. Fewer lines of code written, fewer bugs in the code. In many cases there may be an alternative to placing logic behind perform node. Use the out-of-the-box controls first. If one is not available, build a custom control.

Function/Method Names

Best Practice

Use meaningful names for method names.

Reason

The function names should be self-descriptive and something that makes sense because when you use the controls and processes, the method names are shown as operations. In order for the control/process user to understand and use the operation properly, appropriate names should be used.

Details

There are a number of places in the development where WebLogic Workshop generates default names for methods, including:

- Client request nodes
- Perform nodes
- Auto-generated method names

WLDJ ADVERTISER INDEX

ADVERTISER	URL	PHONE	PAGE
Acsera	www.acsera.com/demo	408-245-1000	21
ColdFusion Developer's Journal	www.sys-con.com/coldfusion	800-303-5282	25
Compuware	www.compuware.com	800-521-9353	51
Cyanea	www.cyanea.com/j2ee	877-CYANEA8	52
H&W Computer	www.hwcs.com/wldj08.asp	800-338-6692	15
HP	http://devresource.hp.com/d2d.htm	800-752-0900	47
Intel	www.intel.com/ids/bea	800-628-8686	33
Intersperse	www.intersperse.com	626-535-2000	3
IT Solutions Guide	www.sys-con.com	201-802-3021	45
Java Developer's Journal Resource CD	www.sys-con.com	888-303-5282	49
NetIQ	www.netiq.com/solutions/web	408-856-3000	9
Pantero	www.pantero.com	781-890-2890	11
Quest Software	www.quest.com/weblogic	949-754-8000	35
ReportingEngines	www.reportingengines.com	888-884-8665	41
SandCherry	www.sandcherry.com	866-383-4500	29
SYS-CON Reprints	www.sys-con.com	201-802-3026	31
Tangosol	www.tangosol.com	617-623-5782	2
Web Services Edge 2005 East	www.sys-con.com/edge	201-802-3004	49
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

When a process control is generated from the process, the control uses these method names for its operations. This means that these names are shown in the control palette for drag and drop, making it essential to use meaningful names.

Captions and Node Names

Best Practice

Use captions and node names that make business sense.

Reason

The JPD is a graphical representation of a business process. In order to understand it and achieve business value, the node names and captions for groups and operations should be appropriately named.

Details

Use names and labels that make business sense. For example, use "Get Customer" instead of "Get row from I_CUST table in Oracle". The process label is a useful query string so design the values rather than leave it up to the implementor. For example, if a support person will need to find an order number, have it represented on the process label. Set Process Label to relevant query values.

Parallel branches are synchronized only at their termination points. A join condition is defined at the termination of multiple branches to specify how the termination of branches terminates the overall parallel activity. Valid join conditions are AND and OR.

Naming Conventions

Best Practice

Define and follow naming conventions for all of the resources.

Reason

If naming conventions are not followed, it will make application maintenance a nightmare.

Details

In a WebLogic Integration application, there are several resources created by developers and the IDE. Once you start developing, you will have a wealth of resources. If proper naming conventions aren't followed, the project will become unmaintainable. It is recommended that you define and use naming conventions for folders, controls, transformations, xquery files, processes, and other resources.

A rule of thumb following the standard Java convention of lower-case package names and names starting with upper case for classes: any names that will be part of a package should be lower case and any resource that will be used as a class will be upper case. This means that folder names will be in lower case and controls will start with upper case.

For channels use the suffix, "Channel" and start the first letter in uppercase. An example would be <Xyz>Channel.

Grouping Nodes

Best Practice

Create logical groups in the process as appropriate and use proper labels.

Reason

Grouping nodes improves clarity and provides more control over exception handling.

Details

You can create a group from one or more nodes or other groups. You can simplify the display of your business process in the Design View by collapsing a group of nodes into a single node. A group can provide an extra level of exception handling logic – exception handlers that you specify for a group catch exceptions that are not handled by exception handlers defined for nodes inside the group.

Namespaces

Best Practice

Use proper namespace values.

Reason

Namespaces are frequently used in package names of generated classes.

Details

Use appropriate namespaces in resources like XML schema and channel files. When XML beans are generated, a namespace is used for the package name of the XML bean objects.

Transformations

Best Practice

Create transformations separately and use them in processes.

Reason

If a temporary transformation is created from the process, you don't have much control over it.

Details

Transformations are created automatically and placed in the same folder as the process. Whenever a transformation is changed, a new xq file is created and placed in the same folder. The old xq file is orphaned and left there. This results in orphan files and ultimately results in an explosion of files. A new transformation control is created every time a transformation is tried. So, organize transformations in a separate package, create the transformation, and use it in a process. Even if you create a temporary transformation, you can organize it by changing the names and moving it to a different package.

Organizing the Project

Best Practice

Organize the project by grouping similar resources.

Reason

If the project isn't organized properly it will be unmaintainable and will make it difficult to understand and troubleshoot

Details

Create folders (which in turn act as packages) and group controls, transformations, and processes separately. For example, you can create packages called controls and transformations and place the control files and transformation files in the respective folders. This will group, for example, transformation files and xquery files in the transformations package. Separate groups of logically related processes into separate directories.

Summary

This article introduced you to the benefits of modeling business processes and explained best practices in building business processes on BEA WebLogic Integration 8.1. It focused on making team development and application maintenance easier.

In our next article, we will focus on best practices in building business processes with scalability, recovery, exception handling, guaranteed delivery, and performance.

References

- *WebLogic Workshop Documentation:* http://edocs.bea.com/workshop/docs81/index.html
- *WebLogic Integration Documentation:* http://edocs.bea.com/wli/docs81/index.html

Compuware

OptimalJ®

THE POWER TO Develop, Transform, Reuse

Put your J2EE™ application development into overdrive—with unmatched quality and unprecedented flexibility—using Compuware OptimalJ. Increase developer productivity up to 90%. Seamlessly instill quality into development. Forge existing infrastructures and new technologies into an integrated whole.

Realize business agility with applications that are truly reusable, time and time again.

THE POWER IS RIGHT HERE.

THE LEADER IN IT VALUE.

COMPUWARE®
www.compuware.com

Jumping 2 Enormous Eggplants.

A new way to look at J2EE.

Cyanea/One gives you unprecedented visibility into mission-critical application performance. With the transaction-based performance management capabilities of Cyanea/One, you can drill down to the method level to determine and resolve performance issues. Cyanea/One monitors complex, multi-platform, service-oriented environments 24 x 7 from a single web-based console, all without having to touch the application code. If you are developing enterprise applications, you know the hurdles you sometimes have to overcome; with Cyanea/One, your applications will reach new heights.

For more information, visit www.cyanea.com/j2ee or call 1.877.CYANEA8.

Cyanea®