FIRST LOOKS AT WORKSHOP 8.1 / PORTAL 8.1

BEA WebLogic DEVELOPER'S JOURNAL

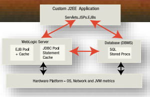
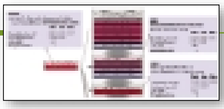
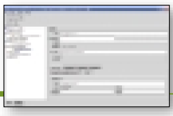
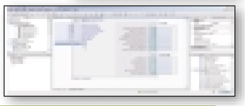
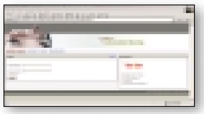
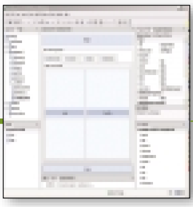
JUNE 2003 - Volume:2 Issue:6

weblogicdevelopersjournal.com

Web Services Edge WEST 2003
web services EDGE conference & expo
SEPT. 30 - OCT. 2, 2003
Santa Clara, CA
PAGE 37

RETAILERS PLEASE DISPLAY UNTIL JULY 31, 2003
$8.99US $9.99CAN
06
0 09281 03424 7

SYS-CON MEDIA

Hewlett-Packard

http://devresource.hp.com/wldj.html

BEA Systems

http://dev2dev.bea.com/useworkshop

Wily Technology

www.wilytech.com

BEA WebLogic DEVELOPER'S JOURNAL

EDITORIAL ADVISORY BOARD
TOM ANGDUCCI, LEWIS CIRNE, STUART HALLOWAY, KEVIN JONES, TYLER JEWELL, WAYNE LESLEY LUND, SEAN RHODY, BLAKE STONE

FOUNDING EDITOR
PETER ZADROZNY

EDITOR-IN-CHIEF
JASON WESTRA

EDITORIAL DIRECTOR
JEREMY GEELAN

MANAGING EDITOR
SEAN RHODY

EXECUTIVE EDITOR
GAIL SCHULTZ

EDITOR
NANCY VALENTINE

ASSOCIATE EDITORS
JAMIE MATUSOW, JEAN CASSIDY

ASSISTANT EDITOR
JENNIFER STILLEY

WRITERS IN THIS ISSUE
ARISTOTLE ALLEN, PETER CHAPMAN, LEWIS CIRNE, RINI GAHIR, YARON GOLAND, PETER HOLDITCH, JOE MITCHKO, JUSTIN MURRAY, SEAN RHODY, JASON SNYDER

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 A. KIRCAALI

VP, BUSINESS DEVELOPMENT
GRISHA DAVIDA

SENIOR VP, SALES & MARKETING
CARMEN GONZALEZ

PRODUCTION CONSULTANT
JIM MORGAN

ART DIRECTOR
ALEX BOTERO

ASSOCIATE ART DIRECTORS
LOUIS F. CUFFARI • RICHARD SILVERBERG

ASSISTANT ART DIRECTOR
TAMI BEATTY

VP, SALES & MARKETING
MILES SILVERMAN

ADVERTISING SALES DIRECTOR
ROBYN FORMA

DIRECTOR OF SALES AND MARKETING
MEGAN RING-MUSSA

ADVERTISING SALES MANAGER
ALISA CATALANO

ASSOCIATE SALES MANAGERS
CARRIE GEBERT • KRISTIN KUHNLE

PRESIDENT, SYS-CON EVENTS
GRISHA DAVIDA

CONFERENCE MANAGER
MICHAEL LYNCH

FINANCIAL ANALYST
JOAN LAROSE

ACCOUNTS RECEIVABLE
KERRI VON ACHEN

ACCOUNTS PAYABLE
BETTY WHITE

VP, INFORMATION SYSTEMS
ROBERT DIAMOND

WEB DESIGNERS
STEPHEN KILMURRAY • CHRISTOPHER CROCE

ONLINE EDITOR
LIN GOETZ

CIRCULATION SERVICE COORDINATORS
NIKI PANAGOPOULOS • SHELIA DICKERSON

JDJ STORE MANAGER
RACHEL MCGOURAN

EDITORIAL OFFICES
SYS-CON Publications, Inc.
135 Chestnut Ridge Road, Montvale, NJ 07645
Telephone: 201 802-3000 Fax: 201 782-9637
SUBSCRIBE@SYS-CON.COM
BEA WebLogic Developer's Journal (ISSN# 1535-9581) is published monthly (12 times a year)
Postmaster: Send Address Changes to
BEA WEBLOGIC DEVELOPER'S JOURNAL,
SYS-CON Publications, Inc.
135 Chestnut Ridge Road, Montvale, NJ 07645

SYS-CON MEDIA

Management 101

BY SEAN RHODY
MANAGING EDITOR

From time to time I hear people say "those who can, do; those who can't, manage." Usually a developer mutters this as he begins another 80-hour week courtesy of a slip in the project plan. Of course, once you get to be management yourself, you realize there's more to it than simply ticking off hours on a project plan.

This holds true for application servers as well as development teams. It seems that development features often take precedence over management in the planning of release cycles for application servers. To a certain extent, this can be pushed up a little in origin, to the J2EE specification itself, which up until now has been somewhat sketchy on the management aspect.

While we're on the subject of the specification, let me throw my usual plug in for management nirvana – standardize the deployment and management console please! JMX is all well and good for instrumenting applications, but we need a standard set of properties, appearances, and behaviors to make it easier for an application developer to move from one server to another.

I realize that the console is one area where vendors tend to differentiate themselves, but as a veteran of successive versions of WebLogic, and of J2EE, I can speak for the vast majority who are tired of relearning how to deploy an application with every new version of the specification or the server. Enough is enough.

But enough of that rant. Back to management itself. We need to be able to do three things in an application server – monitor an application, manage aspects of the application's behavior, and maintain the application through successive releases of code.

From a monitoring perspective we need to know how many transactions (I use the word transactions to indicate a completed logical unit of work by the container, but others may disagree) have taken place, how many per second, the average throughput, the peak throughput, peak sustained load, refused connections, exceptions, etc. The whole concept of monitoring is being able to look at specific concepts around an application to determine how well it is running, over a variety of metrics.

Monitoring would be of little value if we were unable to do something about it – that's what management is all about. From the basics of being able to add another server, or deploy a component on another machine, to being able to change the routing of messaging, management allows us to control and change aspects of an application in real time. Some of these aspects are generic, such as locations and distribution of queues, beans, and pages. Others are specific, and must be engineered through JMX, which again will be fed back to the console.

Finally, we need to maintain applications – throughout their life cycle. Some of this has to do more with source code control and configuration management, but the ability to deploy into an application server and maintain an application is also crucial.

Of course, this doesn't imply that BEA hasn't done a good job on creating a management environment for WebLogic – it has. Many of the features I've cited are part of the WebLogic environment. What I want is for the environment to stay stable in the future – not switch from command line, to console, to JSP app, to whatever is next. Granted, that is somewhat the nature of the JCP, which is why we need BEA to be pitching for a management standard.

Enough ranting. Time for me to tell a project team they have three days to do an entire application. Ah, the joys of management.

AUTHOR BIO...
Sean Rhody is the managing editor of *WebLogic Developer's Journal* and editor-in-chief of *Web Services Journal*. He is a respected industry expert and a consultant with a leading consulting services company.

CONTACT: sean@sys-con.com

ENHANCING APPLICATION MANAGEABILITY

PART 1

BY **JUSTIN MURRAY**

AUTHOR BIO...

Justin Murray is a technical consultant in the application development resources organization in HP. He has worked for HP in various consulting and teaching roles. Justin has consulted at a technical level on customer projects involving Java, J2EE, and performance management, as well as specializing on application performance tuning for HP-UX. Justin has published several technical papers on these subjects.

CONTACT...

justin.murray@hp.com

When we build enterprise applications based on either a J2EE-compatible application server or an XML Web services platform, we tend to leave the manageability of our application as a problem for the base platform to solve. We therefore may not do any work in our business logic to enhance the manageability of our application in production. This is not a good policy and can lead to an application that is more difficult to manage than it should be when it is deployed.

J2EE application servers, a foundation for XML/SOAP Web services applications, give us tools for monitoring our software servers and, to an extent, our applications. However, software developers need to exploit these facilities to give more *application*-level manageability, as opposed to *platform*-level manageability. The difference between these two can be compared to knowing the number of items in my shopping cart (an object at the application or business logic level) as opposed to what is happening in the BEA WebLogic Server (WLS) message queues that are supporting it.

The absence of manageability can cause serious problems. The application software can become an unmanageable entity once it is deployed because the operations staff doesn't understand it. Lack of attention to manageability raises the cost of supporting and maintaining the software product significantly. Software developers can be engaged in solving post-deployment management problems that could have been handled in steps taken earlier in the software life cycle. Those steps will be described here.

This article describes a range of choices for the developer, including:

- Writing messages from the application code to human-readable files, and having those messages appropriately processed by a management tool
- Encapsulating some of these logs in management templates
- Using prebuilt Java Management Extensions

USING BEA'S WEBLOGIC

(JMX)–based managed bean objects and tools that manipulate them
- Building and using your own JMX managed bean objects for specific purposes

The concept behind these JMX managed bean objects will be explained later.

These steps can be viewed as incrementally increasing the levels of manageability in an application. They may be followed separately or all together for one application. The article positions the various technologies to help the developer integrate them and build a better application. The good news is that much of the hard work has already been taken care of by the application servers and by management tools. However, there is still some work to be done in the application.

Background

As developers, we sometimes believe we have done enough work towards making an application manageable if we have produced messages in a text log file for our application, and no more than that. This can be sufficient for the management of very simple application deployments. Should two copies, or instances, of the application be required to run in parallel, then the log file method will need more design attention.

Application manageability is not limited, however, to knowing simply whether the application is alive or not, or whether the application has produced an error message in its log file. For certain applications we may well need to know how many user transactions the application processed over the past hour or two, how long on average each transaction took, and what percentage of them failed to complete. Sophisticated applications have queues of requests, some of which are outstanding and some are being processed. The system manager in such cases needs to know the condition of each queue, so that he or she can respond to problem situations appropriately. Should a queue of incoming requests be causing delays to the end user, for example, the administrator may wish to allocate some of those requests to another identical process, thus avoiding a problem.

We need a framing definition of what the term "manageability" means in order to judge how much of it to build into any one application. This "level of manageability" determines what technologies we use and how much effort we expend on this aspect of the application design.

Definition of Manageability

We define the term "manageability" as the ability to exercise administrative and supervisory actions and receive information that is relevant to such actions on a component.

Manageability is further broken down into three main pieces:

1. ***Monitoring:*** The ability to capture runtime and historical events from a particular component, for reporting and notification.
2. ***Tracking:*** The ability to observe aspects of a single unit of work or thread of execution across multiple components (e.g., tracking messages from senders to receivers).
3. ***Control:*** The ability to alter the runtime behavior of a managed component (e.g., changing the logging level of an application).

Manageability covers the activity of ensuring that the application is alive and functioning normally, as well as checking that an application's performance is as expected. When a problem occurs, manageability tools enable the troubleshooting engineer to find the problem. These aspects of manageability are used to assess the various technologies.

Further, manageability covers the monitoring, tracking, and control of more than one instance of the same software. For example, applications built on J2EE application servers are frequently replicated on several computers for load balancing and fault tolerance reasons. Operators need to be able to see and control the levels of activity of each one of these instances at all times. They can then reroute requests or take other actions to allay problems and maintain service level agreements.

Key Design Decisions in Manageability

The decisions the developer must make in applying manageability functionality to his or her software are:

- What level of information is most useful to the operator of the software?
- What actions can be taken on the software as a result of this information?
- What level of detail (fine-grained access to individual objects, or coarser-grained access to subsystems, modules, or otherwise) is best for application manageability?
- Is the management software to be allowed to interact synchronously with the application code or must all actions be taken "after the fact"?

Approaches to Building Manageability into Applications

This section describes a set of technology approaches to building manageability into an application. They are presented in ascending order of "work to do" on the part of the developer. They are not mutually exclusive and in many applications the right level of manageability is achieved through a combination of several of these techniques. Most application developers will recognize the first option, writing messages to a log file, as something they would normally consider doing with any application. This means that legacy applications that produce log files will be easy to arm with this level of manageability, which fits in the "monitoring" category above.

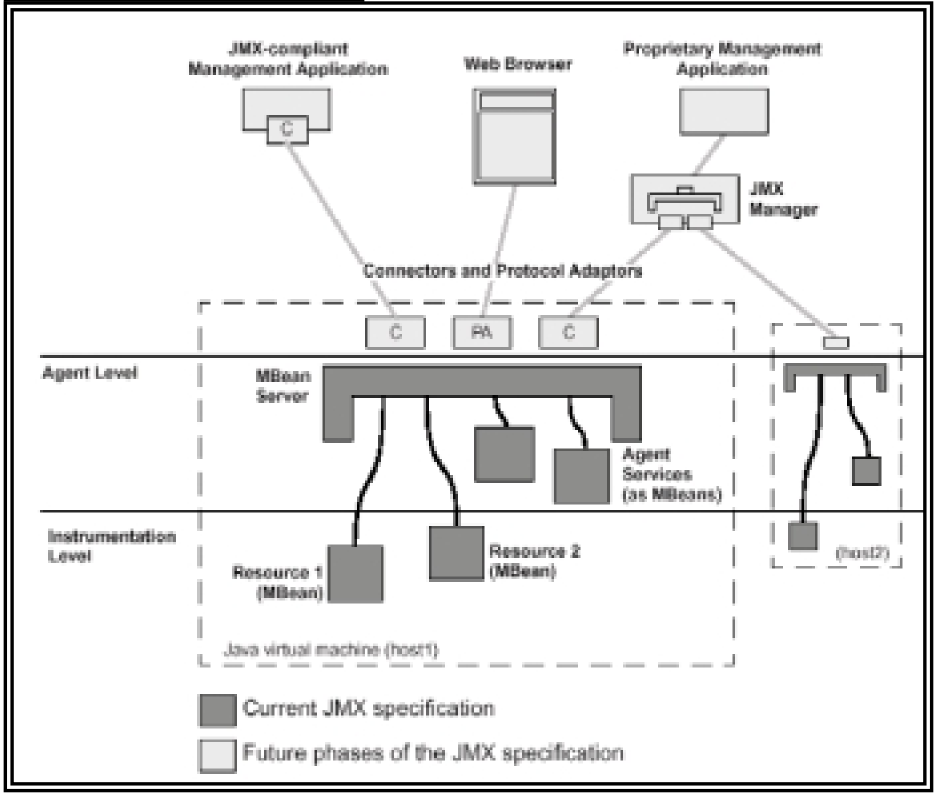
Application Log Files

Many applications are designed to produce a set of textual, human-readable error messages in their log files or to the screen using standard programming I/O mechanisms. These log files are either viewed by an operations person for identification of issues, or scanned by a separate process that takes critical messages and displays them in some form on a management console. Management infrastructures such as HP's OpenView Operations (OVO), with its message sourcing and processing functions, allow for this type of capability and also allow sophisticated filtering of those messages.

Direct Logging from the Application to the Management Console

Each application takes its own approach to solving this problem. The "opcmsg" command, for example, in the HP OpenView management suite, allows the direct logging of messages to OpenView management consoles, along with a similar C and Java API for programmatic access to the console. Programmers choose whether to build calls to this logging API into their program based on the urgency of the need to get this message to the console at a point in time without further filtering. Labeling the various text messages in a log file with

FIGURE 1


JMX-compliant Management Application
Web Browser
Proprietary Management Application
JMX Manager
Connectors and Protocol Adaptors
C
PA
Agent Level
MBean Server
Agent Services (as MBeans)
Instrumentation Level
Resource 1 (MBean)
Resource 2 (MBean)
Java virtual machine (host1)
(host2)
Current JMX specification
Future phases of the JMX specification

JMX levels

severity levels or categories such as “informational”, “warning”, “critical”, and others can help with the interpretation of those messages by a management tool. These labels are essential for operators who don’t know the internals of the software. The management tools will pick up “critical” messages from a potentially large set of messages and highlight them on the display device.

Indirect Logging from the Application

However, the application programmer does not have to embed calls to the console logging APIs, mentioned earlier, in the application. The application developer can concentrate on logging messages to a file or other persistent storage and later use the management tools to selectively display those messages.

In the OVO toolkit are utilities for encapsulating application log files (and other sources of data, such as binary files and SNMP traps) as message sources. Further, there are screens for setting up “templates” or patterns for processing these messages at a centrally located management server, and then distributing these templates to the managed nodes so that all instances of an application can be monitored in this way. Such processing patterns may define that certain message types be excluded from appearing at management consoles, for example. The processing template may restructure the format of the messages, or may aggregate a set of messages into one, to avoid overloading the operator with too many messages. The choices for the template designer are many and are fully documented in the OVO administrator’s guide. All the developer or IT operations person needs to do is:

1. Identify the file or other source of messages (such as SNMP or the event log in Windows).
2. Choose an existing message template from the supplied set, or create a new one.
3. Attach a template for message processing to the message source.
4. Distribute this message template to the locations (machines, application sites) where it is needed.

The developer can choose the monitoring options for such log files and make decisions on factors such as:

- The interval that the log files should be read at
- Whether to read from the beginning of the file or from the last scanned position
- Whether to close the file after an access has been made to it, and other factors

Issues in Log File Encapsulation and Message Template Processing

Filtering messages: The message logging approach can easily become subject to the problem of producing too many messages to give the operator a meaningful picture of what is happening with the software. Careful attention to labeling each message as “Informational”, “Warning”, “Critical”, and possibly other categories can help with this. Developers should decide with the operators whether they want to suppress all noncritical messages.

Filtering of the messages to highlight those that are of interest in certain situations can be achieved. These filters can then feed messages into a subsequent notification command or piece of logic. Tools exist in the management platform that can take these messages and demand that the operator respond in some fashion to the message.

Correlation: Message correlation is also a task that requires the user or software designer to work harder with log files. Either the end user/operator has to correlate the messages, which apply to one situation, by hand, or a special piece of functionality has to be written to perform the correlation for him. The OpenView Operations templates take care of this issue for the developer.

Further details on log file handling and message processing can be found in the OpenView Operations Concepts Guide (OVO-Concepts) and in the OpenView Operations Administrators Guide (OVO-Admin).

Log Files Summary

This message file encapsulation approach is recommended as a minimal starting level for building manageability into an application. It is a necessary, but not sufficient, condition for adequate application manageability. With this approach, we have achieved part of the “monitoring” aspect of manageability described in the earlier definition section. That is, we can visualize at least those parts of the application that the developer takes care to tell us about, but we cannot exercise control over it as yet. We tackle the “control” subject in the next section.

The Java Management Extensions Standard

The Java Management Extensions (JMX) have become the accepted industry standard for managing Java applications. The specification for JMX within the Java com-

The ServerSide Symposium

www.theserverside.com/symposium?sys1

munity process (JCP) is a Java Specification Request (JSR) named JSR3. A separate specification, JSR77, which is part of the J2EE 1.4 specification, is a description of the "model" of objects that each application server must expose through JMX. This version of the J2EE specification,1.4, was still in preparation at the time of this writing. Further detail on these submission requests can be found at JSR3 and JSR77, respectively. The JMX specification provides for the construction of the manageability aspect of Java applications in a standard way.

The JMX environment is composed of three levels of software

- Instrumentation
- Agent
- Distributed services

These levels are shown in Figure 1 (the distributed services layer is the topmost one).

At the Instrumentation level, there are managed bean objects, or MBeans. These are Java objects that conform to either

- A simple style of object construction in the Java world called JavaBeans (for simple MBeans) or
- A JMX standard called "DynamicMBean" for more flexible management or
- The JMX "Model MBeans", which are an extension to Dynamic MBeans that provide more generic templates for management instrumentation

Conforming to the JavaBeans format is straightforward. It implies that an object is serializable and has a null constructor. It is common practice to implement "getter" and "setter" methods on all of that object's properties or data members (class variables) that may be required to be visible to external management tools. Creating MBeans in this style means that once the MBeans are registered with the JMX server, called the MBean server, they can be reached for monitoring and control purposes by the tools at the distributed services layer at the top of the diagram.

JMX requires the use of an MBean server shown at the Agent Level in Figure 1. MBeans are required to register their presence with an MBean server in order to be noticed by the management framework. The MBean server handles the management messages that are flowing to and from objects that have been previously registered as MBeans. Certain properties of the object that conform to the MBean interface can then be viewed by management tools and, in some cases, the behavior of the object can be changed using the functionality of management consoles. This is done through the connectors and other adapters that allow a management console to extract data from a JMX server, not through direct manipulation of the MBeans themselves.

Key Issues with MBeans for the JMX Developer

The MBean mechanism is powerful in the sense that it provides an industry-standard method for "wrapping" a business object with another MBean object, the latter being dedicated to manageability.

Trade-offs in using the MBean Interfaces Directory

The specification for JMX does not determine whether a business object may itself be an MBean. This is possible, since the MBean hierarchy is made up of interfaces. The disadvantage of this approach is that the business object now contains both business logic and manageability logic, causing it to be more complex. This approach may provide a benefit in the sense that a reference or pointer may be eliminated between the business object and its peer MBean object if they are both contained in one. This will be discussed later.

The developer has a choice of making their business object conform to the appropriate JMX-style interfaces or building separate JMX bean objects that are specifically for the management of their business objects. Why would this separation be interesting? Well, both techniques have their advantages.

Using the inheritance method, where business objects inherit their JMX management interface, every such business object instance will need to be registered with the MBean server – and there could be thousands of them. With a separate object for manageability, which is the one conforming to the JMX interfaces, it alone registers with the MBean server, causing the number of entries to be fewer. This single management object will then be responsible for managing a group of business objects. There may be a need for only one management object for all instances of a business class. This is a trade-off and there is no right answer for all applications. The developer must make a design decision at this point. This will be discussed later.

Communication and Linkage

The JMX specification does not specify *how the communication is achieved between the business object and the MBean*. This is entirely under the application developer's control.

Key decisions therefore must be made by the developer on questions such as

- Are all or just some of the attributes (member variables) of the business object exposed to management tools?
- Are these attributes only readable or writable?
- At what frequency will updates be carried to the business object from the MBean and vice-versa?
- Will there be an MBean for a set of business objects or for each business object?

The application developer has full control over the amount of communication going on between these business objects and MBean pairs. The application developer can choose to have a reference from one to the other or a bidirectional reference beween them.

Conclusion

In this first of a two-part article, I started from the position that the manageability of the application determines its acceptance in production. I looked at a definition of application manageability and at some options for adding it into your Java/J2EE application, from the very simple message logging to the more comprehensive JMX approach. These approaches should be seen as complementary rather than competing with each other. In the second part of this article I'll look at the management tools and how they help us build more manageability into our applications, whether through applying existing templates or through writing code.

References

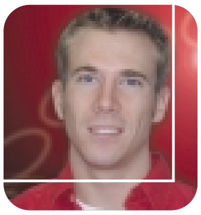
- *BEA JMX:* http://edocs.bea.com/wls/docs70/javadocs/index.html
- *JSR3:* www.jcp.org/en/jsr/detail?id=3
- *JSR77:* www.jcp.org/en/jsr/detail?id=77
- *OVO-Metrics:* HP OpenView Operations – Metrics Guide
- *OVO-Admin:* HP OpenView Operations – Administrators Guide
- *OVO-Concepts:* HP OpenView Operations - Concepts Guide
- *OVTA:* HP OpenView Transaction Analyzer: www.openview.hp.com/products/transaction_analyzer/index.asp
- *Poole:* HP OpenView Architectures for Managing Network Based Services

Quest Software

http://java.quest.com/qcj/wldj

Diagnosing Application-Database Performance Bottlenecks

FOLLOWING SIMPLE RULES SAVES TIME, IMPROVES PRODUCTIVITY



BY **PETER CHAPMAN & RINI GAHIR**

AUTHOR BIO...

Peter Chapman is a product manager with Quest Software. With more than five years of experience in software architecture design and performance tuning, as well as high tech product management, Peter's responsibilities include identifying market needs, defining new product requirements, and developing product marketing strategies.

Rini Gahir is the product marketing manager for J2EE Solutions at Quest Software. With more than eight years of experience in ERP, e-commerce, and software application marketing, Rini's responsibilities include identifying market needs, defining new technologies to meet customer requirements, and steering product development strategies.

CONTACT...

rini.gahir@quest.com

The Rise of J2EE

J2EE has arrived as the standard enterprise-computing platform for Web application development, and is gaining strength and popularity every day. J2EE supports legacy applications and interfaces, multiple operating systems, distributed and clustered environments, and high-volume mission-critical applications with support for security and managed operations.

By providing a framework and blueprint for developing distributed, scalable applications, J2EE allows companies and their developers to focus on writing modular pieces of custom application code and forget about the underlying details of security, resource management, and scalability.

Industry-leading application servers, such as BEA WebLogic Server, provide a great number of features and services. Reliability, availability, and scalability are provided by the clustering and failover features of multiple WebLogic Server instances. Other services like security and resource management – such as execute thread pools, EJB caches, and JDBC pools – are also provided. Third parties write Java Database Connectivity (JDBC) drivers to further abstract and simplify the coding of data access across different database management systems (DBMS).

The result is a powerful platform that greatly simplifies and abstracts the low-level details of developing distributed, high-volume enterprise applications.

The Challenge of J2EE Performance

Sounds like nothing can go wrong, right? Nothing, until the application fails to meet the performance criteria demanded by end users or service-level agreements. Success of a development project and business initiative can hinge on the ability to detect, diagnose, and resolve these performance problems quickly.

Due to their increased complexity, performance bottlenecks in these multi-tiered, distributed J2EE applications are much more difficult to diagnose than in earlier monolithic application environments. J2EE environments contain multiple interconnected layers of software and hardware components that all interact to service any given end-user request. Performance team members – the architect, developer, app server administrator, and database guru – have their own view of the system, and potentially their own silo or "device-centric" diagnosis tool. But how does this team work together to isolate problems? Without comprehensive visibility into all components in a J2EE system and their interactions with one another, how is a performance expert to figure out which server is slow? Which component is slow? Which resources are scarce? Is the database engine the primary bottleneck, or just a minor contributing factor?

The challenge can be daunting for the unprepared or ill equipped. Confusion can leave team members scratching their heads, or worse, pointing their fingers.

"Is It the Application, or Is It the Database?"

One of the most common frustrations heard from companies that are facing underperforming distributed J2EE applications is "Is it the application, or is it the database? Where do we start trying to fix this?" All too often, someone hazards a guess and ends up searching through reams of code looking for incorrect or inefficient algorithms, or combing through SQL statements and database tables. In other words, they pick either the application or the database (or in the worst case, both) and try to optimize that isolated piece of the puzzle.

Unfortunately, this view of problem solving is often overly simplistic as neither the application, nor the database, nor the WebLogic Server, operate in isolation. A comprehensive approach to solving this problem requires gaining visibility into all three pieces of this interrelated system: WebLogic resource utilization and configuration, the application architecture, and the database query execution, as well as underlying hardware infrastructure performance.

With insight into these subsystem interactions, performance team members can effectively isolate problematic components and triage the problem to the correct functional expert for repair.

Visibility into Interrelationships is Key

Visibility *and* interrelationships are the two key words here. Without *both* of these, detection and root cause diagnosis are extremely difficult.

We need visibility into the runtime JMX performance metrics of the WebLogic Server. We need visibility into the timing and structures of SQL queries and stored procedures run by the database engine. And most important, we need visibility into the end-to-end timing of the end-user requests across the distributed system, with all component interactions with the JDBC connection pools and DBMS calls explicitly mapped out on a *per request basis*.

Visibility is needed into not just method-level timings of the application, but into the way in which each component interacts with another to form the application architecture. Isolated metrics of method and JDBC timings without context to the application architecture, which are provided by most J2EE performance monitoring tools, are nearly useless. So are isolated response times of SQL statement execution, without regards to where the statement originated, how many times it was called, or what other interactions with the database its end-user request made.

The ideal tool solution provides insight into how the custom J2EE application interacts with the WebLogic server resources and DBMS, and how these interactions contribute to the overall response time of each end-user transaction. A high-level representation of these interactions is outlined in Figure 1.

Typical Application-Database Performance Bottlenecks

By now you're probably thinking, "Enough of the theory, how do I fix *my* application?" The following sections will discuss three main classes of performance bottlenecks, distilled from measuring and analyzing the performance of dozens of real-world J2EE applications, including J2EE systems of several prominent financial, telecommunications, and manufacturing Fortune 100 companies. By no means is this meant to be an exhaustive list, or a "debugging checklist" that can be followed step-by-step to tune any application. Each J2EE application, due to its distributed nature and complicated architecture, will have its own unique performance characteristics, but here are some common pitfalls to avoid.

I'll look at the three classes of application-database performance, with specific examples gathered by running the BEA PetStore sample application, and performance data collected and analyzed with Quest Software's PerformaSure.

Excessive Database Calls

• ***"Client"-side data processing and ResultSet scrolling***

By far the most serious performance bottlenecks in J2EE-database applications come from excessive calls being made from the user application to the database engine. These unnecessary extra calls do not necessarily have to be the Execute() or Update() of SQL queries, but nearly always involve other interactions with the database, such as ResultSet operations. A common mistake is to specify too narrow a query, and then scroll through the returned data one row at a time using ResultSet.Next(). This leaves the performance of the application at the mercy of the dataset within the DBMS – for large table sizes, this scrolling can be massive; I have seen data at client sites where over 50,000 calls were made to ResultSet.Next() for each SQL query executed.

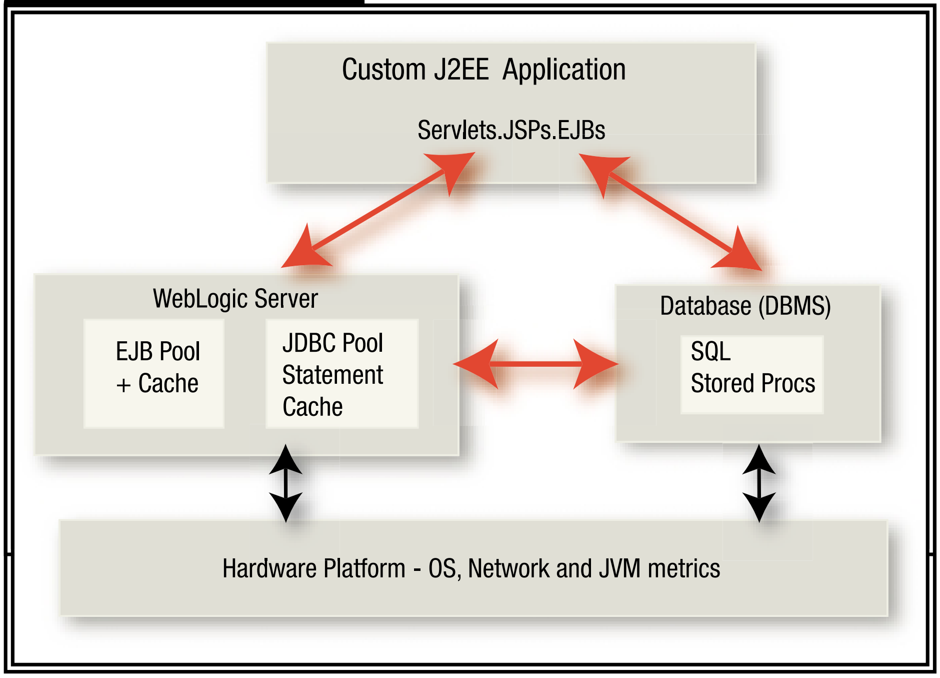
If some data processing must be done within the application code, consider fetching the required data in bulk from the DBMS, to avoid the application having to call back to the database repeatedly to retrieve each row in the dataset.

Example 1: Excessive ResultSet Scrolling

Figure 2 shows the PerformaSure reconstruction of "commitorder" HTTP request within the PetStore application, as the request executes servlet, JSP, and EJB code, eventually calling a SQL statement in the DBMS. The color-coded scale on the right-hand side indicates which methods executed quickly (cool blue), as well as the expensive hotspot methods (hot red). The tooltip at the right provides the overall timing and call count of the request. Clearly the database node at right is a hot spot, and needs further investigation.

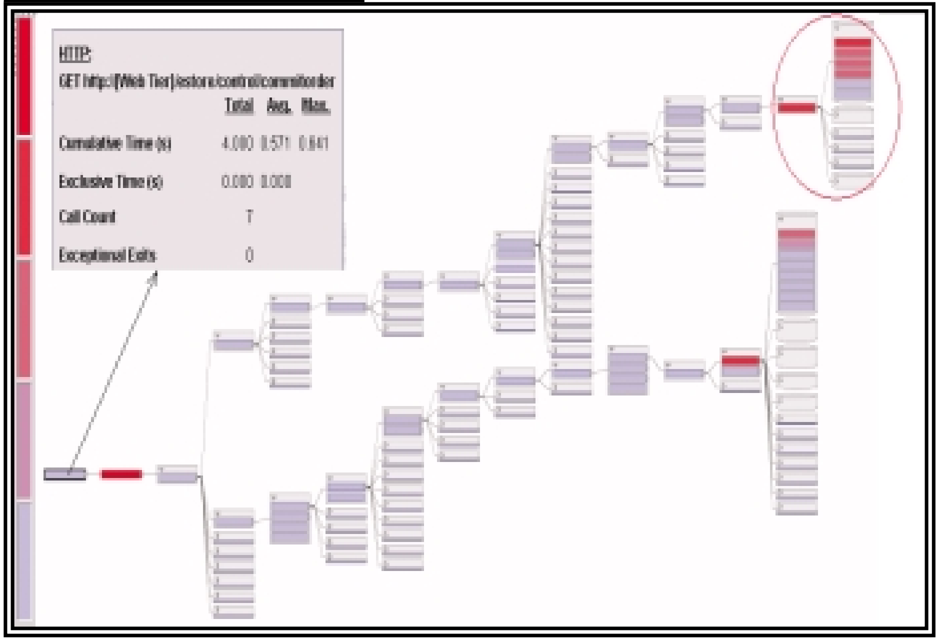
By zooming into the hot spot identified, we can see a detailed timing and call count breakdown of all JDBC operations that were executed by this transaction, such as the opening of a JDBC connection, the creation and execution of a "SELECT" SQL state-

FIGURE 1



Application/database interactions

FIGURE 2


HTTP:
GET http://[Web Tier]/estore/control/commitorder

	Total	Avg.	Max.
Cumulative Time (s)	4.000	0.571	0.641
Exclusive Time (s)	0.000	0.000	
Call Count	7		
Exceptional Exits	0		

Transaction Performance Map showing application architecture and hotspots

ment, the ResultSet.Next() scrolling, and finally, the closing of the connection. The EJB method "GetItem" was called 7 times (corresponding to the seven commitorder HTTP requests), executed a fast running SQL statement 7 times, and then scrolled through the ResultSet 672 times. This excessive chatter with the DBMS took more time than executing the actual query! This is not a scalable architecture – as the dataset grows in the DBMS and more concurrent end- users execute transactions, this type of performance problem only worsens.

- ***Two queries instead of one***

Another rule of thumb is to leave the design of SQL queries and updates to the database expert, who is intimately familiar with the various table schemas and indexes. Too often the developer writing an EJB has an idea of what data he or she would like to pull from the database engine, or the update needed. The difficulty lies in writing the fewest and most efficient queries and updates needed to perform the task. Learning to select only the data you really need in the application is crucial. It reduces the amount of processing the RDMS must perform as well as minimizing the number of queries and data sent across the network.

Any set-based processing is implemented most efficiently by the DBMS rather than pulled over the network and performed within application logic in the EJB layer of WebLogic Server. While the EJB layer exists to hold the "business logic" that encapsulates the rules and conditions for which the application queries and updates data, the actual implementation details are best handled by the database engine. Low-level query-processing logic, such as selecting initial data into temporary tables and making further queries based upon that data, is best handled by the DBMS, in stored procedures.

Database Connection Pool Problems (JBDC)

- ***Connection Pool Leaks***

A connection pool "leak" occurs when a component (usually an EJB) within a user application requests a connection from a pool, queries or updates some data, and then fails to release the connection. While it is simple to detect that a connection pool has quickly reached its maximum by examining the WebLogic JMX performance metrics ("Connections" and "Waiters") and observing slow DataSource.GetConnection() response times, it is difficult to pinpoint the source of the leak within the application code itself. This becomes even more difficult if multiple end user requests, and hence multiple pieces of application code, allocate connections from the same JDBC pool. Which request isn't freeing connections?

To solve this, a tool is needed that explicitly maps the application's interaction with the data source on a per request basis. The example below shows the "CommitOrder" HTTP request from PetStore allocating and freeing the connections to the data source

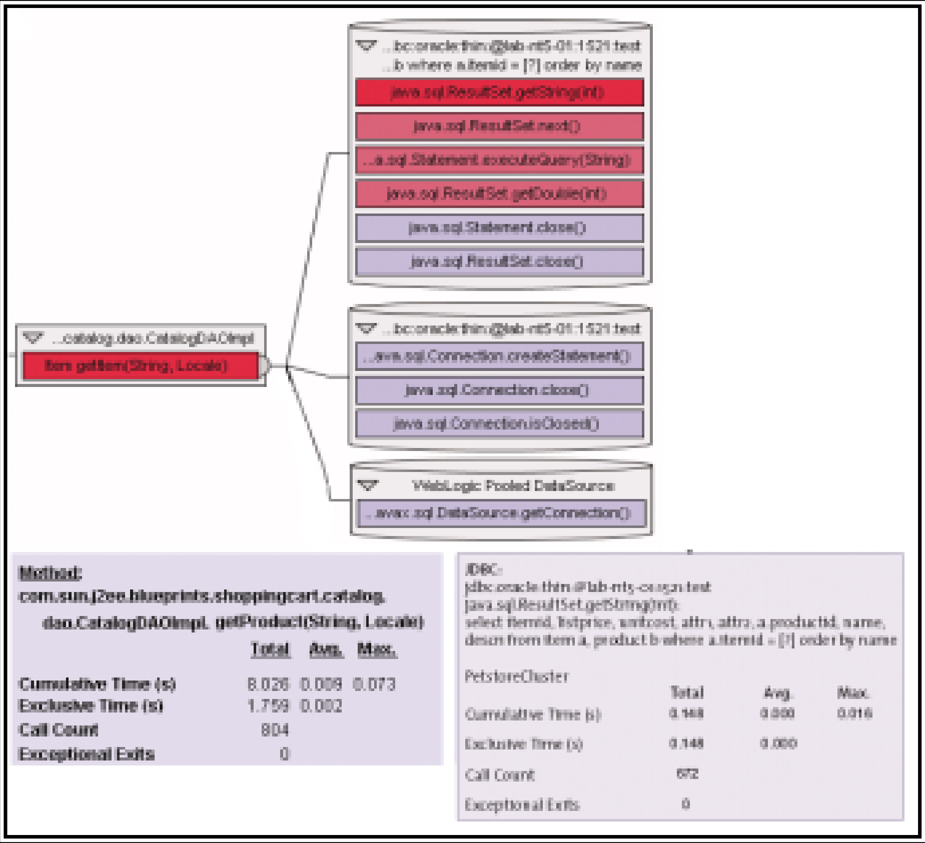
Example 2: Connection Pool Leak

Figure 4 shows two telltale WebLogic JDBC metrics that can indicate a connection pool leak. The first graph shows the

Precise Software

www.precise.com/wldj

FIGURE 3


...bc:oracle:thin:@lab-nt5-01:1521:test
...b where a.itemid = [?] order by name
java.sql.ResultSet.getString(int)
java.sql.ResultSet.next()
...a.sql.Statement.executeQuery(String)
java.sql.ResultSet.getDouble(int)
java.sql.Statement.close()
java.sql.ResultSet.close()
...catalog.dao.CatalogDAOImpl
Item getItem(String, Locale)
...bc:oracle:thin:@lab-nt5-01:1521:test
...ava.sql.Connection.createStatement()
java.sql.Connection.close()
java.sql.Connection.isClosed()
WebLogic Pooled DataSource
...avax.sql.DataSource.getConnection()

Method:
com.sun.j2ee.blueprints.shoppingcart.catalog.
dao.CatalogDAOImpl. getProduct(String, Locale)

	Total	Avg.	Max.
Cumulative Time (s)	8.026	0.009	0.073
Exclusive Time (s)	1.759	0.002	
Call Count	804		
Exceptional Exits	0		

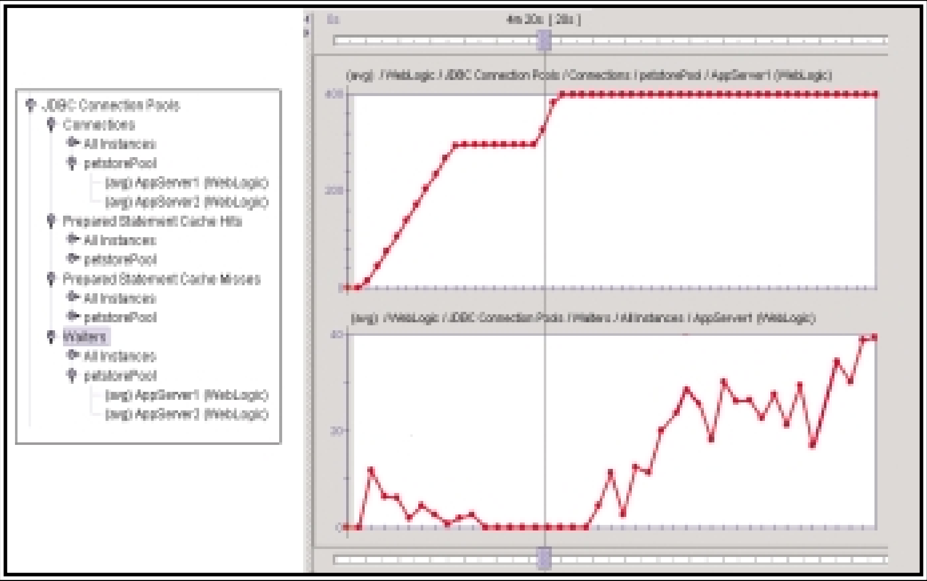
JDBC:
jdbc:oracle:thin:@lab-nt5-01:1521:test
java.sql.ResultSet.getString(int):
select itemid, listprice, unitcost, attr1, attr2, a.productid, name,
descn from item a, product b where a.itemid = [?] order by name

PetstoreCluster

	Total	Avg.	Max.
Cumulative Time (s)	0.148	0.000	0.016
Exclusive Time (s)	0.148	0.000	
Call Count	672		
Exceptional Exits	0		

JDBC breakdown of identified hotspot

FIGURE 4



WebLogic JDBC Connection Pool Metrics

number of connections in the pool quickly increasing to 400, the maximum size of the pool. At the same time that this maximum is reached, the number of waiters (requests waiting for a free connection) increases consistently, hinting that a connection leak may be occurring. But where in the source code is this happening?

Figure 5 shows the "product" request and identifies two separate pieces of application code within the request that allocate and release JDBC pool connections. A quick pan and zoom into these areas provide details of how the connection pool is being used.

In the second area we identified in Figure 5, we can immediately verify the source of our performance problem. The EJB calls GetProduct() a total of 804 times in this time period, resulting in 804 calls to executeQuery() (tooltip not shown). Checking the counts of the getConnection() and Connection.close() shows that while 1,012 JDBC connections were requested, only 756 were freed. Confirmation that this is causing performance degradation is evident by the red coloration and timing information for the GetConnection() call. It is important to note that while connection leaks are easy to detect with metric data, the root cause of these leaks is difficult, if not impossible, to pinpoint in the code if multiple transactions, or multiple pieces of code within a transaction, all allocate connections from the same JDBC pool.

- ***Size of JDBC Connection Pool***

A good rule of thumb is to take the size of the execute thread pool size, multiply it by the average number of concurrent database connections that each end-user transaction (thread) utilizes, and add 10% for peak periods of load. This average number of connections per transaction is usually one, making the JDBC connection pool and the execute queue the same size notwithstanding any desired peak load buffer. This pool size can be iteratively tuned by running tests with large JDBC pools and observing the average and peak values of JDBC pool utilized for any given execute queue size. The timing of calls to Database.GetConnection() should also be monitored to ensure there is no significant waiting time for JDBC resources.

A second point to remember when using JDBC pools in production systems is to always set the initial pool size equal to maximum pool size. By doing this, the performance overhead of creating all items in the pool occurs during the WebLogic Server startup, and not during end user runtime.

Incorrectly Formed Database Queries

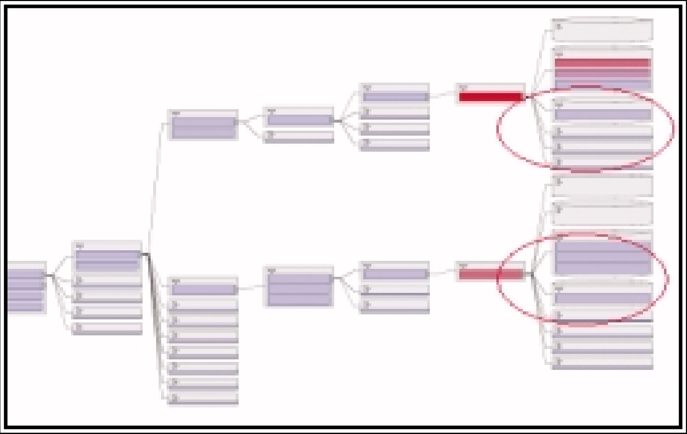
- ***Poorly formed SQL statements or stored procedures***

This common problem is related to the "two queries instead of one" performance bottleneck discussed earlier. In this case, a single SQL statement or stored procedure is called only once (or the appropriate number of times) per end-user request, yet still

Yahoo

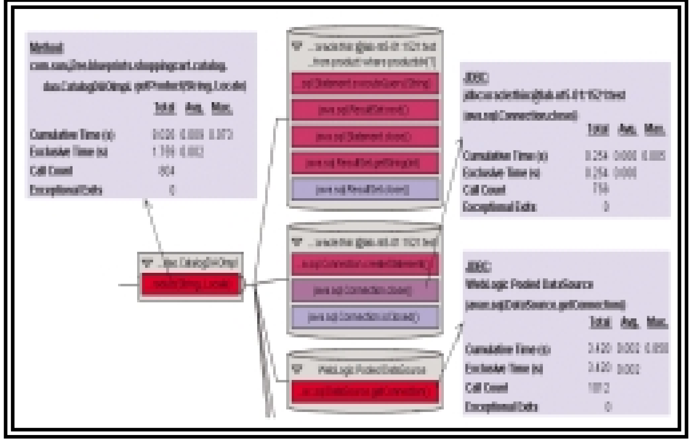
www.enterprise.yahoo.com/bea-testdrive

FIGURE 5



Application map of "product" HTTP request

FIGURE 6



Identifying a connection pool leak in application code

takes a significant amount of time to execute. Luckily, long-running statements are easy to detect with advanced performance tools, and as long as it has been confirmed that the application is efficiently interacting with the database, the database administrator can be called in to tune the offending SQL statement or stored procedure.

The key is having a tool that allows a single user to rule out the application design as a problem by breaking down the various JDBC operations that are performed for each statement with exclusive timing information and call counts. Then the problem can be triaged to the DBA for further drill-down into table schemas, indexes, and locking using his or her silo DBMS performance tools.

Another performance problem with database operations is exceptions thrown during the execution of stored procedures or the failure of completing database transactions resulting in Rollback() calls. In several scenarios we encountered, these were very quick fixes for the DBA – turned around in a matter of hours. The problem was gaining visibility into what methods in the application code were calling what stored procedures that were throwing exceptions, and having the appropriate information triaged to the right expert. Exceptional exit information is provided by the tool tips shown in the previous examples, as are calls to Rollback() and Commit() operations, when the application makes use of such calls.

- ***Not making effective use of statement caching***

When the structure of a query to be executed is known during application development, but will be executed with different parameters at runtime, it is best to use prepared statements, and pass parameters at runtime to these statements. This allows for caching of precompiled queries that can then be accessed through the WebLogic prepared statement cache, instead of being passed to the DBMS repeatedly and compiled there repeatedly.

The default value is zero – no statements or data are cached without user modification of the setting. The effectiveness of the cache size limit you chose for your application can be validated by comparing values of the "Prepared Statement Cache Hits and Misses" within the WebLogic JMX performance metrics – generally, the cache size should be equal to the number of commonly executed queries amongst all end user request types.

Summary

Applying rules of thumb such as these, within an organization that recognizes the value of testing early and often, can reduce the time spent in seemingly endless QA cycles, and greatly increase the production-readiness of the application. By dedicating resources to faithfully reproducing a staging environment that closely mirrors the production environment both in terms of hardware architecture and expected end user load, many performance bottlenecks can be detected, diagnosed, and resolved before causing production headaches and unsatisfied customers.

By employing an effective J2EE performance diagnosis tool, in both staging and production environments, costly guesswork and departmental fingerpointing can virtually be eliminated. An effective tool and problem-solving methodology allows for the efficient triage of problems to the correct functional expert and his or her silo resolution tools. Reducing the number of people involved in tracking performance problems at any given time reduces frustration, costs, time to problem resolution, and hence time to market. The end result: happier performance teams and end users.

Quest (Software) PerformaSure displays dynamic graphical maps of custom J2EE applications and how they utilize both the WebLogic Server resources and database management system on a per end-user request basis. Its unique Tag-and-Follow technology reconstructs the end-to-end transaction execution path across distributed J2EE systems, clearly identifying application-database performance bottlenecks, while relating application timing to application server resource metrics.

Quest PerformaSure Application-Database Performance Bottleneck Data

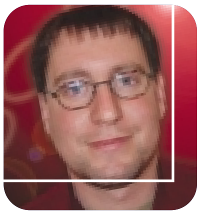
- **Custom Application Performance Map on a per request basis**
  Time spent and number of calls made to:
  - **Obtain and Release a DBMS connection**
  - **Prepare SQL statement & pass parameters**
  - **Execute statement**
  - **Manipulate the returned Resultset**
  - **Commit or roll back a database transaction**
  - **Exceptions thrown**
- **WebLogic JMX Resource Utilization Metrics**
  JDBC Connection Pools
  Prepared Statement Caches
  EJB Pools + Caches
- **DBMS Execute and Update times**
  SQL Statements and Stored Procedures .

Quest Software

http://java.quest.com/jcsv/wldj

STRUTS AND J2EE

INCREASED PRODUCTIVITY THAT'S ALWAYS DESIRED



BY **ARISTOTLE ALLEN**

AUTHOR BIO...

Aristotle Allen is a senior development analyst at PJM Interconnection LLC. Ari has worked in high-tech manufacturing, ASP, and utilities, and chooses to specialize in Java, especially J2EE.
http://aballen.phathookups.com

CONTACT...

aballen@ptd.net

Struts is a framework provided by Apache, designed to handle the presentation layer of your J2EE applications. The J2EE blueprints recommend that you use a Model 2 approach for your presentation layer, and Struts does just that for you. It doesn't try to re-create what is already available, nor is it unnecessarily complex. In fact, it leverages other J2EE technologies you may already know, like servlets, JSP, and JavaBeans, to provide a flexible presentation layer at a minimal cost. Best of all, when you write an application with Struts you'll find that it is easy to maintain and easy to extend because you used a framework rather than just hacking out tons of completely unrelated JSP pages.

The Model 2 pattern, also known as the model, view, controller (MVC) pattern, consists of three components (see Figure 1). The first is a "model," which is an object that represents your data and the state of your application; this object has no need for any logic, it just contains data. The second component is a "view," which is simply a presentation of the data to the user. The view does not change the data directly; it marshals data from the model to the user interface, or it sends user input to the controller. The controller is usually the most complex component of the three. It is responsible for receiving input from the user interface (UI), and for changing the data in the model. Beyond that the controller is where your logic resides, and can control many aspects of an application based on the model data and the user input.

If this paradigm isn't clear to you or seems a bit complex, just read on. In the next few paragraphs you'll see that it's really quite simple. Our example will be a simple form to change a password written with Struts.

Let's start with the model. The Struts component that implements a model is the ActionForm (see Listing 1). It's just a JavaBean with two additional methods. The first is the validate method, which performs simple validation on the data such as the string comparisons, or checking for required values here. The reset method initializes your form.

You may have noticed that over half the code in this simple three-variable form consists of accessors (getter methods) and mutators (setter methods). Struts 1.1 contains features that can help you rid yourself of this redundant code. Instead of extending ActionForm, just extend DynaActionForm. The DynaActionForm contains getters and setters for your data in the form of get(String) and set(String, Object). DynaActionForm also inherits the reset() and validate() methods so you can simply code them normally.

If you want to get rid of your ActionForm class altogether, Struts 1.1 contains a validator plug-in that will allow you to map validation rules to fields via XML configuration files. You can use predefined rules provided by Struts, or even write your own. The validator is actually from the Apache commons project, and was pulled into Struts to satisfy the need for validation without the code. For the sake of brevity I will not go into the details of the validator here. After all, my form is so simple I just want my validation method to verify that the passwords match. Suffice it to say that if you need to perform a validation that is already provided by Apache, it's the way to go.

The second component is the view. You can write this component simply by using JSP, but Struts offers much more for you to use in the view. There are six taglibs provided by Struts: HTML, logic, bean, nested, template, and tiles. Your bread-and-butter taglibs are HTML (obviously), bean, and logic (see Listing 2).

Each element of the HTML library will render HTML. The errors tag will print out an error to the screen. For example, if your ActionForm fails validation the ActionErrors returned will be printed out here.

The forms "form" tag starts a form and the action attribute matches the action element of an html form, except in this case you want to post to your controller, called an Action. The other HTML tags here are basic text fields and password fields. When they are rendered the values of their respective ActionForm properties are placed in the fields, and their input will be mapped automatically to your ActionForm properties. Remember that the ChangePassword-ActionForm is also a bean so you can use the bean library to access it. If you want to perform some special manipulation of the data, rather than using scriptlets you can use the logic library, which will let you iterate through lists and control blocks with easy-to-read JSP tags. Take note though that your view should contain only presentation logic, nothing more.

There is also another plug-in for the view, called the tiles plug-in. It allows you to create reusable "tiles" and layout "tiles" in JSP. Imagine being able to make a "BorderLayout" tile. Then include your header tile NORTH, and your footer tile SOUTH, by default. Each specific page overrides CENTER. I happen to like this plug-in, but it also requires its own configuration files and a lot more detail. I recommend you look at it, but hold off until your second Struts application.

The third component is the controller. With a Struts application you have two controllers. The "true" controller of the application is the ActionServlet, provided by Struts in the form of a servlet. This servlet is configured with an XML file that tells Struts how to map your action forms to JSP pages to Action classes. That configuration, however, cannot possibly contain all of the possible submissions and appropriate reactions. Since your users are going to input dynamic data, you need to write a class, which extends Action to handle that input. Your action works with the action servlet to control the application.

The action servlet is like an air traffic control center – it controls everything. The action class is like an airplane passing through the control center's air space. The plane has a destination, but it cannot get there without the control center. Conversely, the control center won't have anything to do without a plane to direct. It looks at the plane's destination and based on where it's located (its state), it may send the airplane along or it may tell it to land. Listing 3 is an example action.

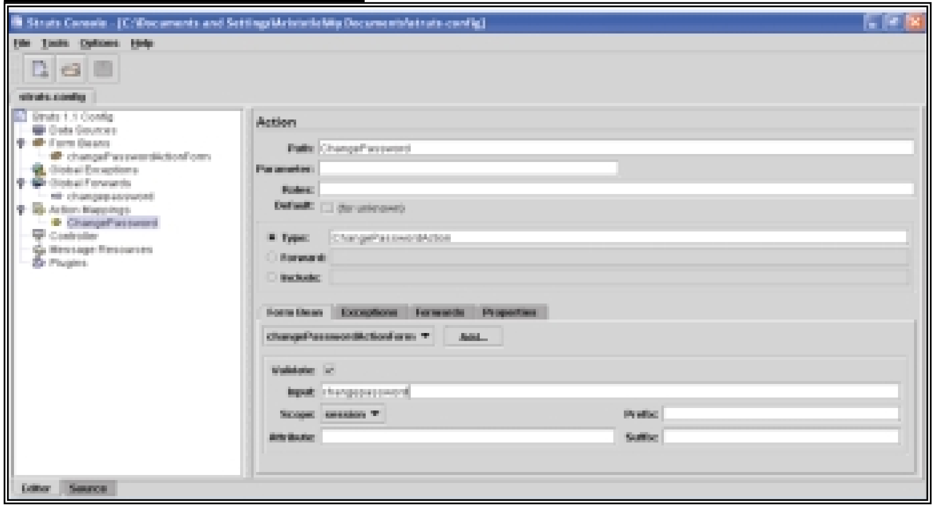
The only method you have to write is the execute method (called the "perform" method in Struts 1.0). This method returns an ActionForward, which is also defined in the XML configuration file. The ActionForward tells the action servlet where to direct the application to next. It allows you to redirect the user, based on user input, and it places the destination outside the code, again in the XML configuration file, so we can easily make changes to the application without recompiling anything.

The Action is where your application logic should exist. It is also where Struts meets your EJBs, messaging queues, JDOs, and so on. The trick here is to keep it simple. My changePassword method may contact the local LDAP, a session bean, or an application-specific database. Just put it in another method, or if there is a great deal of business logic being performed here, put it in another business object.

Now that you have your three components, I'll show you how they all fit together. It's all in the Struts configuration file. When you set up the action servlet the typical mapping is to "*.do". The only required parameter is "struts-config". This parameter should name where your XML configuration file is, such as "WEB-INF/struts-config.xml". There are also three other variables I recommend you set initially. Validate=true will turn on validation for your XML configuration file, so you will know if you put something incorrect into it. Debug=true and detail=2 will turn on a good deal of debugging information so you can see what the application is doing. These are good for development, but once your app is ready for production there you will probably want to turn them off (see Listing 4).

The configuration file contains several ele-

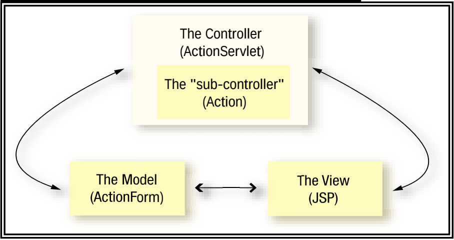
FIGURE 2



Path: ChangePassword
Parameter:
Roles:
Default: ☐ (for unknown)
● Type: ChangePasswordAction
○ Forward:
○ Include:
changePasswordActionForm ▾
Validate: ✓
Input: changepassword
Scope: session ▾
Prefix:
Attribute:
Suffix:

Struts console

FIGURE 1



Model, view, controller (MVC) pattern

ments: action mappings, message resources, form beans, forwards, plug-ins, and data-sources. This is where you define your forwards and loosely couple all of your components. Don't underestimate the configuration though. The components you have to write are pretty simple, but all it takes is one little typo in your configuration file, to keep it from deploying properly. Rather than editing this file by hand I recommend you use a tool such as the "Struts Console" by James Holmes at www.jamesholmes.com/struts (see Figure 2). It will help get your configuration files together quickly, and make maintenance easier too. It works with both Struts 1.0 and 1.1 DTDs, and it will even create tiles and validation and tiles XML files. Most important, it will help you avoid having to waste your valuable time finding a simple mistake in an XML file.

There really isn't much left for you to do. The framework is written for you, and is probably more than flexible enough to suit your needs. All you have to do is write a few lightweight components, set up your configuration file(s), wrap it all up in a WAR, and deploy it. Stick with the 1.0 features for your first few forms. Once you are comfortable with those you can add in some of the 1.1 features if you desire.

Conclusion

The more you use a framework like this the more you'll appreciate it. Here all of the repetitive parts of your navigation, validation, and display are already written for you. The applications using the framework are easy to maintain and to extend. You can use it to easily lay a Web front end on top of your J2EE infrastructure. You will have to spend a little time up front to learn the framework, but the payoff is worth it. Once you know the framework, you will be able to rapidly create applications that use it, and because of the bump you get in productivity you'll want to use it again.

Listing 1

```
import org.apache.struts.action.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class ChangePasswordActionForm extends
org.apache.struts.action.ActionForm {

  private String oldPass;
  private String newPass1;
  private String newPass2;
  public ChangePasswordActionForm() {
  }

  public ActionErrors validate(ActionMapping parm1, HttpServletRequest
parm2) {
    ActionErrors errors = null;
    if( parm2.getParameter("action").equals("Change Password") ) {
      if(! getNewPass1().equals(getNewPass2()) ) {
        errors = new ActionErrors();
        ActionError e = new ActionError("passwords.do.not.match");
        errors.add("errors.do.not.match", e);
      }
    }
    return errors;
  }
  public void reset(ActionMapping parm1, HttpServletRequest parm2) {
    setNewPass1("");
    setNewPass2("");
    this.setOldPass("");
  }
  public void setOldPass(String oldPass) {
    this.oldPass = oldPass;
  }
  public String getOldPass() {
    return oldPass;
  }
  public void setNewPass1(String newPass1) {
    this.newPass1 = newPass1;
  }
  public String getNewPass1() {
    return newPass1;
  }
  public void setNewPass2(String newPass2) {
    this.newPass2 = newPass2;
  }
  public String getNewPass2() {
    return newPass2;
  }
}
```

Listing 2

```
<%@ taglib uri="http://jakarta.apache.org/struts/tags-html" prefix="html"
%>
<html:errors/>
<html:form action="changePasswordAction.do">
<table cellpadding="20">
<tr><td>Choose a user</td><td><html:select property="user"><html:options
property="userList" labelProperty="userList"/></html:select></td></tr>
<tr><td>Enter Old Password</td><td><html:password
property="oldpass"/></td></tr>
<tr><td>Enter New Password</td><td><html:password property="new-
pass1"/></td></tr>
<tr><td>Confirm New Password</td><td><html:password property="new-
pass2"/></td></tr>
<tr><td>&nbsp;</td><td><html:submit value="Save"
property="action"/>&nbsp;<html:submit value="Cancel"
property="action"/></td></tr>
</table></html:form>
```

Listing 3

```
 public class changePasswordAction extends Action {
  public ActionForward execute(ActionMapping actionMapping, ActionForm
actionForm, HttpServletRequest httpServletRequest, HttpServletResponse
httpServletResponse) {
    DynaActionForm form = (DynaActionForm)actionForm;
    String action = httpServletRequest.getParameter("action") != null ?
httpServletRequest.getParameter("action").trim() : "";
    ActionForward forward = actionMapping.getInputForward(); //send to
change password action jsp
    if(! action.equals("") ) {
      if(action.equals("Save")) {
     changePassword(httpServletRequest.getSession().getProperty("user-
name", form.get("oldPass"), form.get("newPass1")));
          forward = actionMapping.findForward("password.changed");
      }
      else if(action.equals("Cancel") ) {
        form.reset(actionMapping, httpServletRequest);
      }
    }
    return forward;
  }
```

Listing 4

```
   <?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE struts-config PUBLIC "-//Apache Software Foundation//DTD
Struts Configuration 1.1//EN"
"http://jakarta.apache.org/struts/dtds/struts-config_1_1.dtd">
<struts-config>
  <form-beans>
    <form-bean name="changePasswordActionForm"
type="org.apache.struts.action.DynaActionForm"/>
  </form-beans>
  <global-forwards>
    <forward name="changepassword" path="ChangePassword.do"
redirect="false" />
  </global-forwards>
  <action-mappings>
    <action path="ChangePassword" type="ChangePasswordAction"
name="changePasswordActionForm" scope="session" input="changepassword" />
  </action-mappings>
</struts-config>
```

Informatica

www.informatica.com

The number of Web service business process (BP) specifications trying to make their way to standards status makes it difficult to tell who is doing what, especially given that many efforts are redundant.

The Race to Create Standards

THE ROAD TO MATURITY

BY YARON Y. GOLAND

This article makes sense out of the morass by classifying Web service BP specifications into four categories.

- ***Business analyst graphs:*** When business analysts write the high-level business logic for a BP, they need a standard way to communicate that logic, usually graphically, to each other and to BP programmers.
- ***Message choreography:*** When multiple BPs want to interoperate there needs to be an agreement on not only what messages are to be sent back and forth, which is provided by WSDL, but also the order in which messages are to be sent.
- ***Platform-independent business process programming languages:*** When writing the code to actually execute a BP it can be very compelling to use a solution that will run on Java, .NET, and everywhere else.
- ***Platform-specific business process programming languages:*** BP languages written directly for platforms like J2EE get a jump on everyone else by inheriting from mature, proven platforms.

We will explore each of these programming categories and explain what specifications are available in each category and their status. Please keep in mind that the status reflects the state of the specifications at the time of this writing (April 2003).

Business Analyst Graphs

The job of business analysts is to express the business logic that a BP is supposed to implement. Business analysts typically are not programmers so they express their logic through a combination of graphs and prose. It is the job of BP programmers to take the business logic provided by the business analysts and create running code.

Specifications such as BPDM and BPMN (see Table 1) are intended to make it easier for business analysts to communicate with each other and with BP programmers through UML-based models. Such models both standardize notation amongst business analysts as well as provide a foundation by which code skeletons can be automatically generated for BP programmers from business analyst's graphs.

BPDM has not yet begun substantive work and there are concerns about how appropriate BPMN is for business analysts. BPMN is so complicated it almost seems to be targeted at BP programmers.

Message Choreography

For multiple BPs to interoperate, all participants in the conversation must agree not only as to what messages are to be sent, which is specified by WSDL, but also the order in which they are expected. For example, a WSDL description says that a purchaser can receive purchase orders and a shipper receives shipping orders, but it cannot say that once a purchaser receives a purchase order the next thing the purchaser will do is send out a shipping order.

Message choreography specifications (see Table 2) tackle this problem by providing mechanisms to describe messaging order. There are two general approaches to message choreography – Turing-complete and declarative.

Turing-complete solutions, typified by BPEL Business Protocols and BPML Abstract Processes, use programming language to describe message choreography. Declarative solutions use directed graphs.

The trade-off between the two types of solutions is that the Turing-complete solutions can describe the message choreography in more detail than a declarative solution but at the cost of making the message choreography harder to understand and possibly more brittle. To understand a Turing-complete message choreography you must literally read and understand source code. In addition, one of the keys to interoperability is to know what not to say; in some ways Turing-complete solutions enable you to say too much. In practice the two solutions are complementary in that you can create a higher-level description using a declarative solution and then fill in details using a Turing-complete solution.

Although WS-Chor has taken WSCI, which is the foundation of BPML, as an input, it is not yet clear if WS-Chor will try to standardize a Turing-complete or a declarative message choreography solution. At the time of this writing the issue was still open within the working group.

AUTHOR BIO

Yaron Y. Goland is a senior principal technologist in the Office of the CTO at BEA, with responsibility for BP standards. Yaron is a coauthor on BPEL and BEA's representative to the W3C Choreography Working Group.

CONTACT...

ygoland@bea.com

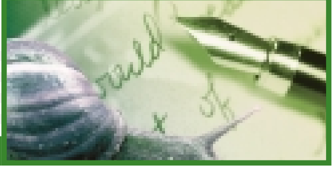


TABLE 1

SPECIFICATION	ORGANIZATION	STATUS
Business Process Definition Metamodel Request for Proposal (BPDM)	Object Management Group (OMG)	Effort started in January 2003
Business Process Modeling Notation (BPMN)	Business Process Management Initiative (BPMI)	At version 0.9

Business Analyst Graphs

TABLE 2

SPECIFICATION	ORGANIZATION	STATUS
Business Process Execution Language for Web Services (BPEL): Business Protocols	OASIS	Call for participation released in April 2003
Business Process Markup Language (BPML): Abstract Processes	BPMI	Standardized by BPMI but may be submitted to a more widely recognized standards organization
W3C Choreography Working Group (WS-Chor)	World Wide Web Consortium (W3C)	Formed in March 2003. This group contains the Web Service Choreography Interface (WSCI)

Business Choreography

TABLE 3

SPECIFICATION	ORGANIZATION	STATUS
BPEL - Executable Processes	OASIS	Call for participation released in April 2003
BPML - Executable Processes	BPMI	Standardized by BPMI but may be submitted to a more widely recognized standards organization
JSR 207 Expert Group: Process Definition for Java (PD4J)	Java Community Process (JCP)	Formed in March 2003
WS-Chor	W3C	Formed in March 2003

Business Process Programming Language

Business Process Programming Language

BP code is typically written using a dedicated BP programming language (see Table 3), usually via a graphical tool. Two flavors of BP programming languages are being developed: platform independent and platform specific.

The term *platform independent* is generally taken to mean a BP execution language that can run on top of J2EE or .NET. But the term "independent" is a misnomer. It would be more accurate to say "a new platform that runs on top of existing platforms." For example, BPEL and BPML, both "platform independent" solutions, define a full XML-based programming language, an exception handling model, a compensation model, a threading model, a message handling model, a type system, and a process life-cycle model. By any reasonable definition they constitute platforms in their own right.

Both BPEL and BPML are still very new so their feature set is limited. For example, they can only deal with XML values, which can only be manipulated using XPATH 1.0. They don't have easy to use models for communicating with local resources such as databases and file system; they don't have solutions for issues such as deployment or access control, etc. But these limitations only reflect their relative immaturity rather than oversight or design flaw. Over time, as the specifications mature, they will inevitably grow in features and complexity.

JSR 207 (Process Definition for Java; PD4J), which was proposed and is being led by BEA, typifies a platform-specific standard. PD4J takes the full-featured, fully developed, standardized J2EE platform and provides a few simple extensions to enable BP programming. While it is possible to write BPs on top of J2EE today, PD4J adds BP-specific extensions to J2EE in order to make it easier to write BP programs.

BPEL in particular has been positively received by efforts such as Siebel's Universal Application Network (UAN), which will use BPEL to write templates that describe best practices for certain types of BPs, such as a BP for entering information about a new employee into multiple independent data stores in a company. By using BPEL, Siebel's UAN BPs can potentially be imported and run across a wide variety of platforms. Since Siebel partners with all the major application vendors, this flexibility is desirable and the cost in terms of limited functionality is acceptable.

PD4J, scheduled for completion in March 2004, will be the officially approved Java standard for BP programming. In addition, because it benefits from J2EE,PD4J will have access to more features and be more mature than BPEL or BPML. Therefore, PD4J will almost certainly be the richest and most portable BP programming solution available.

WebLogic Integration 8.1 will ship with support for Java Processes (JPD), which is the foundation technology for PD4J. When the PD4J expert committee releases the official PD4J standard BEA will migrate JPD programs to the standardized format.

Users will be able to move between platform-independent and platform-specific business process programming languages via import/export functions. For example, after the release of WLI 8.1, BEA will provide users with tools to enable the import and export of BPEL process definitions to and from JPD and upon standardization, PD4J

It is not clear if WS-Chor will attempt to create a platform-independent programming execution language. The issue is being actively discussed within the working group.

Conclusion

Looking at the status of the various listed specifications described above you can see that the race to create Web service–based BP standards is just starting. Unlike more mature business process standards such as ebXML there is still an enormous amount of work to be done before Web service business processing can be said to be mature. In the meantime, you are encouraged to review the various specifications and to evaluate them in terms of their technical completeness as well as how open a standard they might eventually become.

References

- *Business Process Definition Metamodel - Request for Proposal (BPDM):* www.omg.org/cgi-bin/doc?bei/2003-01-06
- *Business Process Execution Language for Web Services (BPEL):* <http://dev2dev.bea.com/technologies/webservices/BPEL4WS.jsp >http://dev2dev.bea.com/technologies/webservices/BPEL4WS.jsp
- *Business Process Markup Language (BPML):* www.bpmi.org/specifications.esp
- *Business Process Modeling Notation (BPMN):* www.bpmi.org/specifications.esp
- *JSR 207 Expert Group - Process Definition for Java (PD4J):* http://jcp.org/en/jsr/detail?id=207
- *W3C Choreography Working Group (WS-Chor):* www.w3.org/2002/ws/chor/
- *Web Service Choreography Interface (WSCI):* www.w3.org/TR/wsci/

JavaOne

www.java.sun.com/javaone/sf

JavaOne

www.java.sun.com/javaone/sf

BEA WebLogic Workshop 8.1

J2EE APPLICATION DEVELOPMENT MADE EASY

Reviewed by Joseph Mitchko

Last year, BEA introduced WebLogic Workshop, a revolutionary product based on declarative annotations that took away most of the pain and aggravation of developing J2EE-based Web services on the WebLogic Application Server platform. Not being satisfied with just Web services, BEA extended this technology, with WebLogic Workshop 8.1, to include Web applications, portals, and other J2EE integration-based applications.

New Features

For development of loosely coupled applications that can maintain their public contract while underlying data structures change, Workshop 8.1 includes support for XML Schema and XQuery Mapping. Based on the XQuery XML standard, the visual mapping tool allows you to map XML elements to Java data elements with simple point-and-click operations (see Figure 1). In addition to straight one-to-one mapping, you can also use a number of built-in XQuery functions such as "concat," allowing you to combine various fields into one. All of the hard work of handling the complex data transformations is performed automatically.

In addition to XQuery, Workshop 8.1 provides support for XMLBeans, a strongly typed Java object interface for XML data that allows a developer to manipulate raw XML data using the productivity and flexibility benefits of the Java language.

Java Controls

WebLogic Workshop 8.1 includes a number of new Java Controls to help you connect to various IT assets, including FTP, e-mail, Tuxedo, Portal, and Integration Controls. Remember that as a developer, interacting with a Java Control is as easy as setting properties and handling events. The control itself handles all the hard parts, especially important as you begin to interface with more complex products such as Tuxedo.

Developing Java Controls is easy using Workshop's visual development environment. For those not familiar with it, Java Controls provide a simplified interface for accessing information from external resources, including databases, external Web services, and EJB components. A Java Control provides familiar interfaces, including methods, events, and property settings. A Java Control, similar to a JavaBean, can be dropped into an application or Web service on the Visual Workshop IDE. Controls can be used in any platform and are easily reusable, thereby optimizing developer efficiency. All the developer needs to do is link the application logic to the control. In addition to the standard list of controls for database access, EJBs, and so on. Workshop now allows you to create your own custom controls for proprietary legacy applications, opening the door for custom third-party vendors to develop and sell Java Controls.

Framework Extensions

WebLogic Workshop also includes framework extensions for developing portal and advanced workflow–based applications that work in conjunction with WebLogic Portal 8.1 and WebLogic Integration 8.1, providing WebLogic developers with a single IDE for a variety of application types.

Web Service Improvements

WebLogic Workshop 8.1 now supports both RPC and document-literal Web services, including support with WS-Security making it easy to integrate with .NET-based Web services. Here I couldn't resist. Using the Order Entry Service example that comes with Workshop, I fired up Microsoft Visual Studio .NET and was quickly able to create a .NET client for the Workshop example and execute the service from a .NET ASP application. Amazingly, it all took under 5 minutes and worked the first time.

Java Pageflow View

One of the major new changes to Workshop is the introduction of a visual development interface for Java Pageflow (JPF) files. Based on the Struts model-view-con-

Company Information:
BEA Systems, Inc.
2315 North First Street
San Jose, CA 95131
Tele: 1.800.817.4232

Web:
www.bea.com
E-mail:
sales@bea.com

Download Information:
Free one-year development subscription at http://dev2 dev.bea.com/subscriptions/ index.jsp

Testing Environment:
OS: Windows-XP
Hardware: 1GHZ Athlon, 1G RAM

Wily Technology

www.wilytech.com

troller (MVC) architecture, the visual interface allows you to see the flow of a Web application including user action decision flow and business logic. Tag libraries and drag-and-drop wizards are included to help you bind information on each page to external data sources, including databases, Web services, and Java controls. Workshop automatically provides support for sessions and state management, making this one serious development IDE. Gone are the days when all of your JSP files were thrown into a jumbled directory heap.

IDE Improvements

For those of us who need all the help we can get when coding and debugging our Java code, WebLogic Workshop 8.1 comes with a number of IDE improvements to help us identify and correct troublesome code. I especially like the red lines on the editor's vertical scroll bar that identifies the location of coding errors.

The WebLogic Workshop 8.1 IDE is well organized and provides you with various views, editors, property panels, etc., to assist you in your development work (see Figure 2). When using prebuilt Java Controls to your back end, you can literally create a Web service as fast as you can drag and drop Java Controls and methods in the design view panel.

What makes Workshop so powerful is that at any time you can compile, deploy, and test your code, all from the same IDE – allowing for an iterative "write-and-run" approach to development. This is something that would not be possible in the iterative sense if you had to handle all of the configuration and deployment details yourself. One aspect of the test browser that I thought might have been improved on for this release is having Workshop automatically create a test form for the Web service request. You still have to manually edit a skeleton version of the SOAP request in an edit box, something that can be rather tedious and error prone, especially if the number of parameters in the request grows.

Platform Support

WebLogic Workshop 8.1 is supported on both Windows 2000 and XP operating systems. For Linux users, it is also supported on Red Hat Advanced Server 2.1. For Sun Microsystems 8 and 9 and HP-UX 11.0 and 11i, only runtime versions (deployed application and Web services) are supported.

WebLogic Workshop 8.1 increases the productivity level of your development team by handling most of the hard-core J2EE "plumbing" work. Using BEA best practices, Workshop performs EJB configuration and deployment, JMS setup and configuration, and other tedious and unexciting activities associated with J2EE development. All of this work takes plenty of CPU and memory, so make sure that your development workstation is up-to-date.

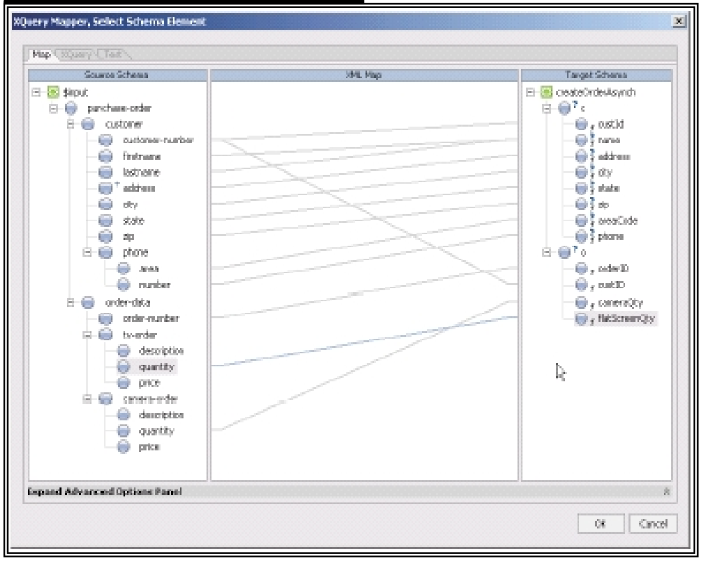
Installation

As usual, installation was very straightforward, although I forgot for the umpteenth time what I entered for the admin password. Make sure you jot it down; it's not as easy to recover the password as it used to be.

Conclusion

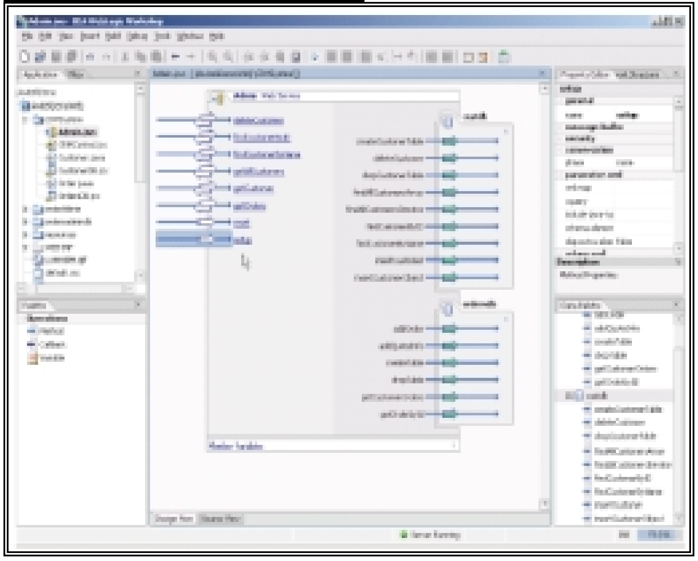
BEA WebLogic Workshop 8.1 is a powerful tool for developing sophisticated J2EE-based applications requiring integration with Web service–based assets both within the enterprise and abroad. Workshop 8.1 allows anyone with minimal Java coding skills to do some fairly complex J2EE development. Its power and ease of use take most of the drudgery out of J2EE development, and present a new level of competition for completing architectural platforms that make a similar claim – specifically Visual Studio .NET.

FIGURE 1



Visual mapping between XML and Java

FIGURE 2



BEA WebLogic Workshop 8.1 visual development environment

AUTHOR BIO

Joe Mitchko is a technology specialist working for a leading Internet service company and is product review editor and contributing writer for *Web Services Journal*.

CONTACT...

jmitchko@rcn.com.

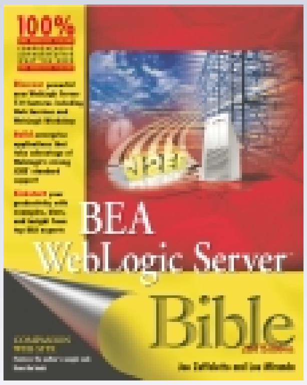
Quest Software

http://java.quest.com/dd/wldj

FROM CHAPTER 4:

Creating a WebLogic Environment

BY **JOE ZUFFOLETTO & LOU MIRANDA**



BEA WebLogic Server Bible, 2nd Edition
Joe Zuffoletto
Lou Miranda
ISBN: 0-7645-2602-2

Setting Up a WebLogic Environment

Our goal in this part of the book has been to introduce you to WebLogic Server's many capabilities and to help you understand how to field a team that can effectively take advantage of them. We've also given you some ideas on how to design J2EE applications that can be deployed with WebLogic.

The rest of this book is hands-on, showing you in great detail how to configure WebLogic's features, how to write code that uses those features, and how to deploy and test that code. To derive the most benefit from this book, you need to install WebLogic Server and create a WebLogic environment to play in. This chapter shows you how to do so.

We start off by showing you how to install the simplest possible WebLogic development environment on a computer and verify that it is working correctly. We then show you how to connect WebLogic to two popular relational database systems, Oracle and Microsoft SQL Server, so that you can write and test database code. Finally, we give you some tips and techniques on how to create a WebLogic development environment that supports a complete application development workflow, from development to testing to final deployment.

We want to make this chapter concrete without making it too long, so we limit our discussion to WebLogic Server 7.0 running on the Windows 2000 Server operating system. WebLogic runs on a multitude of platforms, including several flavors of Unix, but showing how to install and configure WebLogic on each platform would take too much space. Many developers, especially individual developers, start with Windows 2000 anyway, so we think the greatest number of readers will benefit if we focus on that platform. Knowledgeable Unix developers can easily adapt the instructions we give here to the version of Unix that they are using. The steps to install WebLogic on those platforms are fairly similar and well-documented in the WebLogic manuals.

After you have finished reading this chapter, you will have a fully functional copy of WebLogic Server running on your machine, and you'll be ready to conquer the material in the rest of this book. So let's get to work!

Configuring a Computer for WebLogic Development

WebLogic Server's published hardware requirements are modest and sufficient for one developer working with one computer. The minimum requirements for an Intel-based, Windows system are the following:

- ***Hardware:*** Pentium 200MHz or faster, 236MB free disk space (plus 170MB for temporary installation files), 256MB RAM
- ***Operating System:*** Microsoft Windows NT 4.0 or Microsoft Windows 2000

Obviously, the more capable your computer, the more productive you will be. For example, Lou runs WebLogic on a desktop running Windows 2000 Professional (this is certified only for development; Windows 2000 Server is certified for production). This machine has a 733MHz Pentium III chip, a 40GB hard drive, and 640MB of RAM. In addition to WebLogic Server, it comfortably supports his Oracle 8i installation, so he can develop and test database code.

Note that the stated minimums are not at all sufficient for large production systems. Requirements for them will vary, depending on user traffic, application complexity, clustering configurations, and so on.

In addition to the hardware and software requirements, WebLogic Server 7.0 requires the following to be installed on the computer running WebLogic Server:

- ***Java Development Kit (JDK) 1.3***. As of this writing, JDK 1.3.1_03 is bundled with WebLogic 7.0 and is automatically installed by WebLogic's installer.
- ***Netscape 4.7.x and 6.2*** are supported on all platforms (Windows and Unix). Microsoft Internet Explorer 5.x or 6.0 are supported on Windows. You need one of these Web browsers to run the WebLogic Console application (used to administer WebLogic).

Note

Having a supported browser for the WebLogic console mainly has to do with the Java applet tree on the left. Using this applet is optional in WebLogic 7.0 – you can configure the console to display only the web pages (no Java applet) in the main console window instead. For example, you can use any browser on Mac OS X (which is BSD Unix under the covers) and have no trouble administering WebLogic servers through the console.

Premiering
June 2003
at
JavaOne

www.sys-con.com

SERVER FOCUS: RUNNING ENTERPRISE LINUX
LINUX
BUSINESS & TECHNOLOGY
SPECIAL PREMIERE ISSUE!
Linux Takes a Road Trip
A guide to laptop Linux
FREE CD!
Migrating Your Existing Servers to Linux
Maintaining and Running Oracle on Linux
What Skills Should Linux Experts Arm Themselves With?
Look at Dell's Latest Offerings

Millions of Linux Users One Magazine

Linux Business & Technology

There is no escaping the penetration of Linux into the corporate world. Traditional models are being turned on their head as the open-for-everyone Linux bandwagon rolls forward.

Linux is an operating system that is traditionally held in the highest esteem by the hardcore or geek developers of the world. With its roots firmly seeded in the open-source model, Linux is very much born from the "if it's broke, then fix it yourself" attitude.

Major corporations including IBM, Oracle, Sun, and Dell have all committed significant resources and money to ensure their strategy for the future involves Linux. Linux has arrived at the boardroom.

Yet until now, no title has existed that explicitly addresses this new hunger for information from the corporate arena. *Linux Business & Technology* is aimed squarely at providing this group with the knowledge and background that will allow them to make decisions to utilize the Linux operating system.

Look for all the strategic information required to better inform the community on how powerful an alternative Linux can be. *Linux Business & Technology* will not feature low-level code snippets but will focus instead on the higher logistical level, providing advice on hardware, to software, through to the recruiting of trained personnel required to successfully deploy a Linux-based solution. Each month will see a different focus, allowing a detailed analysis of all the components that make up the greater Linux landscape.

Regular features will include:

- *Advice on Linux Infrastructure*
- *Detailed Software Reviews*
- *Migration Advice*
- *Hardware Advice*
- *CEO Guest Editorials*
- *Recruiting/Certification Advice*
- *Latest News That Matters*
- *Case Studies*

SAVE 30% OFF!

REGULAR ANNUAL COVER PRICE $71.76

YOU PAY ONLY

$49.99

12 ISSUES/YR

*OFFER SUBJECT TO CHANGE WITHOUT NOTICE

SUBSCRIBE TODAY!

WWW.SYS-CON.COM
OR CALL
1-888-303-5282

LINUX EDGE conference&expo

June 3–5	LONDON
June 24–26	BERLIN
September	HONG KONG
October	CALIFORNIA

FOR ADVERTISING INFORMATION:
CALL 201 802.3020 OR
VISIT WWW.SYS-CON.COM

SYS-CON MEDIA
The World's Leading i-Technology Publisher

WebLogic Portal 8.1

BUILDING ON THE STRENGTH OF A CLASSIC

Reviewed by Jason Snyder

WebLogic Portal 8.1 Beta is out and builds upon the successful and well received WebLogic Portal 7.0. Portal 7.0 dominated industry reviews last year, winning many "best technology" awards (for example, Best Enterprise Portal Solution in the 17th annual Software & Information Industry Association (SIIA) Codie Awards) as well as recognition as the best portal product. Most important: Java developers loved it.

The keys to Portal 7.0's success were its open architecture, ease of use, and overall high-quality portal features such as personalization and interaction management. The architecture allowed an ease of integration with either BEA WebLogic Integration or other third-party business process providers. Its user interface allows an ease of look and feel updates, and a division of labor between GUI specialists and Java developers. Portal features such as personalization (delivering audience-specific content), entitlements (restricting access to the correct resources), dynamic navigation, and interaction tracking were intuitively designed in a way that simplified developer usage.

So how do you improve on a classic? First you build on its strengths. Portal 8.1 provides all of the benefits of Portal 7.0 while adding new features that improve integration with the rest of the WebLogic platform. More wizards and graphical component designers were added. BEA has also added more Java Controls, which provide a head start on application development by simplifying API or J2EE resource calls. In addition, Portal 8.1 now allows for the addition of multiple portals within the same overall Web application.

One of the biggest changes in WebLogic 8.1 involved the development environment updates to WebLogic Workshop, which has evolved into a complete developer tool capable of integrating Web services, Enterprise JavaBeans (EJBs), JavaServer Pages (JSPs), and busines processes into the portal environment. This provides a one-source tool for most areas of portal development. A debugging source code editor has also been added to Workshop. The numerous updates to Workshop itself are covered in a related article on page 28.

Portlet functionality has also been dramatically improved. The drag and drop creation of portlets and their components works well and greatly eases development. JSP portlets, Web services portlets, and Java Pageflow portlets can all be created at the click of a few buttons (and maybe a little typing). The portal development interface combines visual logic with vast utility, and provides what BEA calls "Converged Portal Development," using Workshop to build custom portals that combine business processes and Web services.

Portal Sample

The sample portal represents a fictional company called Avitek (see Figure 1). There is an initial sign-on page, "Avitek Inweb," followed by three tabs representing "Content," "News," and "Bookmarks."

A portal created by WebLogic consists of a single XML file, which is constructed automatically by Workshop when the Portal Designer is employed to build a portal. A snippet from the sample.portal Avitek example XML file is shown below:

```
  <netuix:desktop markupName="desk-
top" markupType="Desktop"
title="Avitek
     Intranet - A platform for
Information Sharing">
       <netuix:lookAndFeel
definitionLabel="avitek"
markupName="avitek"

markupType="LookAndFeel" skele-
ton="default" skeletonPath="

/framework/skeletons/"
skin="avitek" />
```

Luckily, you don't really care what this looks like as you are using WebLogic Workshop. To open the Avitek portal simply open the XML file. Workshop uses the XML file to determine the portal contents and presents the developer view (see Figure 2).

Let's go through what you see from the sample.portal view in Workshop.

BEA Systems, Inc.
2315 North First Street
San Jose, CA 95131
+1.800.817.4BEA (US toll free)
+1.408.570.8000 phone

Requirements:
BEA WebLogic Portal requires BEA WebLogic Server 7.0

Sales:
North American Sales
Ready to Purchase BEA Products
+1.800.817.4BEA (1.800.817.4232)
toll-free (24 Hours/Day)
+1.415.402.7250 fax

For Europe and Asia:
www.bea.com/framework.jsp?CNT=sales1.htm&FP=/content/about/contact/

On the upper left, there are two tabs labeled Application and Files. The Application tab has a directory structure and provides access to all of the modules that are currently part of the portalApp sample application. The Files tab lists the file names for all of the same directory modules.

Below that area is a Palette tab that lists available construction elements. Currently, Book and Page are showing. A Book provides the ability to add a collection of Pages to one portal area. A Page represents a specific area of the portal.

In the middle of the Workshop screen is the visual representation of the portal application being built. Files from the application and palette tabs can be dragged and dropped to this area when adding new functionality. You can see four pages within the sample application. The first page has two columns, each with a portlet. A portlet is an access point to an application within the portal. Within the sample, a Login portlet is on the left and a Dev2Dev portlet is on the right. Both use JSPs to provide their content.

Immediately to the right of the portal representation are the Property Editor and Document Structure tabs. The Document Structure tab provides a directory structured listing of the various components of the portal desktop. The Property Editor provides the ability to set specific values for each of these components. Key Property Editor attributes for a Page include Title, which specifies the name of the page; and Navigation, which identifies and type of navigation (tabs or text). Other properties exist, most are fairly well named and, after you look them up once or twice, remain in memory.

The Data Palette falls below the Property Editor in the sample screenshot. It lists the Portal data components. For the sample, all of the available portlets are listed. These can easily be dragged into any of the pages.

Finally, the tour of the sample portal concludes with the tri-tabbed section immediately below the footer of the portal representation. The tabs are Build, Output, and Content Preview. The Build tab shows the results of any application build process. The Output tab shows functional output that occurs given Workshop usage. For example, if you have the debugger on, the various debugger outputs are depicted. Dynamic content can be previewed with the Content Preview tab.

After reviewing the sample and working through a tutorial (see Reference section), I created some sample portals. Creating them is straightforward once you get the hang of it. First, create an application to contain the portal. Follow that with the project to build within. Then it's simply a matter of adding Books (collections of pages) or Pages (containers for portlets). Stylesheets can be included very easily as can specific JSPs for the header or footer.

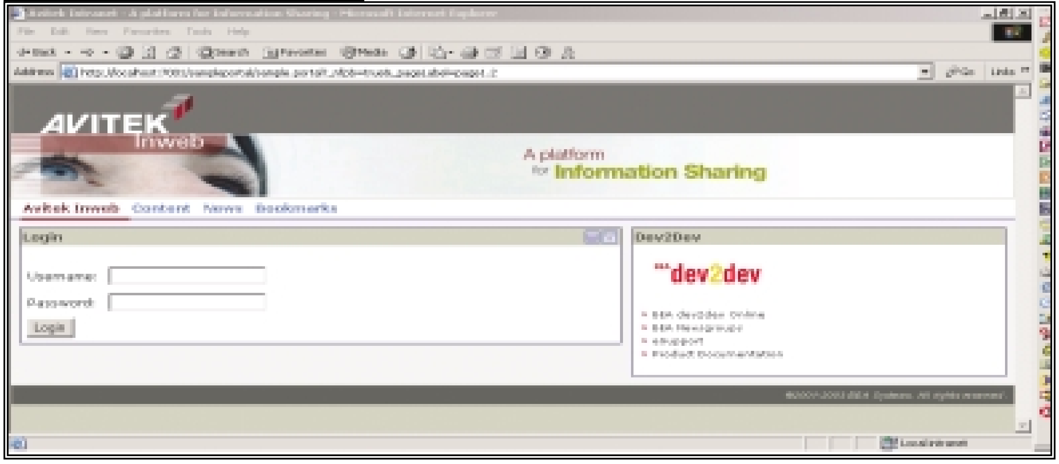
Most items employ a drag-and-drop mechanism for inclusion. The other main area for updating (aside from code) is the Property Editor tab on the right-hand side of the Workshop screen. The usage pattern I settled on was to select the item in the Document Structure tab and then update the properties in the Property Editor tab. You can also just select the item on the page, but sometimes I had trouble identifying specifically what I had selected.

The Workshop Test Browser allows a portal developer to see the outcome of their changes easily. Just remember to save your changes prior to refreshing it.

It is also possible to import any file, module, library, or project into your project or application. This allows a group of separate HTML developers, for example, to develop the code elsewhere and integrate it into the portal application. Once JSP or HTML has been imported into the project, creating portlets from the pages involves a drag-and-drop, which then calls the Portlet wizard.

When generating a Web service portlet, the interface provides three types of generated portlet code for access, which provides a degree of time savings. The choices available are Form, which takes parameters from an HTML form and calls the interface; Call Generation,

FIGURE 1



Avitek sign-on page

which creates a stubbed out portlet where you code in the source for the parameters; and Web service(s) Interfaces, which documents the interface usage.

The integration with Pageflows provides one of the more powerful aspects of WebLogic Portal. Pageflow provides a series of conditional statements provided with page outcomes. Possible outcomes include other pages, Struts ActionForms, or Workshop Java Controls. Java Controls allow logic to be reused as a simple interface to all developers. WebLogic Workshop includes several built-in controls, such as database access.

Deploying portal applications is relatively straightforward. After the server is started, access the console and deploy your entire portal application using straightforward directory options (similar to the deployment of JSPs or EJBs). WebLogic Portal 8.1 provides advanced administrator capability once the portal is created. Using the single XML file created by WebLogic Workshop when the portal is created, the Administration Portal allows the various components of the portal to be modified and reassembled. In addition, an administrator can set entitlements for user groups and individual areas of the portal.

Documentation

The online Workshop documentation provides a good understanding of what the different screen attributes mean, and provides a basis for some portal programming. But I had difficulty getting a sense of context from just Workshop or the sample alone. I recommend using the dev2dev eWorld tutorial for getting up to speed quickly after going through the Workshop documentation. Used in combination, I think you get a good understanding of what is possible (mostly from the Workshop documentation), and with the tutorial, a sample of potential use. The tutorial does not provide examples of all of the Portal features, but definitely provides enough to know where to look if it isn't covered.

Conclusion

BEA has done it again. WebLogic Portal 8.1 Beta is an excellent tool and greatly simplifies the potentially difficult development of a portal. It provides more wizards and has simplified usage, and makes portal development that much easier. Much of its usage is intuitive, and because of the overall consistency of the application, patterns for usage from one area can be reapplied to other areas, allowing faster learning. I was able to create JSP, Web service, and Pageflow portlets within the portals easily and soundly. I also added interaction management features through the handy designers without much trial and error.

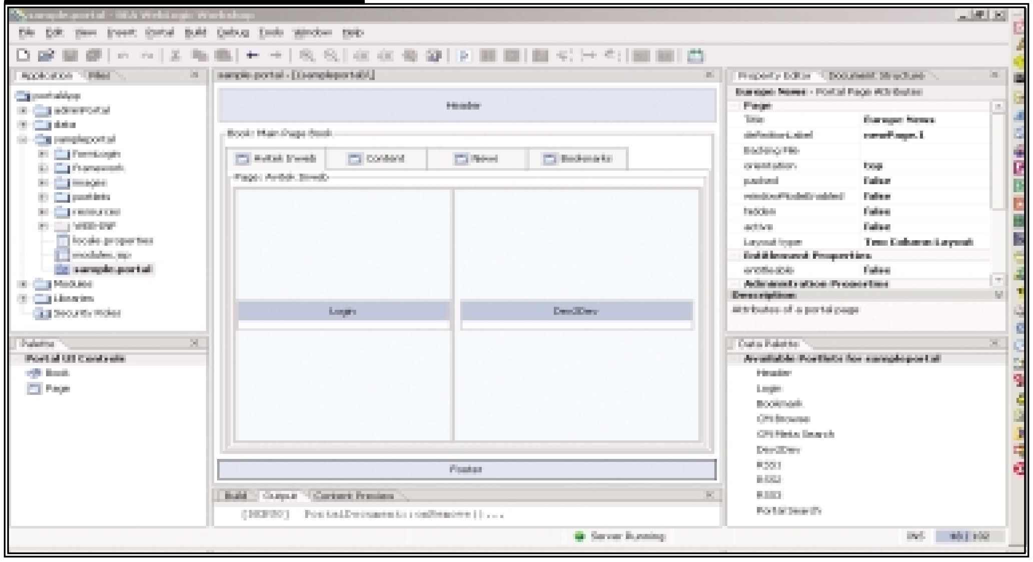
Most important, significant updates have been made to WebLogic Workshop. This article has only touched the surface of what Workshop provides. BEA has done an excellent job integrating the various Web development software packages into one platform. It is through the use of this platform that the real power of Workshop can be demonstrated.

References

The following sites should be helpful for anyone reviewing BEA Portal:

- eWorld BEA Portal 8.1 Tutorial: http://dev2dev.bea.com/resourcelibrary/tutorials/Portal_Lab.jsp
- BEA Portal 8.1 Release Notes: http://edocs.bea.com/wlp/docs81/relnotes/index.html
- Workshop Overview: includes a section on WebLogic Portal: http://edocs.bea.com/workshop/docs81/doc/en/core/index.html

FIGURE 2



Avitek developer view

AUTHOR BIO

Jason Snyder is an architectural expert for CSC Consulting in Boston, and has served as the lead architect for several J2EE development projects. He has over 10 years of experience in software development, OO design, and application architecture.

CONTACT...

jasonsnyder@townisp.com

International Web Services Conference & Expo

Web Services Edge WEST 2003

web services EDGE conference&expo

SEPT. 30 - OCT. 2, 2003

Santa Clara, CA

EXTENDING THE ENTERPRISE WITH WEB SERVICES THROUGH JAVA, Mac OS X AND XML TECHNOLOGIES

Featured technologies and topics will include:

- Focus on Java
- Focus on Web Services
- Focus on WebLogic
- Focus on Mac OS X
- Focus on XML

XML

WebLogic

Mac OS X

LINUX EDGE conference&expo

web services EDGE conference&expo

BOSTON

February 24-27, 2004

For more information visit

www.sys-con.com

or call

201 802-3069

Over 100 participating companies will display and demonstrate over 300 developer products and solutions.

Over 2,000 Systems Integrators, System Architects, Developers, and Project Managers will attend the conference expo.

Over 60 of the latest sessions on training, certifications, seminars, case studies, and panel discussions will deliver REAL World benefits, the industry pulse and proven strategies.

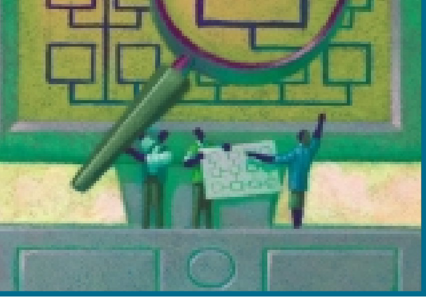
Contact information: U.S. Events: 201 802-3069 or e-mail grisha@sys-con.com

WebServices Journal · JAVA Developer's Journal · LINUX Business & Technology · XML Journal · SAMS · WebLogic Developer's Journal · wireless Business & Technology · CF Advisor · ColdFusion Developer's Journal · PowerBuilder Developer's Journal

WEB SERVICES EDGE WEST 2003 CALL FOR PAPERS NOW OPEN

Submit your papers online at: www.sys-con.com/webservices2003west

PRODUCED BY SYS-CON EVENTS



This article demonstrates how to create a gateway class for sending JMS messages generically with WebLogic. This is beneficial for any asynchronous messaging effort, and provides a basis for future JMS development. A generic access path for sending JMS messages is provided. We will also cover setting up a JMS topic within WebLogic 7.0. A sample message-driven bean is provided as well.

JMS and WebLogic 7.0

THE BENEFITS OF ASYNCHRONOUS PROCESSING

BY **JASON SNYDER**

Program communication is generally “synchronous,” meaning that the client sends a request and waits for a response (e.g., client/server or distributed object RMI call). Asynchronous messaging provides an alternative means of program communication in which the initiating program writes a message to the JMS session, and the WebLogic container ensures delivery. The initiating program does not get anything back. This allows the program to continue other processing or return control to the user.

Asynchronous messaging can vastly increase response times to the user because it does not require additional business logic processing time. However, because asynchronous processes cannot return an immediate response, separate asynchronous notification methods must also be employed. Potential notification solutions include e-mail, a status or work queue view, or a scrolling update bar within the application.

Many software projects are affected by poor performance. While asynchronous messaging can improve performance, JMS solutions are uncommon in today’s business environment. Few success stories have been observed beyond simple asynchronous data transfer, which may not even be JMS’s best usage.

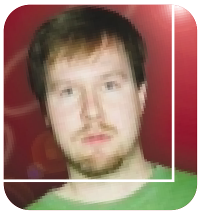
Part of the problem is that many designers and business requirements teams think only within the request-response box. The concept of “I push a button, and then the process starts, and it is done when I get a confirmation screen” is easily understood by all. While this process may be necessary for some applications, the asynchronous alternative should always be considered as well. It’s not as foreign a concept as it seems. Consider how e-mail works…your system is not locked between the time that an e-mail message is sent and received, and yet there is the confidence that it will arrive at its destination.

Using JMS

When can asynchronous processing be used? Almost anything that is currently processed synchronously can be developed asynchronously, especially processes that do not return anything to the user. Some appropriate applications include:

- ***Business processes requiring complex calculations and/or many synchronous reads and formulations:*** Developers know that these processes are very prone to causing performance problems. However, if these activities are run asynchronously, the user can continue processing while the complex business process is being resolved. The asynchronous solution to this business process also lends itself to being run on a dedicated server.
- ***Several separate, unrelated processes are kicked off by one main task:*** In the synchronous world, this must be managed carefully, and many times these tasks are simply kicked off in sequential order, although they needn’t be. Asynchronously, each process can be kicked off without waiting for the others to finish.
- ***Logging:*** Some logging solutions become fairly complex, involving writing to databases and separate files and sending e-mail. Asynchronously processing a logging solution allows it to be as complex as needed, without each enhancement impacting performance.
- ***Necessary communication to down servers:*** The JMS specification provides durable subscribers that allow the WebLogic container to hold the message until its destination becomes active.
- ***Large-scale processing of voluminous information:*** JMS automatically provides priority processing; thus, the lower priority items don’t threaten the ability of the system to begin processing more critical tasks.

Many more applications for asynchronous messaging exist, but I hope that at this point you are more interested in learning how it works. If not, it’s time to move on, as this article is about to get slightly more technical.



AUTHOR BIO

Jason Snyder is an architectural expert for CSC Consulting in Boston, and has served as the lead architect for several J2EE development projects. He has over 10 years of experience in software development, OO design, and application architecture.

CONTACT...

jasonsnyder@townisp.com

Quick JMS Overview

The JMS specification includes a series of Java interface definitions that provide a means for asynchronous messaging. WebLogic provides an implemention that allows the application server to send or receive JMS messages through the container. Message delivery is guaranteed.

There are two types of asynchronous messaging: point to point (P2P or PTP), and publish and subscribe (Pub-Sub). P2P involves a direct connection to one other listening party. It uses a queue as its primary sending mechanism. Pub-Sub allows the sender to post to a topic to which multiple listeners may have access. There is no direct connection between sender and receiver in the Pub-Sub model. JMS provides alternative implementations for each type. WebLogic allows P2P messages to be browsed prior to receipt and Pub-Sub messages to be sent even if the receiving client is not active at the time of attempted delivery.

The initial interface in the JMS API hierarchy is the ConnectionFactory. The QueueConnectionFactory or TopicConnectionFactory enables the client to open a connection to the JMS message broker. The JMS message broker serves as the messaging hub, through which all routing occurs. Coding the message broker is not necessary, since it is included with WebLogic. For a Pub-Sub message, a TopicConnection is opened. Similarly, P2P uses a QueueConnection.

Within the connection, a TopicSession (Pub-Sub) or QueueSession (P2P) is established. The session allows the client to provide transactional characteristics and acknowledgement ability. If the session falls within the transaction, then rolling back the client application will automatically roll back the JMS message. Committing it will allow the container to begin processing that message. On the receiving side, enabling a transaction will result in an acknowledgement being sent only after the message processing has been committed.

The acknowledgment mode determines the degree to which the message broker informs the client that its message has been successfully sent and received. Its options are:

- ***DUPS_OK_ACKNOWLEDGE:*** Acknowledges when processor is free, minimizes work, can result in a duplicate message.
- ***AUTO_ACKNOWLEDGE:*** Automatically acknowledges receipt after the MessageListener has completed processing.
- ***CLIENT_ACKNOWLEDGE:*** Acknowledges receipt when the acknowledge() method on Message object is called.

For a Pub-Sub application, the session then provides access to one or more TopicPublishers or TopicSubscribers, depending on sending or receiving intent. Each topic represents some logical grouping of messages that can be received. The TopicPublisher provides the ability to write to a topic and assign priority and message persistence. Message persistence refers to the ability of the message to survive system failure. Similarly, a P2P application provides access to QueueSender or QueueReceiver, which creates a queue.

Once the topic or queue is accessed, the actual creation of the message is the only remaining step. Messages may be a variety of different types, including TextMessage, MapMessage (for HashMap sending), XMLMessage (WebLogic specific), or ObjectMessage.

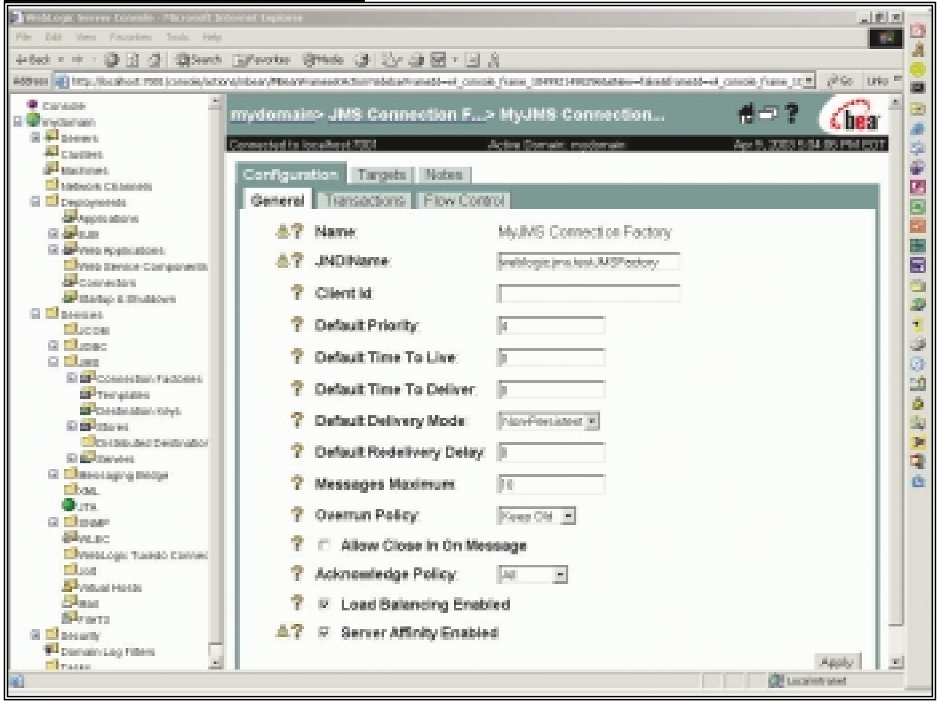
JMS messages have three distinct sections: header, properties, and the body. The header is used to determine routing and delivery information. Properties are optional and provide a degree of personalization to the message, as these properties can be reviewed by the message received and then acted upon prior to opening the message. The body is the message itself, which falls within a message-type structure.

Sample JMS Sender

A simple JMS program is presented in Listing 1. This program will allow another program to instantiate the JMSGateway class with an existing topic, and then call a sender method on the JMSGateway. A similar program for sending queue messages could also be created. The listing shows methods for sending simple text, but additional methods for sending other types like a HashMap or serializable Object could easily be added. It should be noted that this class is meant for the developer to become familiar with sending JMS messages, and not for direct production release.

On line 15, a topic name is passed into the constructor to allow the calling class to define the topic. The topic must be set up within the WebLogic domain. Within the constructor (line 23), a JNDI lookup for ConnectionFactory generates the connection to a JMS server, in this case my local WebLogic server. The connection factory is used to generate the connection between the client and server (line 25) using createTopicConnection. The connection is created in "stopped" mode – while stopped, the connection can transmit messages but cannot send them. After the session and

FIGURE 1


mydomain> JMS Connection F...> MyJMS Connection...

Configuration | Targets | Notes

General | Transactions | Flow Control

Name: MyJMS Connection Factory
JNDIName: weblogic.jms.test.JMSFactory
Client Id:
Default Priority: 4
Default Time To Live: 0
Default Time To Deliver: 0
Default Delivery Mode: Non-Persistent
Default Redelivery Delay: 0
Messages Maximum: 10
Overrun Policy: Keep Old
☐ Allow Close In On Message
Acknowledge Policy: All
☑ Load Balancing Enabled
☑ Server Affinity Enabled

Configuration page

topic have been successfully created, the connection is started (line 32).

From the gatewayConnection, the gatewaySession (line 27) that sends messages through the connection is created. Each session remains a single threaded context and retains messages until acknowledged (or conditions are met). The first parameter identifies the transaction level of the session. If transacted, the session will buffer messages until the client issues a commit or rollback. The second parameter represents the acknowledgeMode (if not transacted), which was detailed earlier.

After the session has been created, the topic JNDI name is looked up (line 29) and that topic is passed to a method on the session to create a TopicPublisher (line 31). The connection is started (line 32), and the session is now ready to process a message.

The client accessing this class should then call the sendTextMessage method (line 35) with the text message to be sent as the parameter. The other message types supported by WebLogic 7.0 are the StreamMessage, MapMessage, TextMessage, ObjectMessage, BytesMessage, and XMLMessage. This method could also be expanded to allow for the setting of priority or other header type information.

When the message processing has been completed, the close method (line 41) should be called. This releases local resources from the connection.

WebLogic 7.0 JMS Setup

The next step is to set up JMS within WebLogic. Luckily, WebLogic's admin console is fairly straightforward.

1. After bringing up the console, choose the JMS Service configuration from the menu on the left.
2. Choose Connection Factories to create the required ConnectionFactory for the JMSGateway class. Select the "Configure a new JMS Connection Factory…" option.
3. You will see a page with three tabs (see Figure 1). On the first tab, "Configuration," fill out the JNDI name with "weblogic.jms.testJMSFactory". The other values can be set as desired. Press the "Apply" button when you are finished. On the second tab, "Targets," select your test server. Select "Apply" and you should be able to view your ConnectionFactory on the selection menu.
4. You must next create a JMS server. Select the Servers folder from the JMS list. Select "Configure a new JMSServer…". Update the Configuration and Targets with the name of your JMS server and target server.
5. Select the "Configure Destinations" link from the bottom of the Configuration tab. Select "Configure a new JMSTopic…" and name the topic. Provide the JNDI name with whatever your test program passes to the JMSGateway. Leave the other values with defaults or update as you experiment.
6. That's it. Other options can be created as needed or as your JMS learning dictates.

A Sample Message-Driven Bean

Finally, a reader is needed for the text message being sent. The newest EJB, a message-driven bean, will be used (see Listing 2). A message-driven bean (MDB) defines a specific destination (topic or queue) in the deployment descriptor it is associated with. When that destination is written to, the container directs that message to an instance of the MDB. MDBs do not have home or remote interfaces, and exist as stateless services. They cannot be called directly, and do not return values or pass exceptions.

Because MDBs were designed for the purpose of accessing JMS Destinations, creating them is a snap. The SampleReceiverBean is a complete MDB that will take a TextMessage and print it out. In reality, you would want the MDB to kick off a real business process. The only method that involved any coding is the onMessage method (line 8). The onMessage method is where you would add your processing logic, keeping in mind that any reuse dictates that that logic occurs outside of the MDB.

Deployment descriptors must be set up. A sample ejb-jar (see Listing 3)and weblogic-ejb-jar (see Listing 4) are provided as reference, but setup should be as your deployment descriptors process dictates. Key values to keep in mind are acknowledge mode, and destination-type (topic or queue) in the ejb-jar and destination-jndi-name (topic or queue JNDI name) in the weblogic-ejb-jar. Remember to match the key values that you set up in the JMS server.

Summary

This brief synopsis is only an introduction, and does not provide all of the JMS information needed to perform detailed processing. I recommend you try the example and then perform some additional research to learn more. The following sites should be helpful:

- *Programming WebLogic JMS:* http://e-docs.bea.com/wls/docs70/jms/index.html
- *Designing Message Beans from BEA for 7.0:* http://e-docs.bea.com/wls/docs70/ejb/message_beans.html#1050867
- *Implementing JMS-based Web Services:* http://edocs.bea.com/wls/docs70/webserv/jms.html
- *O'Reilly summary article:* www.onjava.com/pub/a/onjava/2001/05/22/ejb_msg.html?page=1

Listing 1: JMSGateway code

```
import javax.jms.*;
import javax.naming.*;
import java.io.*;
import java.util.Properties;

1. public class JMSGateway
2. {
3.   private TopicConnectionFactory gatewayFactory;
4.   private TopicSession gatewaySession;
5.   private TopicPublisher gatewayPublisher;
6.   private TopicConnection gatewayConnection;
7.   private Topic gatewayTopic;
8.
9.   public final static String      J
JNDI_FACTORY="weblogic.jndi.WLInitialContextFactory";
10.   public final static String
JMS_FACTORY="weblogic.jms.testJMSFactory";
11.   public final static String WEBLOGIC_URL="t3://localhost:7001/";
12.   public final static String
INTER_CONTEXTS="weblogic.jndi.createIntermediateContexts";
13.
14. // Constructor for JMSGateway
15.     public JMSGateway(String topicName)

          throws Exception {
```

A LIMITED TIME SAVINGS OFFER FROM SYS-CON Media

SUBSCRIBE TODAY TO MULTIPLE MAGAZINES

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

RECEIVE YOUR DIGITAL EDITION ACCESS CODE INSTANTLY WITH YOUR PAID SUBSCRIPTIONS

3-Pack

Pick any 3 of our magazines and save up to $275.00

Pay only $175 for a 1 year subscription plus a FREE CD

- 2 Year – $299.00
- Canada/Mexico – $245.00
- International – $315.00

6-Pack

Pick any 6 of our magazines and save up to $350.00

Pay only $395 for a 1 year subscription plus 2 FREE CDs

- 2 Year – $669.00
- Canada/Mexico – $555.00
- International – $710.00

9-Pack

Pick 9 of our magazines and save up to $400.00

Pay only $495 for a 1 year subscription plus 3 FREE CDs

- 2 Year – $839.00
- Canada/Mexico – $695.00
- International – $890.00

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

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

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

☐ **Linux Business & Technology**

U.S. - Two Years (24) Cover: $143	You Pay: $79.99 /	Save: $63 + FREE $198 CD
U.S. - One Year (12) Cover: $72	You Pay: $39.99 /	Save: $32
Can/Mex - Two Years (24) $168	You Pay: $119.99 /	Save: $48 + FREE $198 CD
Can/Mex - One Year (12) $84	You Pay: $79.99 /	Save: $4
Int'l - Two Years (24) $216	You Pay: $176 /	Save: $40 + FREE $198 CD
Int'l - One Year (12) $108	You Pay: $99.99 /	Save: $8

☐ **WebLogic Developer's Journal**

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

☐ **Java Developer's Journal**

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

☐ **ColdFusion Developer's Journal**

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

☐ **Web Services Journal**

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

☐ **Wireless Business & Technology**

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

☐ **.NET Developer's Journal**

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

☐ **WebSphere Developer's Journal**

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

☐ **XML-Journal**

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

☐ **PowerBuilder Developer's Journal**

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

OFFER SUBJECT TO CHANGE WITHOUT NOTICE

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

SYS-CON MEDIA

```
  Properties env = new Properties();

  // provide JNDI properties for BEA
  env.put(Context.INITIAL_CONTEXT_FACTORY,JNDI_FACTORY);
  env.put(Context.PROVIDER_URL,WEBLOGIC_URL);
  env.put(INTER_CONTEXTS,"true");

  InitialContext jndi = new InitialContext(env);

  // Look up JMS connection factory
  gatewayFactory =
     (TopicConnectionFactory)jndi.lookup(JMS_FACTORY);

  // Create the JMS connection
  gatewayConnection =
      gatewayFactory.createTopicConnection();

// Create the JMS session object
  gatewaySession =
      gatewayConnection.createTopicSession(false,
Session.AUTO_ACKNOWLEDGE);

  // Look up the JMS topic
  gatewayTopic = (Topic)jndi.lookup(topicName);

 // Create the JMS publisher
gatewayPublisher =
      gatewaySession.createPublisher(gatewayTopic);

 gatewayConnection.start();
}

/* Create and send text message using the gatewayPublisher */
protected void sendTextMessage(String pTextMessage) throws
JMSException {
  TextMessage gatewayMessage = gatewaySession.createTextMessage();
  gatewayMessage.setText(pTextMessage);
  gatewayPublisher.publish(gatewayMessage);
}

/* Close the JMS connection */
public void close() throws JMSException {
        gatewayConnection.close( );
 }
}
```

Listing 2: Sample Message Driven Bean

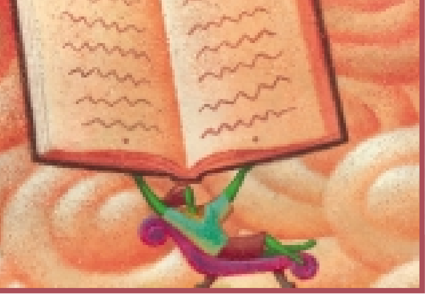
```
import javax.ejb.*;
import javax.jms.*;
import javax.naming.*;

public class SampleReceiverBean implements MessageDrivenBean,
MessageListener {
   MessageDrivenContext messageDrivenContext;
   public void ejbCreate() throws CreateException {
   }
  public void ejbRemove() {
  }
// This is the key onMessage method
  public void onMessage(Message sampleMessage) {
    if(sampleMessage instanceof TextMessage)
   {
       String textEvent = (String) sampleMessage.toString();
       System.out.println("Received text message:"+textEvent);
     }
     else
   {
       System.out.println("Incompatible message type");
     }
   }

  public void setMessageDrivenContext(MessageDrivenContext
messageDrivenContext) {
     this.messageDrivenContext = messageDrivenContext;
   }
}

ejb-jar
```

Listing 3: <ejb-jar>

```
    <enterprise-beans>
        <message-driven>
            <display-name>SampleReceiverBean</display-name>
            <ejb-name>SampleReceiverBean</ejb-name>
            <ejb-class>sample.SampleReceiverBean </ejb-class>
            <transaction-type>Container</transaction-type>
            <message-driven-destination>
                <destination-type>javax.jms.Topic</destination-type>
            </message-driven-destination>
        </message-driven>
    </enterprise-beans>
</ejb-jar>
```

Listing 4: weblogic-ejb-jar

```
<weblogic-ejb-jar>
    <weblogic-enterprise-bean>
        <ejb-name>SampleReceiverBean</ejb-name>
        <message-driven-descriptor>
            <destination-jndi-name>Sample</destination-jndi-name>
            <initial-context-factory>True</initial-context-factory>
            <provider-url>t3://localhost:7001</provider-url>
            <connection-factory-jndi-name>weblogic.jms.testJMSFactory
            </connection-factory-jndi-name>
        </message-driven-descriptor>
        <clients-on-same-server>True</clients-on-same-server>
    </weblogic-enterprise-bean>
</weblogic-ejb-jar>
```



BOOK EXCERPT

FROM CHAPTER 8:

Deriving Application Infrastructure

BY **JATINDER PREM**

BEA WebLogic Platform 7
Jatinder Prem
Bernard Ciconte
Manish Devgan
Scott Dunbar
Peter Go
ISBN 0-7897-2712-9

AUTHOR BIO

Jatinder Prem is the founder and CTO of ObjectMind Inc. (www.ObjectMind.com), a start-up company focused toward providing creative socio-technical solutions to organizations that need assistance in building J2EE enterprise solutions on the BEA WebLogic Platform.

CONTACT...

prem@objectmind.com

Deriving an Application Infrastructure Strategy

For an organization to economically support an evolving application portfolio, an application infrastructure strategy must exist. The right application infrastructure strategy ensures that an organization can accommodate short-term tactical and long-term strategic business considerations efficiently and effectively, the byproduct being technical agility. However, the difficulty lies in deriving an application infrastructure strategy that works for your organization.

An application infrastructure strategy is always based on a series of activities that must be culturally and technically acceptable, and hence sustainable by an organization. The activities discussed here have been derived and distilled from an accumulation of research, and consist of the following:

- Recognizing *infrastructure patterns*, which are used to resolve business requirements.
- Developing a *technology taxonomy*, which identifies the emerging, critical, inconsistent, and redundant technologies within an organization.
- Identifying the *Enterprise Application Integration* (EAI) types, challenges, successes, and failures within an organization.

An application infrastructure team must be formed that includes senior socio-technical staff as well as IT and application architects from different technology-dependent segments of an organization. The term socio-technical staff refers to people who are qualified to synthesize usability engineering with software design and development. The formation of this team is the first step in creating a visible organizational framework to continually evaluate an organization's application infrastructure against emerging technologies.

Recognizing Infrastructure Patterns

In general, a *pattern* describes a problem that occurs repeatedly in a given environment over time and then a solution to that problem that can be reused for that problem's context. When this general definition of a pattern is applied to the context of application infrastructure, an *infrastructure pattern* is the information and knowledge[md]the “what is” and “how to”[md]that are characteristic to an existing class of applications, which are identified to facilitate solution reuse for future applications in the same class. Infrastructure patterns are blueprints to how business problems can be cost effectively resolved through the rapid mapping of business requirements to infrastructure designs. An infrastructure pattern captures experience, best practices, and the end-to-end reuse of technology infrastructure for a given type of business pattern.

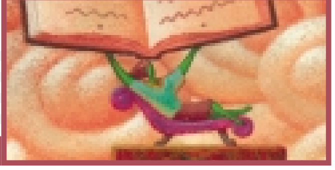
CATALOGING ORGANIZATIONAL INFRASTRUCTURE PATTERNS

Technology-based business solutions should always be driven by business requirements as opposed to technology. Infrastructure patterns also should be business driven. To recognize infrastructure patterns, you have to

1. Observe and categorize your business use cases for your organization's application portfolio according to a common theme for the type of business problem they solve.
2. Extract the following elements that have been applied to implement the associated business application:
   Technology infrastructure (physical, functional, and interface)
   Challenges and best/worst practices (if available) Project plan, including the predicted and actual cost model
   Types of skills and resources employed
3. Develop an infrastructure pattern using the elements that are common denominators to a specific type of business application.

The objective is to categorize and catalog the infrastructure patterns that coexist within your organization. Even though organizations have unique business application scenarios, you can leverage some common infrastructure patterns:

- The *Transaction category* supports business



use cases requiring read/write access to data records.
- The *Publish category* supports business use cases for read-only access to information and data, such as data warehousing applications and viewing files from a Web browser.
- The *Collaboration category* supports business use cases for person-to-person communication centered on shared documents or groups of documents, as opposed to strictly data (Transaction category).

The Advantages of Cataloging Infrastructure Patterns

After you have categorized and cataloged your organization's application portfolio, you can use the derived infrastructure patterns as a blueprint to develop your future applications. By using an internally recognized and established infrastructure pattern, you can

- Leverage architecture, technology, components, products, and the infrastructure configuration to applications that fall into a specific pattern.
- Efficiently map business requirements to proven IT business solutions.
- Estimate project-related development, implementation, and support costs.
- Estimate the length of a software development life cycle.
- Comprehend the challenges and skilled resources required.
- Embrace any best practices and turn bad practices into learning experiences.
- Recognize the skilled resources and responsibilities associated with a pattern.

Developing a Technology Taxonomy

A technology taxonomy can categorize and subdivide an organization's technology. By defining the rules for a category's membership and its boundaries, the infrastructure team can develop a common syntax and semantics for classifying an organization's common set of technologies.

For an organization to have a successful technology taxonomy, the taxonomy must be stable and not open to misinterpretation. Follow these guidelines:

- Ensure that all technologies relevant to the infrastructure derivation effort are categorized.
- Ensure that all boundaries for the technology categories and subcategories are clearly and semantically defined.
- Validate whether the category and subcategory definitions will stand the test of time.

  Through the development of an in-house technology taxonomy, an organization can
- Identify major technology trends
- Identify currently deployed technologies and component architectures and how they are employed toward business applications
- Understand why and when technologies should be combined
- Recognize which technology areas are mission critical

Identifying Enterprise Application Integration (EAI) Types

Much of the application infrastructure to support Enterprise Application Integration (EAI) activities today is focused around preventing application stovepipe scenarios, by enabling business events (transactions) in one application to be available to other applications. However, sharing business events among applications is just one type of integration, and it does not meet other types of integration needs that are becoming more critical to an overall organizational application infrastructure strategy. Examples of other types of integration include Analytic, Directory, Content Management, and Portal, which are all quite prevalent in organizations today.

Implementing the Software Platform Solution to Application Infrastructure

Taking into consideration everything that has been discussed so far, the best approach to implementing any application infrastructure solution is to take the software platform approach, where your business applications and their supporting technologies are part of a related, interlocking software architecture rather than marriages between independent initiatives.

A software platform approach involves the following layers:

- **Platform –** Groups interoperable application infrastructure solutions into their technical domains[md]for example, application server, databases, and integration servers.
- **Services –** Shift the responsibility for certain nonfunctional services from the application domain to the application infrastructure domain, so it can be shared by multiple applications. The objective of services is to minimize the coupling between the infrastructure and application domains.
- **Patterns –** Enable business requirements to be mapped to existing infrastructure designs that provide an end-to-end component or service-based solution for a specific class of application. Infrastructure designs are based on the application infrastructure solutions selected to form the platform.

In the software platform approach, only those application infrastructure solutions that meet current and future business requirements are part of the Platform layer. All others are considered part of an integration initiative. Even though your application infrastructure foundation might be reduced to solutions from a few vendors, perhaps one for each technical domain, there are advantages to this approach:

- Development and management of business solutions are simplified because the number of technologies that an organization must support is greatly reduced.
- The life of existing technology investments is extended because an organization can integrate legacy technology with a modern foundation that developers can use to create composite applications.
- Time-to-market of business solutions is accelerated by leveraging skill sets within a focused and proven technology selection.

By adopting a software platform approach, organizations can develop enterprise business applications through a common set of infrastructure services connected, aggregated, and presented by a common presentation framework. Each technical domain of the platform is based on the open standards that exist for that domain, which not only preserves an organization's investment in that technology, but also enables the technical domains to be interoperable so that composite applications can leverage all the functionality the platform provides. However, it is critical for the software platform to collectively provide Quality of Service (QoS)[md]Reliability, Availability, Scalability, and Trusted Security .

I have a rather deaf and rather elderly grandmother. She is a lovely woman and can spend hours telling tales – sometimes fascinating and sometimes… well, less fascinating – about times past.

Listen Very Carefully; I Shall Say This Only Once!

OUT OF THE MOUTHS OF GRANDMOTHERS

BY **PETER HOLDITCH**

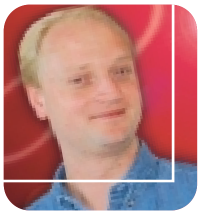
Communication in the reverse direction can be a bit of a challenge – sometimes a raised voice will suffice, sometimes repeating what you want to say several times will do – and sometimes combining the two techniques together finally gets the message across. I swear that she finds it easier to hear things selectively, based on how convenient they are for her. This is a very clever – not to say mystical – filtering technique, to be sure, but still, as she says, what's the point of getting old if you don't get artful! When all else fails, and her ability to hear has apparently reached zero, there's nothing like a good old-fashioned piece of paper and pen (unless she's lost her glasses, but that's another story altogether).
I'm sure you'll agree that my grandmother is very similar to a lot of software systems – in her day she was a woman ahead of her time, you know... Well, okay. Maybe you won't – so I'll explain what I mean…

Nearly any development that anyone ever does will involve creating a bit of new logic, and a lot of passing messages between existing systems. These existing systems are the ones that I think of as like my gran – some very elderly, and all esoteric and rather particular about how you communicate with them. Given this, it's generally a bad idea to build a new system by close coupling all the required elements in a point-to-point fashion. The resultant system would be extremely brittle and unreliable – a kind of giant, unpredictable hairball (imagine my gran and her bridge club all jointly completing a single bingo card). At the very best, all the communication logic would be surrounded by complex retry logic and be very difficult to understand.

That's why loose coupling is king – and reliable messaging often forms the backbone of loosely coupled systems within single organizations. It is for this reason that MOM systems like IBM's MQSeries and BEA WebLogic's own JMS messaging system are used so widely in real-world deployments. With their reliable messaging infrastructures, the sender of a message can know that eventually it will be delivered, whatever happens. They are the software equivalent of my writing notes to granny, and used to decouple systems can increase availability (at the expense of a whole operation *not* necessarily happening in real-time).

Whenever the subject of reliable message queuing is brought up, the subject of guaranteed once and once-only delivery is usually not far behind. It's all very well knowing for sure that a message will definitely be delivered from the point of view of the sender, but from the receiver's point of view it may be important to identify if you are receiving a message that you have actually already seen before. If my grandmother found a note reminding her to buy a pint of milk every day her fridge would soon overflow; and in the same way, if an increase in an employee's base salary was actioned multiple times because the message kept arriving at the HR system it would make for happy employees, but unhappy employers. So how do you make sure that you haven't seen a message before? Well, you need to keep a note of what messages you have received, and check each apparently new one against the list of ones you already received. So off you go to write some more code…

It's here that transactions come into the picture, and save you writing all that infrastructure logic. With an xa-compliant messaging server (for example, the one supplied out-of-the-box in WebLogic Server) the transaction manager can ensure once and once-only delivery. Most software systems, whatever the business logic, receive an instruction, process it, and update their state as a result of the processing. In the situation we're looking at now, the instruction will be received via a reliable queue. In nearly all systems, the persistent state is held in a database, most of which are xa compliant. By the rules of transactions, if you can receive the message, and do the database updates in a single JTA transaction, then either the message will be consumed *and* the updates will be persisted – if the transaction commits – or the message will be left on the queue and the persistent state will be unaltered – if the transaction aborts. It only



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

takes a minute's thought to tell you that the transaction manager, in concert with the xa-compliant queuing system and database, has delivered once and once-only delivery to your application, so to speak. A great example of the use of infrastructure technology – you can count the lines of "once and once-only" logic that you now don't have to write and maintain because the transaction infrastructure did it for you out of the box. It's the equivalent of paying someone to take away gran's reminder notes at the supermarket checkout.

There is, however, a slight design wrinkle that needs to be borne in mind when using transactions like this. Imagine that the supermarket is out of milk. Our guy never gets to take away the reminder note, and gran will end up repeatedly making the trip to the supermarket only to find the milk cooler empty – her short-term memory really shows the signs of her age. Whilst walking up and down the road may keep her fit, there are certainly better uses of her time.

In systems implemented on computers – rather than grandmothers – this is known as the "poison message problem". Imagine our HR system receiving a message that Fred Smyth should receive a 10% pay raise – which couldn't be effected since Fred was actually laid off last month. If the transaction was rolled back as a result of this discovery, then the next thing that would happen is that the same instruction will be received again, and fail for the same reason. It's even less useful to keep computers fit in this way than it is my gran… Clearly this is a bad situation. The logic in the HR system should put this instruction in an error queue and flag the problem to a human to rectify in order that it can continue processing useful work, rather than choking on the (hopefully rare) error. On the other hand, of course, if the pay raise fails because the database is still restarting after a failure, then trying again in a couple of minutes may well be a good idea. In EJB terms, this illustrates the difference between System exceptions (database down) and Application exceptions (logic dictates that this cannot happen) and explains the difference in transactional behavior of the EJB container under these two conditions.

In general, you should code your applications so that they don't roll back transactions for business reasons, rather committing some kind of error notification. However, WebLogic Server's JMS implementation actually has two configurable settings to help with this kind of problem in the event that you don't code for it correctly. To cater to the technical failure case, a session can have an associated redelivery delay – a pause before an attempt will be made to redeliver it – this will hopefully allow the database time to recover (or whatever other transient error condition that exists to be corrected). To keep the logic failures from causing a permanent problem (or to prevent the system from wasting lots of time in the event of a long-term technical outage) there is also a redelivery limit. If a the system attempts to redeliver a message more than this (configurable). number of times, then it will not try again. You can configure whether the message should simply be discarded, or rather sent to an error queue (or topic) for administrative processing. For systems that are concerned about message ordering, WebLogic can also ensure that messages are redelivered in their original order too. I have never needed to give my gran more than one note at a time, so I have never tried that – and it's somewhat beyond the scope of this article.

So that's it for this month. I'm off now for a milky cup of cocoa and an audience with gran. I guarantee that will not happen once and once only.

Keeping Memory Leaks and Stalled Threads in Check

TAKING CARE OF COMMON PROBLEMS

BY LEWIS CIRNE

AUTHOR BIO

Lewis Cirne, founder and CTO of Wily Technology, invented the company's core, patented Java Agent technology. As Wily's CTO, Lewis takes a leading role in the future technological development of the company and its products, with the goal of extending the services offered by Wily to customers deploying high-performance e-business solutions.

CONTACT...

asklew@sys-con.com

Q. What are some of the performance problems you've seen associated with the GC Heap?

There are several different categories of memory-related problems that I've seen in the field. The most common of these is the memory leak. A Java memory leak is the result of objects remaining referenced after an application has completely finished using them. This tends to happen when an object that has a long lifespan within your application holds references to other objects with short lifespans.

Memory leaks manifest as steady increases in the baseline of the JVM's GC Heap Bytes in Use, where the baseline is defined as the bytes in use after a full garbage collect. Full garbage collects look different on different JVMs. The easiest way to know whether a full garbage collect has taken place is to turn on verbose garbage collection statistics (usually, this can be done with the -Xverbose:gc flag). You can also record the GC Heap Bytes in Use and graph them over a long time span – the garbage collection pattern jumps out very clearly with such analysis.

Besides memory leaks, many server-side applications suffer from trying to keep too much session state. In some cases, reducing the session timeout can significantly reduce the memory load on the application server. But frequently the complexity of the result sets returned by queries wasn't planned for and the application had to be re-architected to ask for results in smaller pieces.

Many applications burn CPU cycles on temporary objects. Object instantiation is one of the most expensive calls in Java. If a single transaction requires more than 10,000 temporary object instances (e.g., String or Hashtable$Entry), performance takes a double hit: once on the creation and again on the increased frequency of garbage collection required to reclaim this memory.

Finding the ideal GC Heap settings is a matter of deciding the right balance point between high-frequency, quick garbage collects and low-frequency, slow garbage collects. If your application has a service-level agreement requiring all transactions to be returned in under 10 seconds, then you may not be able to delay your full garbage collects more than a few minutes – longer delays mean more memory to collect and longer stalls of the JVM. Many environments have asynchronous garbage collectors available to address just this problem, and they can be worth investigating if you have strict service-level agreements for your application.

Q. What do I do if I suspect a stalled thread in my JVM?

Your JVM may have a stalled thread for any number of reasons: deadlock, livelock, lengthy timeouts on requests to a back-end system, and so on. Each of these problems has a different signature in a live application, but the first step for diagnosis is the same in all cases: produce a thread dump.

If you're on a Unix platform, thread dumping a JVM is usually as simple as issuing the SIGQUIT to the JVM process. This can be done with the "kill -3 <PID>" command. Be sure that you issue the signal to the JVM of the process you suspect has the hung thread. Many servers have multiple JVMs running, so you should carefully grep the results of your ps command or check for the PID in the WebLogic console before you issue the SIGQUIT.

If your application server is alive in a console window, you can thread dump it by typing <CTRL>-<BREAK> on Windows, and <CTRL>-<BACKSLASH> on Unix. NT Service commands can temporarily be forced to run in a console window, allowing you to thread dump them, by checking the "Allow Service to Interact with Desktop" box in the properties for the service. Be sure to uncheck this later so that the service does not continue to pop up a window.

If you still can't obtain a thread dump when your server is hung, try adding another pool of execute threads to your WLS instance. In some cases, the JVM cannot thread dump when there are no available threads in the WLS; having a secondary queue that is unused by incoming requests will ensure availability.

Once you have a thread dump, you should look for threads that are clearly involved with incoming requests. They usually stick out clearly because they have much deeper stack traces than threads that are just waiting on sockets for incoming requests. If you are unsure whether or not a particular thread is doing something normal, try searching for it on a Web search engine or on the WebLogic performance bulletin board – if it is a normal waiting condition you will probably see lots of other thread dumps on the Internet with waiting threads that look just like yours.

With luck, you will have weeded your thread dump down to a handful of threads that could be the cause of your problem. Some late-edition Sun JVMs have built-in deadlock detection. If you suspect you have a deadlock, you could try reproducing the problem in one of these JVMs to see if it flags the problem. A "livelock" is usually a thread caught in an infinite loop.

By taking several thread dumps in a row you can see which of the several threads you are now paying attention to stays more or less in the same place each time. This takes any normal requests that just happened to get caught in the act out of the picture.

At this point, if you still haven't found your problem thread, or been able to take a thread dump at all, you should consider working with a performance tool designed for handling such threading problems.

• • •

As always, I invite you to send an e-mail to asklew@syscon.com if you have any performance-related questions about JVMs, Java applications, WebLogic Server, or connections to back-end systems.

WLDJ ADVERTISER INDEX

ADVERTISER	URL	PHONE	PAGE
Altaworks	www.altaworks.com/wldj	888-877-7111	52
BEA Systems	http://dev2dev.bea.com/useworkshop	800-817-4BEA	3
BMC Software	www.bmc.com	800-841-2031	51
Hewlett-Packard	http://devresource.hp.com/wldj.html	800-752-0900	2
Informatica	www.informatica.com	800-653-3871	23
JavaOne	www.java.sun.com/javaone/sf	650-372-7054	26-7
Linux Business & Technology	www.sys-con.com	888-303-5282	33
Precise Software	www.precise.com/wldj	800-310-4777	15
Quest Software	http://java.quest.com/jcsv/wldj	800-663-4723	19
Quest Software	http://java.quest.com/dd/wldj	800-663-4723	31
Quest Software	http://java.quest.com/qcj/wldj	800-663-4723	11
SYS-CON E-Newlsetters	www.sys-con.com	201-802-3020	47
SYS-CON Publications	www.sys-con.com/2001/sub.cfm	888-303-5282	41
SYS-CON Reprints	www.sys-con.com	201-802-3026	43
The ServerSide Symposium	www.theserverside.com/symposium?sys1	303-469-0486	9
Web Services Edge West	www.sys-con.com	201-802-3069	37
Web Services Journal	www.wsj2.com	888-303-5282	43
WebLogic Developer's Journal	www.weblogicdevelopersjournal.com	888-303-5282	43
Wily Technology	www.wilytech.com	888-GET-WILY	4, 29
Yahoo	www.enterprise.yahoo.com/bea-testdrive	866-267-7946	17

Advertiser is fully responsible for all financial liability and terms of the contract executed by their agents or agencies who are acting on behalf of the advertiser.

LOOK WHAT'S COMING NEXT MONTH

Diagnosing Application-Database Performance Bottlenecks
Properly diagnosing J2EE performance problems requires visibility into three components: WebLogic resource utilization and configuration, the application architecture, and the database query execution.

The Race to Create Web Services Business Process Standards
The number of Web service business process (BP) specifications trying to make their way to standards status makes it difficult to tell who is doing what. We'll make sense out of the morass by classifying Web service BP specifications into four categories.

Enhancing Application Manageability
The absence of manageability can cause serious problems. Application software can become an unmanageable entity once it is deployed because the operations staff does not understand it. Lack of attention to manageability raises the cost of supporting and maintaining the software product significantly. This article offers choices to developers on solving these problems.

JMS Messaging and WebLogic 7.0
Creating a helper class for sending JMS messages generically with WebLogic is beneficial for any asynchronous messaging effort, and provides a basis for future JMS development. The advantages of creating a generic access path for JMS are highlighted, and a method for doing so is provided.

J2EE and Struts
What can be accomplished by using Struts? As a start, you can standardize navigation, validation and display, and make your applications easy to use, maintain, and extend.

Plus, review of WebLogic Workshop 8.1

BEA WebLogic DEVELOPER'S JOURNAL

News & Developments

Dirig Software Adds Greater Flexibility, Broader Support

(Nashua, NH) – Dirig Software, a developer of enterprise Web-application performance management solutions, has announced the immediate availability of Dirig 4.0, their upgraded product platform. The 4.0 platform introduces Dirig Performance Center for graphical monitoring of applications in a single view, helping customers translate performance metrics into a specific business context. It also updates their existing products – Dirig PathFinder, the Fenway management solution, and the Dirig Agent – with greater system flexibility through wider RDBMS support and integration with the leading server platforms, applications, and standards-based protocols. www.dirig.com

Blue Titan / BEA Announce Joint Solution for Data Integration

(San Francisco) – Blue Titan Software, Inc., has announced a joint solution with BEA Systems that brings service-level control to real-time data applications powered by BEA Liquid Data for WebLogic. The joint solution will enable enterprises to enforce service-level agreements (SLAs); control access to enterprise systems; and monitor performance and usage across disparate systems.

In addition, Blue Titan and BEA are providing a roadmap for introducing a service-oriented approach to real-time data integration.
www.bea.com,
www.bluetitan.com

Panasonic Adopts WebLogic Portal to Personalize Experiences

(San Jose, CA) – BEA Systems, Inc., has announced that Matsushita Electric Industrial Co. Ltd. (Panasonic), the electronics leader, has selected BEA WebLogic Portal software for a new business support system, "Analog Semiconductor Application Portal," within Panasonic's LSI semiconductor company.

Aggregating information on useful public sites, data search engines, and group schedule managing functions, Panasonic's Analog Semiconductor Application Portal has become the central hub for developer knowledge. www.panasonic.co.jp/global/top.html

BEA and Siebel Systems Expand Alliance

(San Jose, CA and San Mateo, CA) – BEA Systems, Inc., the world's leading application infrastructure software company, and Siebel Systems, Inc., a leading provider of multichannel e-business applications software, have expanded their strategic alliance agreement to deliver integration solutions through joint development, sales, and marketing initiatives.

Siebel Systems has named BEA a Strategic Software and Universal Application Network (UAN) Partner in the Siebel Alliance Program and designated BEA WebLogic Integration as one of its strategic integration solutions for UAN, a standards-based approach to multiapplication integration based on best-practice business processes. BEA will offer full out-of-the-box support for UAN in upcoming versions of WebLogic Integration and WebLogic Platform. www.bea.com, www.siebel.com

BEA Supports Customers of Microsoft Windows Server 2003

(San Jose, CA) BEA Systems, Inc. has renewed its commitment to helping customers leverage Microsoft Windows investments and resources with the announcement of support for Windows Server 2003, Enterprise Edition.

BEA delivers a Windows-friendly Java platform. Windows is the leading platform for WebLogic development and among the most popular platforms for WebLogic deployments. BEA WebLogic solutions are interoperable with most Windows Server 2003 applications and services. www.bea.com

BEA and Mercury Interactive Partner on Integrated Solution

(San Jose, CA and Sunnyvale, CA) – BEA Systems, Inc., and Mercury Interactive Corporation, the leader in business technology optimization, have announced a strategic partnership to help customers measure, manage and maximize the quality and performance of BEA WebLogic Platform.

As part of the agreement, Mercury Interactive has developed a business technology optimization (BTO) environment designed specifically for BEA WebLogic Platform 8.1. Based on Mercury Interactive's Optane software suite, this environment includes WebLogic-specific monitors that provide enterprise testing, deployment assurance, and application performance management capabilities to help customers build, integrate, and deliver service-oriented Java applications. In addition, BEA and Mercury Interactive will collaborate on joint selling and marketing activities around the integrated environment.
www.mercuryinteractive.com

Panacya Expands BusinessAware e-Business Management Suite

(Las Vegas) – Panacya Inc. has released the latest version of its BusinessAware e-business management suite. Providing information to business managers, operations staff, and Web application developers within the context of managing business service–level objectives, BusinessAware 2.2 is particularly suited to enterprises deploying BEA WebLogic application servers as well as other common application infrastructure components found in J2EE environments. It provides real-time availability and performance analysis across an enterprise's entire Web application infrastructure.
www.panacya.com

SEAGULL Adds Support for ICL Mainframe Applications

(London and Atlanta) – SEAGULL, a developer of legacy evolution software solutions, has announced that its LegaSuite platform now supports ICL, a transactional legacy mainframe environment for enterprise applications.

SEAGULL's LegaSuite is an integrated platform of software solutions for rapidly evolving legacy systems to strategic Web architectures. Other capabilities include transforming ICL applications into Web services, and accelerating the integration of ICL applications with important infrastructure products such as BEA WebLogic. www.seagull-sw.com

BMC Software

www.bmc.com

Altaworks

www.altaworks.com/wldj