

NOV/DEC 2005 VOL.4 ISSUE.6

WLDJ™

WWW.WLDJ.COM

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

Internationalization of a J2EE Web APP

PLUS...

**Assured Delivery
of Audit Data ...6**

**Business Process
Execution Language
for Java ...10**

**Configuring Eclipse
for Remote Debugging
a WebLogic Application ...16**


**Design for Production Meets the
Application Delivery Process ...48**

...20

Presorted
Standard
US Postage
PAID
St. Croix Press

*Avoiding Middle-Aged Spread for
Your Infrastructure* page 38

*Adding Self-Detection and
Auto-Optimization to the WebLogic 8.1
Platform* page 40



A problem isn't
a problem if it
never happens.

Proactive management.
Real-time control.
Total visibility.
Intersperse Manager.

With BEA WebLogic Platform™, global organizations are building powerful business advantages using SOA. And the ultimate value comes when these vital applications are in full production.

But keeping these complex applications up and running can be a challenge – putting you, and your business, at risk.

Intersperse Manager™ removes those risks, optimizing your investment in BEA and delivering on the promise of SOA. Using non-invasive management standards, such as JMX, Intersperse Manager enables real time visibility, context, and control at all levels.

It gives you true, proactive production application management that corrects issues – before they become problems.

Intersperse Manager helps you build business advantage – and that's an asset worth protecting.

Call Intersperse today for
a free demonstration

866.499.2202

**intersperse™**

Management solutions
for the evolving enterprise
www.intersperse.com



8.1 VALIDATED

BEA WEBLOGIC PLATFORM

The Developer Paradox:

No time to test your code? But **long hours** reworking it & resolving errors?



Check out **Parasoft Jtest® 7.0**
Automates Java testing and code analysis.
Lets you get your time back and deliver quality code with less effort.

■ **Automated:**

Automatically analyzes your code, applying over 500+ industry standard Java coding best practices that identify code constructs affecting performance, reliability and security.

Automatically generates and executes JUnit test cases, exposing unexpected exceptions, boundary condition errors and memory leaks and evaluating your code's behavior.

Groundbreaking test case "sniffer" automatically generates functional unit test cases by monitoring a running application and creating a full suite of test cases that serve as a "functional snapshot" against which new code changes can be tested.

■ **Extendable:**

Industry standard JUnit test case output make test cases portable, extendable and reusable.

Graphical test case and object editors allow test cases to be easily extended to increase coverage or create custom test cases for verification of specific functionality.

■ **Integrated:**

Integrates seamlessly into development IDE's to support "test as you go" development, and ties into source control and build processes for full team development support.

To learn more about Parasoft Jtest or try it out, go to www.parasoft.com/JDJmagazine



Automated Software Error Prevention™

Rebuilding the Tower of Babel

From the Editors

Once upon a time people spoke the same language – but that was long ago. Nowadays people speak hundreds of languages with unique characters, inflection indicators, and other punctuation marks that make each language different from all others.

Prior to the advent of the computer and the breakdown of geographic travel barriers, there was little need to transform information from one language to another. Now we have a truly global information need – one that must be localized for the particular consumer, and that must take into account the local preferences. For example, is the information read from left to right, right to left, or across or down? Does the information fit the formatting after translation, or does the formatting need to be adjusted? Are proper names inadvertently offensive in a different language? These are just some of the issues that occur when we try to rebuild the Tower of Babel and make information a global commodity once again.

Our cover story for this issue is “Internationalization of a J2EE Web Application.” Authors Murali Kashaboina and Bin Liu present a practical solution to making Web applications multilingual. Just as in the aforementioned allegory, languages have different characters and encoding schemes, so application real estate has to be managed. Internationalization is an attempt to move the “language-specific” content to places where it can be conveniently managed. This thorough article provides insight into the design for internationalization, which the authors state, “should be an up-front task and not an afterthought.”

Michael Poulin explains how to achieve reliable messaging via Web services and JMS in his article “Assured Delivery of Audit Data.” He describes two techniques for assuring the delivery of important data in distributed systems, and emphasizes how significant assured delivery is “in light of global operation risk

regulations and related application risk management.”

Michael Havey’s article “Business Process Execution Language for Java” also delves into the subject of languages by examining a popular language for the definition and execution of business processes and its Java extension. The article is excerpted from Michael’s book *Essential Business Process Modeling*, and those who wish to know more about the subject would do well to read it.

Deepak Vohra and Ajay Vohra present a tutorial on “Configuring Eclipse for Remote Debugging a WebLogic Application.” The article takes readers through the preliminary setup, through developing a WebLogic Application, configuring a remote debugging configuration in Eclipse, and finally, remote debugging an application.

Raman Sud offers readers lessons from the world of manufacturing in his article “Design for Production Meets the Application Delivery Process.” He ponders what sort of criteria might be applicable to assessing the comparative health of an application delivery system with regard to “design for production” standards.



The article considers centralizing all configuration artifacts, gauging changes as an application progresses across the life cycle, the definition and enforcement of standards with regard to changes in the IT infrastructure stack, the elimination of wasted effort, and relative consistency.

In “Avoiding Middle-Aged Spread for Your Infrastructure,” Peter Holditch shares his observations about the onset of mid life for both people – and applications. He compares how things functioned in the early

days of each, and what they have matured into since.

Deepak Batra shows readers techniques for “Adding Self-Detection and Auto-Optimization to the WebLogic 8.1 Platform” and offers solutions from his own professional experience.

We hope you will enjoy this issue’s focus on global communication, and have a wonderful holiday season – wherever in the world you are! 🍎

Contact:
info@sys-con.com

EDITORIAL ADVISORY BOARD

Lewis Cirne, Wayne Lesley Lund,
 Chris Peltz, Sean Rhody

EDITORIAL

Founding Editor
 Peter Zadrozny

Executive Editor
 Seta Papazian seta@sys-con.com

Editor
 Nancy Valentine nancy@sys-con.com

Research Editor
 Bahadir Karuv, PhD bahadir@sys-con.com

PRODUCTION

Production Consultant
 Jim Morgan jim@sys-con.com

Lead Designer
 Abraham Addo abraham@sys-con.com

Art Director
 Alex Batero alex@sys-con.com

Associate Art Directors
 Louis F. Cuffari louis@sys-con.com
 Richard Silverberg richards@sys-con.com
 Tami Beatty tami@sys-con.com
 Andrea Boden andrea@sys-con.com

Video Production
 Frank Moricco frank@sys-con.com

CONTRIBUTORS TO THIS ISSUE

Deepak Batra, Michael Havey, Peter Holditch,
 Murali Kashaboina, Bin Liu, Michael Poulin,
 Raman Sud, Ajay Vohra, Deepak Vohra,

EDITORIAL OFFICES

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

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

©COPYRIGHT

©Copyright 2005 by SYS-CON Publications, Inc. All rights reserved. No part of this publication may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopy or any information storage and retrieval system, without written permission. For promotional reprints, contact reprint coordinator, SYS-CON Publications, Inc., reserves the right to revise, republish and authorize the readers to use the articles submitted for publication. All brand and product names used on these pages are trade names, service marks or trademarks of their respective companies. WLDJ and WLDJ.com are registered trademarks of SYS-CON Media. WebLogic is a trademark of BEA Systems. SYS-CON and WLDJ are independent of BEA Systems, Inc.

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

REPRINTS

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



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

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

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

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

Get Wily.™

1 888 GET WILY | wilytech.com

wily
technology

Assured Delivery of Audit Data



By Michael Poulin

RELIABLE MESSAGING VIA WEB SERVICES AND JMS

This article describes two techniques that may be used for assured delivery of important data, specifically, audit data, in distributed systems.

We will review design that leads from assured to guaranteed delivery. This task gets more and more important in light of modern global operation risk regulations and related application risk management.

Business Task and Functional Requirements

Relatively recent operation risk management regulations like Sarbanes-Oxley (SOX), and in some cases Basel II, require collection of “material evidences” of user activities that can affect financial reporting of the company. This includes user activities in the software applications, especially in the financial industry.

In many cases an activity is interpreted not only as a fact of application access, but also as an access to a particular application function and even data. The user activities in the application are supposed to be stored in persistence storage for following audit (for this article we will use relational database for simplicity). Such databases are usually centralized and serve multiple applications; therefore, we are dealing with distributed systems. The aforementioned regulations assume that audit data may not be lost on its way to the database. In automated systems, this means assured delivery of data.

Assured delivery of data is not a new thing in the application landscape. For many years MOM (Message Oriented Middleware) and recently, ESB (Enterprise Service Bus) technologies provided such functionality. The “cons” here are the high product costs and expensive maintenance. Plus, they do not guarantee that sent data is stored in the targeted persistent storage – they only assure that data is reliably transmitted from the sender to

the receiver components or rolled back. This is the basic difference between assured and guaranteed delivery. In the article we will discuss an assured delivery and design a guaranteed delivery feature utilizing widely available J2EE technologies that may be suitable for small companies or for departments of large corporations.

When talking about delivery data with assurance in a distributed system, the first thing that comes to my mind is reliable messaging (RM). If an item of audit data is interpreted as a message, we can concentrate on the delivery mechanisms – in particular, on Web services and Java Messaging Service (JMS). It is interesting to notice that if a task of delivery is slightly extended and includes the reuse of audit data for integration with other systems, e.g., security systems, the Web services-based design has to be reconstructed to become scalable, while the JMS-based design requires just an extension for reliability. Details of these designs will be discussed in the following sections.

Web Services-Based Solution

The reliable messaging implemented as Web services is based on several standards such as SOAP, Web Services Reliable Messaging (WS-RM), WS-Acknowledgement, WS-MessageData, WS-Callback, SOAP-Conversation, and others. WebLogic platform version 8.1 provides SOAP Reliable Messaging solution while version 9.0 offers WS-RM. In both cases, the concept of RM may be demonstrated as shown in Figure 1.

Author Bio:

Michael Poulin is working as a technical architect for a leading Wall Street firm. He is a Sun Certified Architect for Java Technology. For the past several years Michael has specialized in distributed computing, application security, and SOA.

Contact:

mpoulin@usa.com

PRESIDENT & CEO

Fuat Kircaali fuat@sys-con.com

VP, BUSINESS DEVELOPMENT

Grisha Davida grisha@sys-con.com

GROUP PUBLISHER

Jeremy Geelan jeremy@sys-con.com

ADVERTISING

Senior VP, Sales & Marketing
Carmen Gonzalez carmen@sys-con.com

VP, Sales & Marketing
Miles Silverman miles@sys-con.com

Advertising Director
Robyn Forma robyn@sys-con.com

Advertising Manager
Megan Mussa megan@sys-con.com

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

Associate Sales Manager
Kerry Mealia kerry@sys-con.com

SYS-CON EVENTS

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

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

CUSTOMER RELATIONS

Circulation Service Coordinators
Edna Earle Russell edna@sys-con.com

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

SYS-CON.COM

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

Web Designers
Stephen Kilmurray stephen@sys-con.com

Online Editor
Roger Strukhoff roger@sys-con.com

ACCOUNTING

Financial Analyst
Joan LaRose joan@sys-con.com

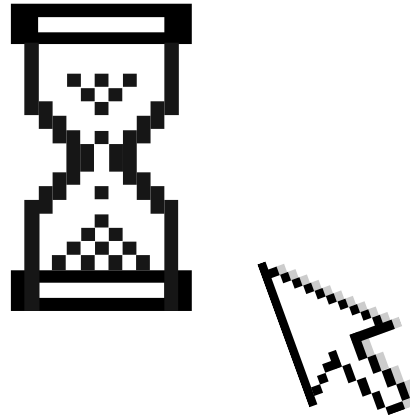
Accounts Payable
Betty White betty@sys-con.com

Credits & Accounts Receivable
Gail Naples gailn@sys-con.com

SUBSCRIPTIONS

subscribe@sys-con.com
Call 1-888-303-5282

For subscriptions and requests for bulk orders, please send your letters to Subscription Department
Cover Price: \$8.99/issue
Domestic: \$149/yr (12 issues)
Canada/Mexico: \$169/yr. All other countries \$179/yr
(U.S. Banks or Money Orders)
Back issues: \$12 U.S. (plus \$8/H)



World Wide Wait.

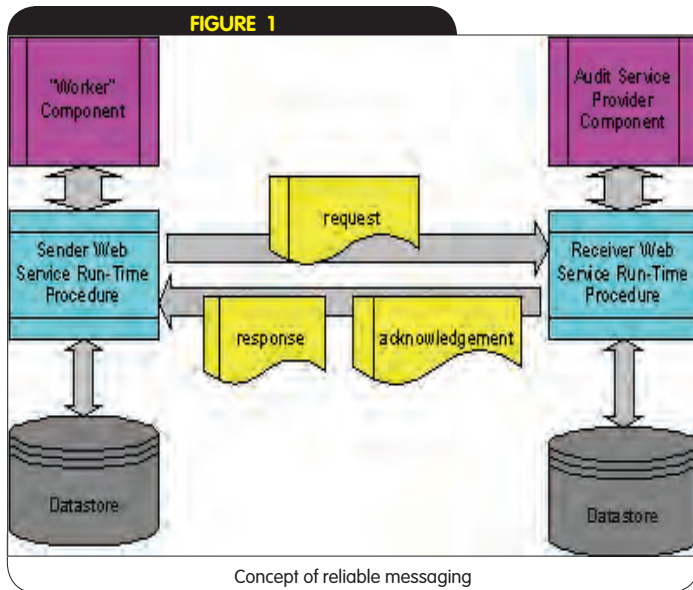
Don't Leave Your Customers Hanging — Ensure a Positive End User Experience with Quest Software.

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

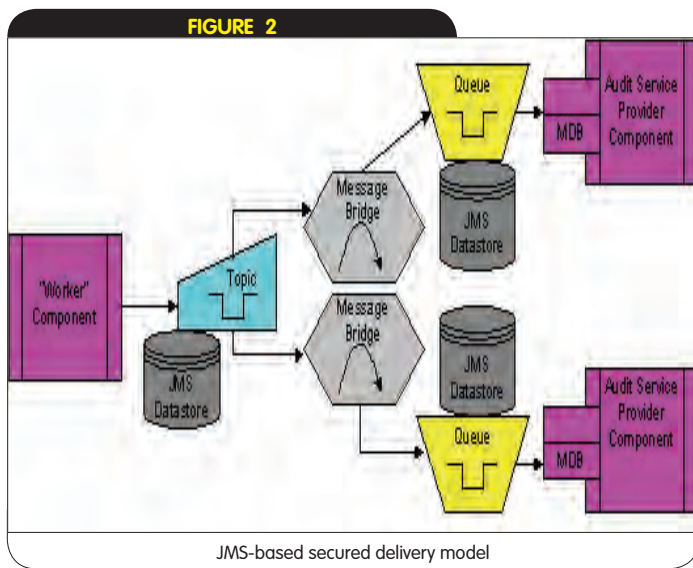
Quest Software's solutions for J2EE application performance management shed light on the true end user experience and help you isolate and fix problems across all tiers. Whether a problem lies in your server cluster, database, framework or Java code, nothing gets you to resolution and optimal performance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

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

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



Concept of reliable messaging



JMS-based secured delivery model

An audit message is created in the Worker Component or Business Application and sent to the Sender Run-Time Procedure. Before the message is sent further, it is persisted locally. This protects the message from being lost if the receiver side is unavailable at the moment. Then the message is sent to the Receiver Run-Time Procedure where it is persisted first of all.

Since message transition is performed in the transaction, the latter can be rolled back in case of any problems on the network or receiver side. If the transaction is rolled back, the sender is notified that the message was not delivered. Depending on configuration, the message may be re-sent by Sender Run-Time Procedure or by the sender.

The Receiver Run-Time Procedure invokes a business method in the Audit Service Provider before the acknowledgement of delivery is sent to the sender. If the receiver – the Audit Service Provider – operates in the Receiver Transaction Context, it has an ability to perform its own operations in the same transaction. For example,

the receiver can store the message in the database. If storing fails and rolls back, the Receiver Transaction Context rolls back and in turn, the process does not remove the message from the persistent store of the Sender Run-Time Procedure. Thus, the audit data is not lost. The only problem with this mechanism is that the Receiver Transaction Context does not automatically roll back if the receiver throws an application-specific exception, i.e., the Audit Service Provider has to take care of such exceptions and explicitly roll back the Receiver Transaction Context if needed. As we can see now, if data is persisted using the same transaction as the Receiver Transaction Context, we get a reliable solution for transmitting audit data into persistent storage.

The RM in the WebLogic 8.1 implementation has one major limit – it works on the WebLogic platform only. WebLogic 9.0 overcomes that limit via support of WS-RM, which works across all platforms that support the same standard. However, if a message has to be sent to multiple storages or transmitted data should be used for integration with other systems (for example, integration between authorization systems built into ALES, Documentum, and Business Objects products), the described Web services-based solution is limited in “vertical” business scalability. In particular, every time a new audit data consumer (or destination endpoint) has to be added to the system, a new sender (or source endpoint) has to be implemented and deployed.

To improve “vertical” scalability, we probably need to change design and set the Worker Component as a Web service while setting the Audit Provider and other integrated data consumers as Service clients. An alternative design might include an intermediary service that is situated between client (again – Worker Component) and integrated services. The intermediary service distributes audit data to all interested services. In both cases, original Web services-based design requires significant modification.

JMS-Based Solution

While JMS is designed for assured delivery from the beginning, we have to use it in a special way to achieve guaranteed delivery. Moreover, since we are discussing practical solutions, we have to address security in the design, despite the fact that it does not contribute to the guaranteed delivery process itself. Information security has not been discussed with regard to Web services because it is a well-known issue and has accumulated a lot of attention already. At the same time, messaging is traditionally considered as internal infrastructure and therefore secured, while actually, it is not (the majority of recent research points out that 75-80 percent of security violations happened inside the company). Therefore, we will examine JMS-based design while keeping in mind “vertical” scalability, security, and reliability.

Design for Security and Scalability

Let’s assume that we deal with two Audit Service Providers. Each one collects only audit data of a certain type. Instead of an Audit Service Provider, it may be another system that integrates with Worker Component via data exchange. If we use just two JMS Queues for message receivers, we will need to modify sender code when we add more receivers, – this solution is not scalable. Therefore, we need to broadcast the message to all interested parties/receivers via, for example, JMS Topic.

Since audit data is sensitive, we cannot just put a message into

a JMS Topic and rely on message filtering on the receiver side to select only appropriate messages; instead, we have to direct the message to the approved receivers only. We can observe several security models for JMS. As we know, JMS Connection Factory access may be protected by user name and password (UN/PW). We believe that this protection is not enough especially if the Topic is used by multiple receivers for integration purposes. It is common practice that, for example, an operation team discloses user name and password to those projects that need integration urgently without notifying the information owner (sender) and without checking security compliance. In WebLogic Trusted Domain configuration (trust between WebLogic Server domains), the password is not required at all.

Sensitive data also may be encrypted; however, it requires dealing with additional encryption service and/or encryption key management infrastructure. Both of them may be not available or may be too expensive.

We propose an inexpensive combination of UN/PW with operational security design. Figure 2 displays one of the possible solutions where messages that are sent to the JMS Topic are directed to particular JMS Queues for approved receivers, e.g., Audit Service Providers.

The Message Bridges shown in the diagram have been known since the WebLogic 7.0 release and play the role of an intermediary. They subscribe to the JMS Topic and transmit messages in the transaction to the JMS Queues. Both Topic and Queues are configured with assured delivery and the Message Bridges have a durable subscription to the Topic. Each Audit Service Provider uses Message-Driven Beans (MDB) for message retrieval from the JMS Queue. The MDB starts a transaction and manages the message transition into the database in the scope of the same transaction.

If a Message Bridge is down, the message stays with JMS Topic. If a JMS Queue is down, the Message Bridge rolls back its transaction and the message still stays with the JMS Topic. If Audit Service Provider or database experiences problems, the MDB transaction rolls back

and the message stays with the Queue. When problems are fixed, it is guaranteed that the message is processed and stored in the database.

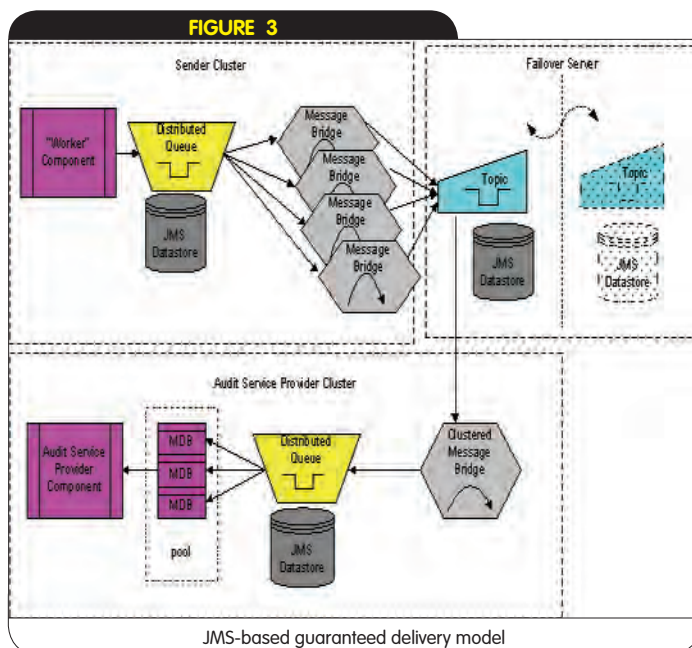
The trick here is not in the technical solution, but in the operational data processing. The matter is that the information owner controls Message Bridges and approves receivers. Upon approval, the receiver is granted the Message Bridge and JMS Queue, and the operation team configures them (in addition to UN/PW). Thus, the solution gets relatively secured and is still flexible and scalable.

Design for Reliability

There are several steps to be performed to achieve reliability and guaranteed delivery via the JMS model described before. The overall system is shown in the Figure 3.

- Step 1: All JMS destinations used in the system have to be configured with assured delivery. All JMS listeners – Bridges and MDB – have to use durable subscriptions.
- Step 2: It is clear that the JMS Topic is the heart of the system. That is why we have to minimize the risk of its failure. We place the JMS Topic on a physically separated server with an automated failover feature. If a working instance of the Topic gets down, the failover instance immediately starts to work. If the sender or receiver's application server goes down, the Topic is still capable of operating.
- Step 3: Since JMS Topic becomes a remote destination, the Worker Component endures a risk of losing audit data if the JMS Topic is not reachable. To protect audit data at this point, we add a distributed (clustered) JMS Queue, which is situated on the same cluster as the Worker Component.
- Step 4: We add Message Bridges to connect each of the instances of the distributed Queue with the JMS Topic. All Message Bridges are situated in the same cluster as the sender's distributed Queue.
- Step 5: For every receiver, create a distributed Queue that serves the receiver (e.g., Audit Service Provider). A receiver's Queue may be isolated from or collocated with the sender component.
- Step 6: Connect the JMS Topic with the receiver's distributed Queue by a Message Bridge. The latter transmits messages in the transaction to the receiver's distributed Queue. This Message Bridge is collocated with the receiver's distributed Queue.
- Step 7: Receivers, such as Audit Service Providers, use MDB to subscribe to the receiver's distributed Queue. MDB operates in the same way as described above.

Now, let's look at how it works together. The sender forms an audit message and sends it to the sender's distributed Queue where it is persisted. The Message Bridge, which connects a particular JMS Queue instance with the JMS Topic, starts a transaction, retrieves the message from the Queue, and sends it to the Topic. If Message Bridge is down or JMS Topic is unavailable, the message stays with the sender's distributed Queue until next attempt. If the message is delivered to the JMS Topic successfully, the acknowledgement is sent to the sender's distributed Queue and the message leaves from the sender's distributed Queue persistent storage. In Figure 3, four images of the Message Bridge represent a case of a cluster with four nodes where the distributed Queue is situated.



– continued on page 50

Business Process Execution Language for Java

BPEL'S JAVA EXTENSION



By Michael Havey

The Business Process Execution Language for Web Services (BPEL4WS, usually shortened to BPEL, which rhymes with “people”) is, as its name suggests, a language for the definition and execution of business processes. Though it is not the only standard process language, BPEL is the most popular, and it’s beginning to saturate the process space.

IBM, Microsoft, and BEA wrote the BPEL specification and subsequently handed it over to the WSBPEL technical committee of the OASIS organization (of which they are members) for standardization. The conceptual roots of BPEL coincide exactly with earlier BPM initiatives of each of the three companies: IBM’s WSFL, Microsoft’s XLANG, and BEA’s Process Definition for Java (PD4J). WSFL is based on Petri nets and XLANG uses concepts of the pi calculus; BPEL, consequently, is a mixture of these two theories. PD4J is the basis for the Java extension to BPEL, known as BPELJ.

BPELJ

“Pure” BPEL emphasizes “programming in the large,” or the activity of defining the big steps of a process in a clean XML form; however, a process that is meant to actually run and be useful invariably has countless little steps, which are better implemented as lines of code than as activities in a process. Functions such as processing input, building output, interfacing with in-house systems, making decisions, and calculating dates either drive or are driven by the process, but are often too complex to encode as part of the process flow.

“Programming in the small” is essential to the development of real-world processes, but is hard

to accomplish with pure BPEL. The best pure BPEL can offer is Web services: if a piece of logic is hard, develop it as a Web service and call it from the process with an invoke activity. This approach works, but it is grossly inappropriate. First, the reason for BPEL’s emphasis on Web services is the goal of communicating processes. Partners call each other as services, but must a partner call pieces of itself as services? Second, the performance overhead of calling a service is obviously a showstopper; the process needs fast local access to the small logic components, as if they were an extension of the BPEL machine.

With these factors in mind, IBM and BEA have written a white paper that presents a standard Java extension of BPEL called BPEL for Java (BPELJ) (see

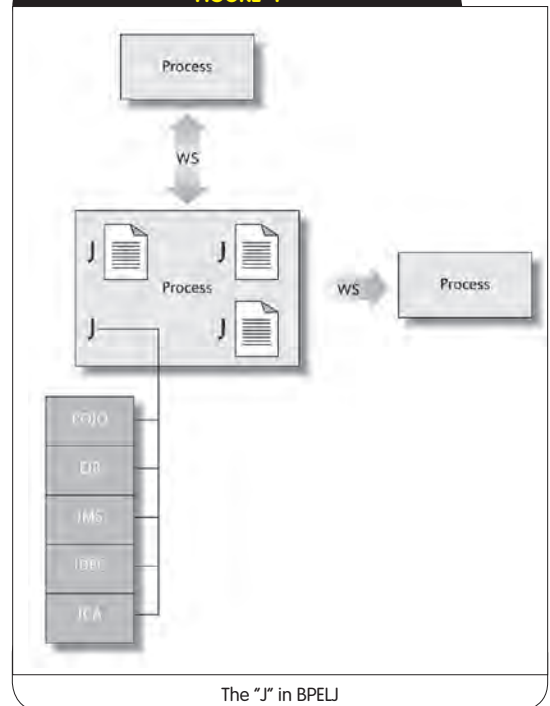
Author Bio:

Michael Havey is an IBM consultant with 10 years of industry experience, mostly with application integration. Michael’s book *Essential Business Process Modeling* was published by O’Reilly in August 2005.

Contact:

haveym@ca.ibm.com

FIGURE 1



The “J” in BPELJ



Pure *Java*
Pure *Excel*
Power for users
Control for IT
Pure *Paradise*

Formula One e.Spreadsheet Engine

*API-driven, embedded, 100% pure-Java,
scalable spreadsheet toolset*

Spreadsheets play an essential role in the business world. And now, they can also perform a vital function in your Java applications. How? With the Formula One e.Spreadsheet Engine, an API-driven, 100% pure Java toolset for embedding Excel spreadsheet functionality in your applications.

Excel-enable your Java applications

Use a state-of-the-art graphical development environment to build fully-formatted Excel report templates that include formulas, functions, colors, merged cells, even charts. It's fast, easy and effective.

Embed live, Excel-compatible data grids

Include interactive Excel forms and reports in your Java applications. Let users manipulate them using their spreadsheet skills and then commit the changes to databases or applications – or save them as an XLS file on their desktops.

Automate complex calculations and rules

Embed Excel spreadsheets that automate complex calculations and business rules on J2EE servers. Stop translating spreadsheet logic into Java code today, and start leveraging the development skills of your organization's spreadsheet experts.

Read and write server-based Excel spreadsheets

Give your users server-based spreadsheets populated with up-to-the-minute information directly from data-bases, XML files and enterprise applications. Get control of runaway data warehouses and spreadmarts now.

Make spreadsheets a part of your strategies

Visit us today at www.reportingengines.com to request a **free trial** of the e.Spreadsheet Engine. We'll show you how to make Excel spreadsheets a vital and productive part of your enterprise computing strategies.



ACTUATE
ReportingEngines™

www.reportingengines.com
sales@reportingengines.com
888-884-8665 + 1-913-851-2200

**FREE TRIALS,
DEMOS AND
SAMPLE CODE**

the first entry in the References section). A BPELJ process, depicted in Figure 1, has chunks of embedded Java code, as well as invocations of separate plain old Java objects (POJOs), Enterprise Java Beans (EJBs), or other Java components. Though it still interfaces with its partner process through Web services, the process internally leverages Java to perform much of the hard work.

BPELJ is an obvious technology choice for companies that intend to deploy their processes on J2EE platforms such as BEA Weblogic and IBM WebSphere. The platform is already Java-enabled, so it is best to use Java capabilities in the construction of the process. Luckily, BEA and IBM are building to BPELJ.

Note: Current BPELJ vendor implementations are difficult to find. Two BPEL toolkits – the Oracle BPEL Process Manager 2.2 and IBM's WebSphere Application Developer Integration Edition 5.1.1 – have Java extensions, but they are proprietary. BEA is planning to develop a reference implementation of BPELJ and possibly release it as open source. Expect major Java application server vendors like BEA, IBM, and Oracle to be the earliest adopters and to evolve the language.

A Glimpse of BPELJ

Listing 1 demonstrates some of the core features of BPELJ. The red-colored parts are BPELJ-specific. The process receives a claim by a Web service request from a client application, then processes it using an EJB, and finally, if the claim is successful, publishes the request to a JMS topic for consumption by interested subscribers. The process extends core BPEL by defining partner links for Java components (lines 16–19), declaring Java variables (lines 25–26), using the invoke activity to call Java components (lines 37–40 and 51–53), using Java expressions to make decisions that affect flow (line 45), and embedding a snippet of Java code (lines 46–50). BPELJ features not included in this example include Java correlation, Java exception handling, and XML-Java binding.

Warning: Some of these

changes are not supported by BPEL 1.1! The Java snippet in lines 46–50, for example, is illegal because BPEL does not support the addition of new activity types. Similarly, the input and output elements (lines 38–39) are not permitted in invoke and receive activities, and the condition (line 45) in the switch activity should be an attribute rather than a child element of case. In their joint whitepaper, BEA and IBM admit these incompatibilities and even suggest alternative approaches that are supported. For example, the snippet could be overloaded in the empty activity. The alternatives are onerous and unintuitive, which explains why the authors chose to cheat. *BPELJ won't be ready for prime time until these issues are resolved.*

BPELJ Source Code

The source code of a pure BPEL process is a set of XML files containing WSDL and BPEL process definitions. BPELJ source code can be either XML files with embedded Java code or Java source files annotated with XML. The former approach is the one adopted in the examples above; likewise, most of the samples in the BPELJ specification are XML with embedded Java.

The latter approach, documented in the BPELJ specification as a viable alternative, makes sense only if the number of lines of Java code is close to the number of lines of XML. Listing 2 shows a Java source file (*MyProcessImpl.java*) containing a comment in lines 1–15 that defines the BPEL process XML. The source file

implements the class *MyProcessImpl*; the source code begins on line 16. Lines 17–20 implement a method that is called in the process on line 10.

A well-designed BPELJ process should be mostly pure BPEL with a smattering of Java. (Analogously, a well-designed Java Server Page is mostly markup with minimal embedded Java.) Significant Java processing can be factored out to special Java partner link types, whose source code resides in traditional Java source files, separate from the process. Consequently, XML-driven BPELJ is preferable to Java-driven BPELJ.

Note: BPELJ is conceptually similar to Process Definition for Java (PD4J) (JSR 207, www.jcp.org/en/jsr/detail?id=207), a specification proposed by BEA to the Java Community Process (JSR 207) for mixing XML and Java for process definition. PD4J is the design model for BEA's WebLogic Integration 8.1. BEA, along with IBM, authored BPELJ (also submitted to JSR 207), and is building to it for its Version 9 release of WebLogic Integration. PD4J favors the Java-with-annotated-XML approach, so perhaps WebLogic Integration 9 will adopt annotated Java as its development model.

Other Language Implementations

BPELJ is the first language extension of BPEL, extending BPEL's capabilities on Java platforms. The same approach is suitable for other programming languages, notably those that figure prominently into

Microsoft's .NET platform, such as C#. Expect to see such implementations soon, as BPEL increases in popularity.

Summary

The main points of this article are as follows:

- BPEL was originally written by IBM, Microsoft, and BEA, but has been handed over to OASIS for standardization. BPEL is based on IBM's WSFL and Microsoft's XLANG.
- BPELJ introduces Java extensions to "pure" BPEL, such as the ability to define Java process variable,





TANGOSOL™



A Cure for Downtime.

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

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

Try before you buy.

Download a free evaluation license at

www.tangosol.com

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

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

evaluate dates and conditions with Java code, and embed code snippets. Other powerful features include Java partner links (enabling invoke calls to local Java classes in addition to pure BPEL's partner Web services) and correlation based on Java classes.

- BPELJ source code can be XML with embedded Java or Java with annotated

XML. The former approach is arguably better from a design perspective. The latter approach is influenced by BEA's PD4J model, used in WebLogic Integration 8.

• • •

This article has been excerpted from the

book *Essential Business Process Modeling* by Michael Havey.

References

- Blow, M., Goland, Y., Kloppman, M., Leymann, F., Phau, G., Roller, D., Rowley, M. "BPELJ: BPEL for Java." March 2004. www-106.ibm.com/developerworks/webservices/library/wl-bpelj

Listing 1: BPELJ insurance automated claim

```

1 <!-- Process attributes:
2   - expressionLanguage is Java by default. Can be overridden to,
3   say, XPath at the element level
4   - Java code embedded in process goes in to the Java package "com.
   mike.claim"
5   - The BPELJ namespace is referenced below.
6 <process name="InsuranceClaim"
7   suppressJoinFailure="yes"
8   expressionLanguage="http://jcp.org/java"
9   bpelj:package="com.mike.claim"
10  targetNamespace="http://mike.com/claim"
11  xmlns:tns="mike.com/claim"
12  xmlns="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
13  xmlns:bpws="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
14  xmlns:bpelj="http://schemas.xmlsoap.org/ws/2003/03/business-process/
   java">
15
16 <!-- Three partner links: one a web service client interface,
   the others
17   Java internal stuff -->
18 <partnerLinks>
19   <partnerLink name="client" partnerLinkType="tns:Claim"
20     myRole="ClaimProvider"/>
21   <partnerLink name="claimProcessor"
22     partnerLinkType="bpelj:com.mike.claim.ClaimProcessorEJB">
23 <partnerLink name="jmsPublisher"
24   partnerLinkType="bpelj:javax.jms.TopicPublisher">
25 </partnerLinks>
26
27 <!-- Two variables, one an XML, the other Java -->
28 <variables>
29   <variable name="input" messageType="tns:ClaimsMessage"/>
30   <variable name="jmsMessage" messageType="bpelj:javax.jms.
   TextMessage"/>
31   <variable name="claimOK" messageType="bpelj:java.lang.Boolean"/>
32 </variables>
33
34 <sequence name="main">
3   <!-- process starts by receiving a claim through the client web
   service -->
36 <receive name="receiveClaim" partnerLink="client" portType="tns:
   Claim"
37   operation="initiate" createInstance="yes">
38   <output part="input" variable="input"/>
39 </receive>
40
41 <!-- now invoke the Java claims processor as a partner link! -->
42 <invoke name="processClaim" partnerLink="claimProcessor"
   operation="execute">
43   <input part="input" variable="input"/>
44   <output variable="claimOK"/>
45 </invoke>
46

```

```

47 <!-- if claim is ok, publish the original input on a JMS
   topic -->
48 <switch name="pubIfOK">
49   <case>
50     <condition>claimOK</condition>
51     <bpelj:snippet name="createJMSMessage">
52       <!-- Use partner link topic publisher to allocate
53        a JMS message
54        and populate it with the claim input message.
55        Note "p_jmsPublisher" is the way to reference
56        the partner
57        link "jmsPublisher" in BPELJ -->
58     <bpelj:code>
59       jmsMessage=p_jmsPublisher.getSession().createText
60       tMessage(input);
61     </bpelj:code>
62   </bpelj:snippet>
63   <invoke name="PubClaim" partnerLink="jmsPublisher"
64     operation="publish"
65     <input part="message" variable="jmsMessage"/>
66   </invoke>
67 </case>
68 <otherwise>
69   <empty/> <!-- do nothing in this case -->
70 </otherwise>
71 </switch>
72 </sequence>
73 </process>

```

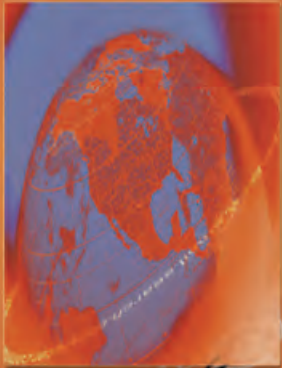
Listing 2: BPELJ sample

```

1 /**
2  * @bpelj:process process:
3  * <process name="MyProcess">
4  *   <variables>
5  *     <variable name="x" type="bpelj:Integer"/>
6  *   </variables>
7  *   . . .
8  *   <bpelj:snippet>
9  *     <bpelj:code>
10 *       x = self.getRandomValue ( );
11 *     </bpelj:code>
12 *   </bpelj:snippet>
13 *   . . .
14 * </process>
15 **/
16 public class MyProcessImpl implements Serializable
17 {
18   Integer getRandomValue( )
19   {
20     return new Integer(myRand.nextInt ( ));
21   }
22 }

```


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



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

AppDev™ VXML for BEA's WebLogic™ Workshop

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



For additional information:

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

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

Info@sandcherry.com
www.sandcherry.com

Download
30 Day Free Trial
www.sandcherry.com



Configuring Eclipse for Remote Debugging a WebLogic Application

A TUTORIAL

By Deepak Vohra &
Ajay Vohra

A J2EE application deployed in the WebLogic server may be debugged in the Eclipse IDE with the remote debugger provided by Eclipse. Without a debugger the error message has to be obtained from the application server error log to debug the application.

With the remote debugger provided by Eclipse, exception breakpoints may be added to the application file to debug. When an application is run in WebLogic and the application generates an error, the application gets suspended and the Eclipse IDE Debug perspective displays the error. In this tutorial we will debug a WebLogic Application Server application in Eclipse.

To debug an application deployed in the WebLogic Server from Eclipse, start the WebLogic Server in debug mode and configure a remote debugging configuration in Eclipse. Next, connect the Eclipse remote debugger to the WebLogic Server and debug applications running in the server. We will develop an example servlet application and deploy the application in WebLogic. First, the servlet is run without any error and subsequently an error is introduced in the servlet to demonstrate the remote debugging feature in Eclipse.

Preliminary Setup

Download the WebLogic 8.1 Application Server and install it. Download the Eclipse 3.0 or Eclipse

3.02 zip file eclipse-SDK-3.0-win32.zip (www.eclipse.org/downloads/index.php) and install it as well.

Developing a WebLogic Application

After installing the WebLogic Server and the Eclipse IDE, develop a servlet application to run and debug in the WebLogic Server. The example servlet application consists of a doGet method, which prints out a String message to the browser. The example servlet, WebLogicServlet.java, is shown in Listing 1.

Create a directory structure for a Web application. Create a WEB-INF directory and a classes directory in the WEB-INF directory. Create a package directory servlets for the example servlet and copy the WebLogicServlet.java file to the servlets directory. Create a web.xml deployment descriptor for the Web application. Copy the web.xml file to the WEB-INF directory. Listing 2 shows the web.xml file.

The example servlet is mapped to URL pattern /weblogic. The structure of the Web application is illustrated below.

```

    /WEB-INF
    |   |
    web.xml classes
           |
           servlets
                |
                WebLogicServlet.class
  
```

The compiling, packaging, and deploying of the Web application is done in the Eclipse IDE with an Ant build.xml file. Develop an Ant build.xml file

Author Bio:

Deepak Vohra is a Sun Certified Java 1.4 Programmer and a Web developer.

Ajay Vohra is a senior solutions architect with DataSynapse Inc.

Contact:

dvohra09@yahoo.com
ajay_vohra@yahoo.com

that consists of targets to compile the WebLogicServlet.java and package and deploy the webapp.war Web application. Listing 3 shows the build.xml file.

Table 1 shows the properties of the build.xml file. Table 2 shows the file's targets.

Set the debug attribute of the javac task in the build target to true to enable compilation in debug mode. By compiling an application in debug mode the line number, which generates the exception in a WebLogic Server application, gets displayed in the Debug perspective. Create a new project in the Eclipse IDE. Select File>New>Project. The New Project frame gets displayed. In the New Project wizard select Java>Java Project. Click on the Next button. The New Java Project frame gets displayed. In the New Java Project frame specify a Project Name, EclipseWebLogic for example, and click on the Next button. In the Java Settings frame add a source folder to the project with the Add Folder button.

The New Source Folder frame gets displayed. In the New Source Folder frame specify a folder name, src for example. A source folder gets added to the project. A message prompt gets displayed to update the source folder and the output folder. In the New Java Project frame click on the Finish button to create the project. A new project gets added to the Eclipse IDE. Next, select File>Import to import the example servlet source folder to the project. In the Import Select frame select File System and click on the Next button. In the Import File System frame select the src folder and the build.xml file and then click on the Finish button. The servlet source files get added to the project.

Run the build.xml file to compile, package, and deploy the servlet Web application to the WebLogic Server. Right-click on the build.xml file and select Run>Ant Build. The Web application .war file webapp.war gets generated and is deployed to the WebLogic 8.1 Application Server applications directory. Next, start the WebLogic Server with the bin/run script. Invoke the example servlet in a Web browser with the URL <http://localhost:7001/webapp/weblogic>. The

WebLogicServlet runs in the WebLogic Server and the output gets displayed in the browser.

Configuring a Remote Debugging Configuration in Eclipse

To remote debug a WebLogic application in Eclipse start the WebLogic Server in debug mode. Set the WebLogic Server in debug mode by setting the debug options in the startWebLogic batch script file. The debugging provided by WebLogic is based on the Java Platform Debugger Architecture (JPDA). Set the JAVA_OPTS variable as:

```
set JAVA_OPTS= -Xdebug -Xnoagent
-Xrunjdpw:transport=dt_socket,address=8787,server=y,suspend=n %JAVA_OPTS%
```

Table 3 shows the different debug parameters. For further explanation of the debug settings refer to the JPDA documentation (<http://java.sun.com/j2se/1.4.2/docs/guide/jpda/>). To demonstrate the remote debugging feature of Eclipse, add an exception to the WebLogicServlet.java file. For example, add a NullPointerException to the WebLogicServlet.java class. Replace

```
out.println("Eclipse WebLogic Remote Debugging");
```

with

```
String str=null;
out.println(str.toString());
```

Next, configure a debug configuration for the Eclipse project. Select the Debug option in the Debug option list. The Debug frame gets displayed. In the Debug frame select the Remote Java Application node. Right-click on the node and select New. In the Debug configuration frame specify a name for the Debug configuration. Select the project that is to be debugged. Select the EclipseWebLogic project previously created in the Eclipse IDE. Select the default value for Connection Type. In the Connection Properties, specify localhost as the Host and specify the Port as the port that was specified in the startWebLogic batch script of the WebLogic server, 8787. Click on the Apply button. A remote Java application debug configuration gets added.

Next add exception breakpoints to the WebLogicServlet.java file. To demonstrate the remote debug feature of Eclipse a NullPointerException was added to the WebLogicServlet.java file earlier in this section. To add a breakpoint to the servlet class, Select Run>Add Java Exception Breakpoint. In the Add Java Exception Breakpoint frame select the NullPointerException. The NullPointerException breakpoint gets added to the servlet class. If a NullPointerException is generated in the servlet application in the WebLogic Server, the application gets suspended and the Debug perspective of the Eclipse IDE displays the exception.

Remote Debugging a WebLogic Application

After configuring a debug configuration for the example servlet application deployed in the WebLogic Server, we will debug the servlet application in the Eclipse IDE. Create a webapp.war Web application from the modified (with the NullPointerException) WebLogicServlet.class file with the build.xml file as explained in the "Developing a WebLogic Application" section. Start the WebLogic

TABLE 1

Property	Description
build	The build directory used to build the Web application
src	The src directory has the source files for the Web application
weblogic.deploy	The WebLogic directory in which the Web application is deployed
dist	The directory in which the Web application war file is generated
j2sdkee	The J2sdkee directory

Properties of the build.xml file

TABLE 2

Target	Description
init	The initialization target to create the Web application directories
compile	Target to compile the Web application
webapp	Target to generate .war file

Targets of the build.xml file

TABLE 3

Parameter	Description
-XDebug	Enables debugging
-Xnoagent	Disables previous debugging agent
-Xrunjdpw	Specifies the connection mechanism, the transport address, and server and suspend values

Debug parameters

Server. The server starts in debug mode with the debug options specified in the startWebLogic batch file.

Next, select the EclipseDebug debug configuration in the Debug frame. Click on the Debug button to connect the remote debugger to the WebLogic Server. The Eclipse remote debugger gets connected to the WebLogic Server. Select the Debug Perspective button to display the debug perspective of the Eclipse IDE. In the Debug perspective the remote debugger is shown connected to the WebLogic server at localhost, port 8787. Access the WebLogicServlet in the WebLogic Server with the URL <http://localhost:7001/webapp/weblogic>. Because the servlet has a NullPointerException, the servlet gets suspended with a NullPointerException in the Debug perspective. The line that produced the exception gets displayed.

The line that throws the exception is the `out.println(str.toString());` code line. The servlet application may be debugged with the different debug options listed by selecting Run in the Eclipse IDE.

Summary

In this tutorial a WebLogic application was debugged in the Eclipse IDE. A WebLogic application may also be debugged from another IDE such as JDeveloper. 🍷

Listing 1

```
package servlets;
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class WebLogicServlet extends HttpServlet {
    public void doGet(HttpServletRequest request,
        HttpServletResponse response)
        throws ServletException, IOException {
        PrintWriter out = response.getWriter();
        out.println("Eclipse WebLogic Remote Debugging");
    }
}
```

Listing 2

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://java.sun.com/xml/ns/j2ee" version="2.4"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">
  <servlet>
    <display-name>WebLogicServlet</display-name>
    <servlet-name>WebLogicServlet</servlet-name>
    <servlet-class>servlets.WebLogicServlet</servlet-class>
  </servlet>
```

```
<servlet-mapping>
  <servlet-name>WebLogicServlet</servlet-name>
  <url-pattern>/weblogic</url-pattern>
</servlet-mapping>
</web-app>
```

Listing 3

```
<project name="weblogicApp" default="webapp" basedir=".">
  <property name="build" value="build"/>
  <property name="src" value="." />
  <property name="weblogic.deploy"
    value="C:\BEA\user_projects\domains\mydomain\applications"/>
  <property name="dist" value="dist"/>
  <property name="j2sdkee" value="C:\J2sdkee1.4"/>
  <target name="init">
    <tstamp/>
    <mkdir dir="${build}" />
    <mkdir dir="${dist}" />
    <mkdir dir="${build}/WEB-INF" />
    <mkdir dir="${build}/WEB-INF/classes" />
  </target>
  <target name="compile" depends="init">
    <javac debug="true" classpath="${j2sdkee}/lib/j2ee.jar"
      srcdir="${src}/WEB-INF/classes" destdir="${src}/WEB-INF/classes">
      <include name="**/*.java" />
    </javac>
    <copy todir="${build}/WEB-INF">
      <fileset dir="WEB-INF" >
        <include name="web.xml" />
      </fileset>
    </copy>
    <copy todir="${build}/WEB-INF/classes">
      <fileset dir="${src}/WEB-INF/classes" >
        <include name="**/WebLogicServlet.class" />
      </fileset>
    </copy>
  </target>
  <target name="webapp" depends="compile">
    <war basedir="${build}" includes="**/*.class"
      destfile="${dist}/webapp.war" webxml="WEB-INF/web.xml"/>
    <copy file="${dist}/webapp.war" todir="${weblogic.deploy}"/>
  </target>
</project>
```


A photograph of two men in an office setting. The man on the left, wearing a blue and white striped sweater, is leaning over the desk and pointing at the computer screen. The man on the right, wearing a plaid shirt, is sitting at the desk and looking at the screen. The background shows office cubicles and desks.

**Create software so brilliant
it can manage itself.**

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

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

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



i n v e n t



By Murali Kashaboina & Bin Liu

Author Bio:

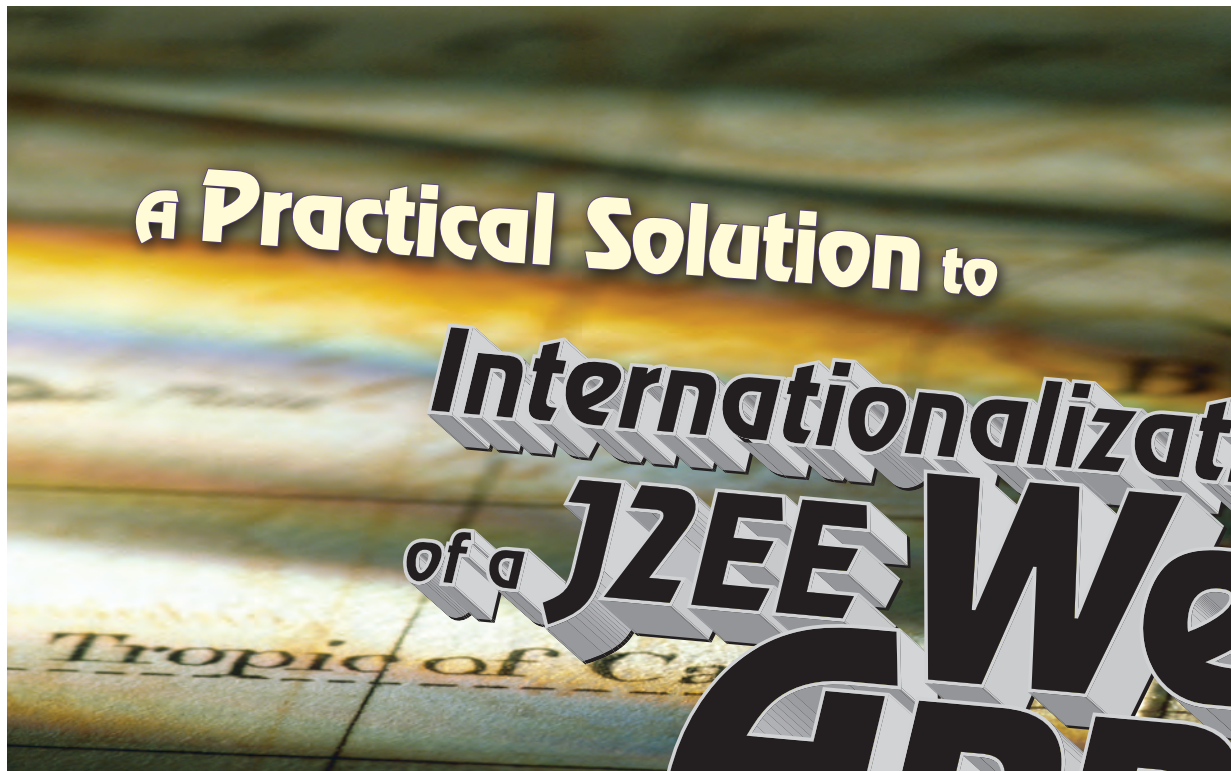
Murali Kashaboina is a senior solutions architect at United Airlines. Murali has more than eight years of enterprise software development experience utilizing a broad range of technologies, including J2EE, CORBA, Tuxedo, and Web services. Murali has published articles in WLDJ and SilverStream Developer Center.

Bin Liu is a lead software engineer at United Airlines. Bin has more than seven years of experience developing distributed applications using J2EE technologies, Web-Logic, Tuxedo, C++, and Web services. Bin has previously published articles in WLDJ.

Contact:

murali.kashaboina@united.com

bin.liu@united.com



MAKING WEB APPLICATIONS MULTILINGUAL

As the Internet crawls to even more remote corners of the globe, the internationalization

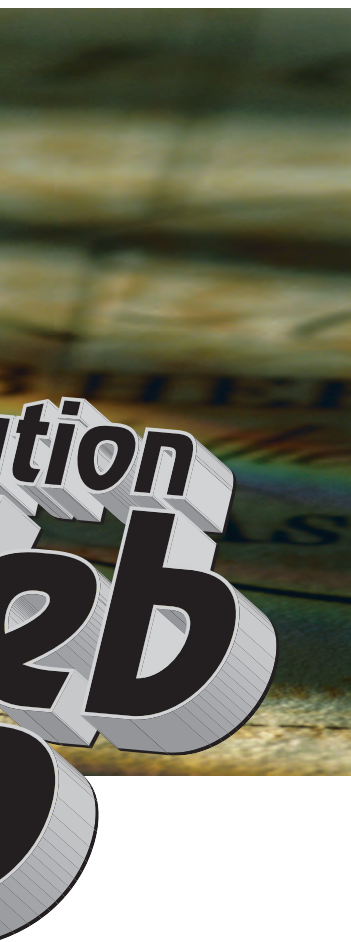
of Web applications exposes a plethora of challenges.

As a real-world example, if an airline starts reaching far more remote destinations across international frontiers, a Web application representing the airline's ecommerce will face numerous challenges in terms of internationalizing the ecommerce itself. These challenges result from many causes. For instance, one basic thing that differs from one country to another is the spoken language. Among countries where the same language is spoken, the colloquiality of the language differs. For example, in some parts of the world, the word "baggage" is widely used to represent personal belongings such as suitcases of a traveler. In some other parts of the world, the same is referred to as "luggage." In terms of ecommerce, there are numerous variations from country to country such as different currencies, different tax laws, different forms of

payment, and – most important – different business rules governing the application.

Essentially, there are two parts to the internationalization of a Web application. The first part is internationalization of the application code. This involves preparing the code so that it can adapt itself to new languages and regions. In practice, this preparation involves the separation of text, labels, display messages, and any other data that is sensitive to language and region of the world. This type of adaptation of code enables generalization of the product in such a way that it can handle new languages and countries without any re-design. The second part is localization of the application. This involves actual adaptation of the internationalized code to a specific language or region (aka locale). In practice, localization involves creation of translated text, labels, and messages, and the addition of any other application data that is specific to a certain locale.

Internationalization is a common problem



that typically gets a blind eye turned toward it during design and development. Internationalization design must be up-front work in the development life cycle and not an afterthought. It is rightly said, “a stitch in time saves nine.” It may not be too easy to design for internationalization up front; however, it will be far more difficult to incorporate internationalization at a later stage when the application has already been developed. Up-front planning for application internationalization can save significant amounts of time and money. There could be myriad ways of addressing this problem; however the following approaches are widely used:

- **Creating internationalized pages that retrieve locale-dependent content using custom tags.** This approach is typically employed if all of the pages consistently follow the same structure and look and feel across different locales. This approach also provides for easy maintenance and future enhancements across all of the locales. This approach may also employ a single source for business logic components that process the logic based on the locale.
- **Creating separate locale-specific pages.** This approach is typically employed if the structure and the look and feel of the pages differ significantly across locales. In this case, there may be separate business logic components for each locale.

- **Using portal technology. Vendors such as BEA provide support for portal technology.** For example, BEA WebLogic Application Server has excellent support for developing internationalized portals in the form of a set of custom tags that can be incorporated within the portal pages. Portal technology is widely used and is definitely an excellent candidate for implementing internationalized Web applications.

This article will delve into a real-time application against the backdrop of a Web-based airline-booking engine that has internationalization requirements complemented by a content management system. The implementation approach is aligned with the first option described above, where a single set of JSP pages are developed that work with locale-dependent content. The presentation tier of the application was built using existing frameworks such as Struts and JSTL custom tags. Both the Struts framework and JSTL custom tags offer internationalization support by providing mechanisms that are built upon the standard Java internationalization classes such as Locale, Resource Bundles, etc. The article will discuss in detail the technicalities involved in extending these frameworks to internationalize the Web presentation by incorporating means to retrieve localized content dynamically from the underlying content management system.

The Fundamental Concepts

Before we delve much into the implementation details, it is worth browsing through some key concepts. Terms such as “character,” “character sets,” “character codes,” “character encoding,” and so on are often heard when people talk about internationalization.

A character is the smallest component of a written language that has a specific name and some semantic value. Each character can have more than one graphical representation. For example, character “A” can be graphically represented as “A,” “A,” or “A.” Independent of the graphical representation, the meaning of the character remains the same. Each such graphical representation is called as a glyph. A set of glyphs is called a font. So a character will have a different glyph in different fonts. A character set comprises of a group of related characters that can be used for some purpose. All the characters on an “English” key board can be grouped into a character set because they provide ability to develop meaningful and informative documents in “English.”

Computers do not understand characters automatically but rather need a coded set of characters to process the data. In a coded character set, each character is assigned with an integer value commonly referred to as code point. American Standard Code for Information Interchange (ASCII) is a good example of a coded character set. ASCII is a small coded character set that comprises 127 characters. There are other coded character sets such as ISO-8859-1 and Unicode. Essentially, the code point of a character in the coded character set is used to identify the right glyph to display on the computer screen.

Character set encoding is yet another term that is widely used. A character set encoding scheme is a set of rules for mapping byte sequences (aka octets) to character code values and vice versa. Coded character sets such as ISO-8859-1, UTF-8, and UTF-16 have their own encoding schemes. For example, different schemes encode the character “ß” into byte sequences as shown in Table 1.

The terms “coded character set” and “coded character set encoding” have different meanings and should not be used interchangeably. To avoid this confusion, the short name “charset” is usually used to represent coded character set encoding. Table 2 shows some of the charsets that support different languages.

TABLE 1

Character: ß Code Point: Decimal= 223 Hex=0xDF Unicode=U+00DF			
Encoding Scheme	Byte sequences	Bytes in Hexadecimal	Number of Bytes
ISO-8859-1	11011111	0xDF	1
UTF-8	11000011 10011111	0xC3 0x9F	2
UTF-16	00000000 11011111	0x00 0xDF	2

Character encoded into byte sequences

TABLE 2

Language	Charsets (Coded Character Set Encodings)
Chinese (Simplified/Mainland) (zh)	GB2312
English (en)	ISO-8859-1
French (fr)	ISO-8859-15
German (de)	ISO-8859-15
Russian (ru)	ISO-8859-5
Romanian (ro)	ISO-8859-2
Hindi (hi)	DEVANAGARI
Japanese (ja)	ISO-2022-JP

Charsets and the languages they support

TABLE 3

Site	POS	Language	Synonyms for labels
Core	US	English	Adjusted fare
Discounted	US	English	Discounted fare

Label display for discounted fares

Table 2 leads to a big question: what character set should be used to support multiple different languages in an internationalized application? For example, the ISO-8859-1 character set will not support Chinese characters that are actually supported by the GB2312 character set. Obviously, there should be a common character set that can encode all of the characters in different languages of the world. Unicode is one such coded character set that promises to provide a unique code point for every character in every language. Java uses Unicode to encode characters. JRE 1.4 supports Unicode 3.0. Unicode is a large character set composed of almost 65,000 characters covering almost all world languages. Unicode encodes characters in 2 bytes, i.e., Unicode is 16-bit encoding with a range of code points from U+0000 to U+FFFF, represented in Unicode hexadecimal.

There is one more character set, known as the Universal Character Set (UCS), which can support all language characters and symbols. However, UCS uses a 31-bit encoding scheme that is not supported by most of the computer applications, whereas 16-bit encoding is widely supported. To address this issue, new transformed encoding schemes have been created based on Unicode and UCS. One of them is UTF-8 (UCS Transformation Format). UTF-8 transforms UCS characters into 1, 2, 3, or 4 byte encodings. UTF-8 preserves ASCII codes and encodes an ASCII character as a single byte. In essence, UTF-8 uses multi-byte encoding to represent characters in 1-4 bytes (octets).

The UTF-8 support for a wide range of characters and the efficient way of encoding makes it the de facto character set that should be used for displaying multiple languages. The application described in this article uses UTF-8 everywhere there is a need to encode content in different languages.

The Internationalization Requirements for the Example Application

The application described in the article is a Web-based airline-booking engine that has points of sale (POS) in different countries. The requirement was to support a number of POS countries (24 all together) such as the US, Germany, the UK, Japan, Korea, Brazil, Canada, China, Uruguay, etc., with room to expand to other countries of the world. For each POS country, the requirement was to provide a list of preconfigured languages specific to each POS in such a way that a user could select a particular language from the list in order to display content in that language. The requirement was to support a number of languages (10 all together) such as English, French, German, Chinese, Japanese, etc., with room to accommodate any other language in the future. By default, when a user lands on the application in a particular POS country, the content is expected to be displayed in the native language of that POS country.

There was another twist to the requirements. There were different flavors of the booking engine, commonly referred to as booking engine sites. For example, there is a core booking site and a discounted booking site. Different sites will have different business rules. The business rules for a particular site can also

vary across different POS countries. For example, the fare rules for a core booking site can be different on the US POS than are those on the UK POS. There were also special requirements in terms of the content displayed. It was expected that words with similar meanings for the same label in a particular language could be used across different sites. For example, Table 3 shows how the label for discounted fares was expected to be displayed on Core and Discount booking sites for the same US POS.

Another interesting requirement was to respect the colloquiality of the language based on the POS country. For example, if the user is in the Canadian POS and if the content is displayed in French, the colloquiality of French as spoken in Canada should be respected. However, if the user is in the French POS and if the content is displayed in French, then the colloquiality of the French as spoken in France should be preserved.

The final and the most critical requirement is that all the content that is localized will be managed through a content management application (CMA).

The aforementioned requirements clearly indicate that the content displayed on the Web application varies based on the POS country, user selected language, and the booking site. Based on this understanding, the following fallback rules to look up content from CMA were defined:

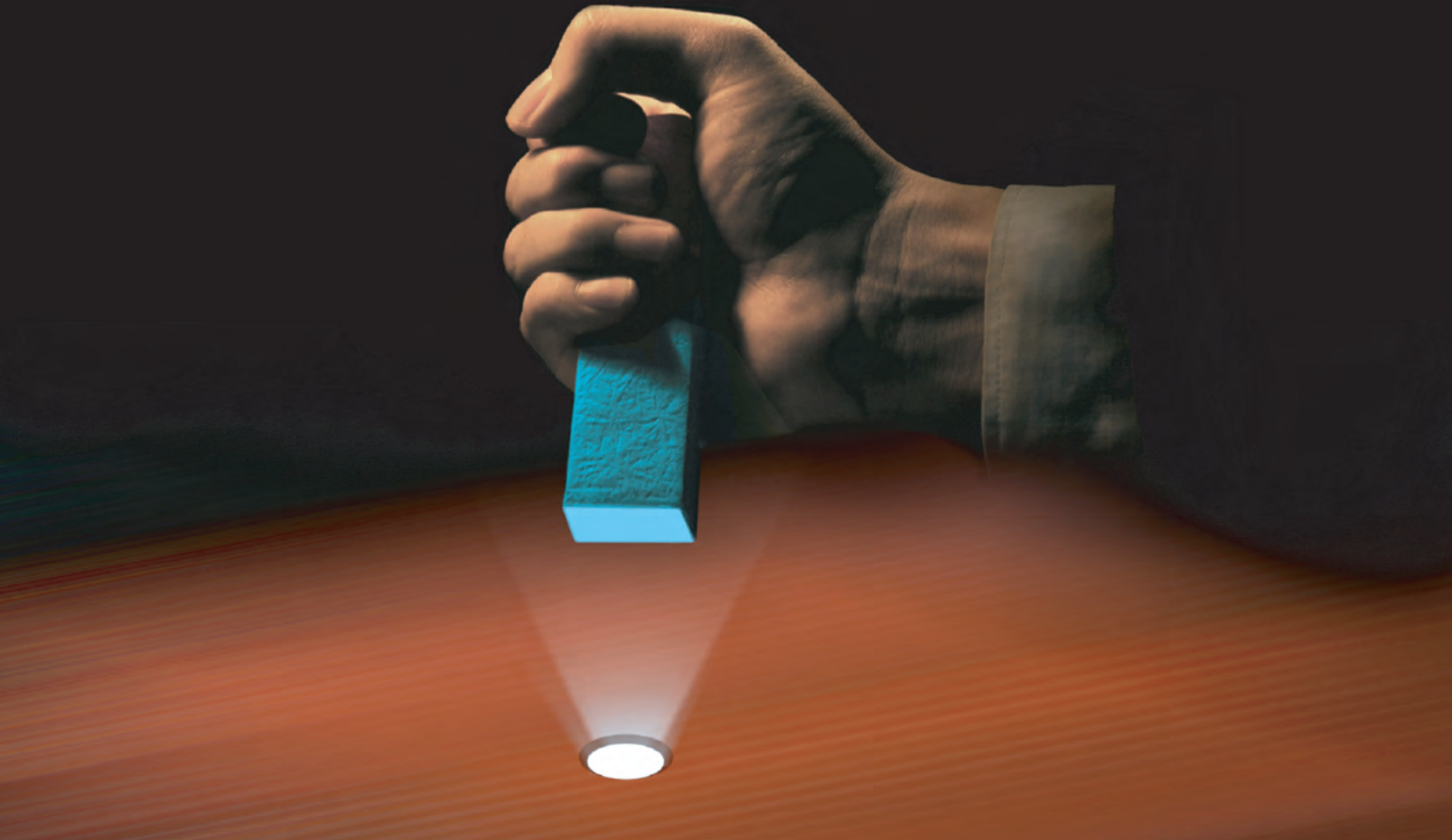
- Consider “core” as default site, “English” (en) as default language, and “US” as default country
- Find the content with exact match by using user selected site, language, and country
- If the previous step fails, find the content by using user-selected site and language, but ignoring the country
- If the previous step fails, find the content by using default site, user-selected language, and country
- If the previous step fails, find the content by using default site and user-selected language, but ignoring the country
- If the previous step fails, find the content by using user-selected site, default language, and default country
- If the previous step fails, find the content by using default site, default language, and default country
- If all of the above steps fail, then the content should be considered to be unavailable

The application that is presented in this article is a prototypical version of the actual application. The prototype helped us to focus on the subject and to provide clear explanations in this article.

The Foremost Thing to Do

The foremost thing to do in terms of internationalization is to identify the locale. The knowledge of a user's locale is the key to application internationalization. Java provides the Locale class in the java.util package to represent a user's locale. The Locale object uses ISO constants for countries and languages to determine a user's geographical, political, and cultural preferences. The Locale object also provides a placeholder for specifying a variant. Essentially, a variant helps in creating a custom locale by specifying additional information that can influence user preferences.

Typically, in the case of Web applications, a browser transmits a header “Accept-Language” in the HTTP request. This header contains more than one user-selected locale, and each locale is a combination of ISO language code and country code. Users normally have options on the browser to select a list of preferred



Think Crystal is a good fit for your Java application? Think again.

Think JReport.

Forcing a Windows reporting solution into a J2EE environment is never going to result in a perfect fit. Only JReport is built from the ground up to leverage J2EE standards and modular components for seamless embedding in any Web application.

JReport is ready out-of-the-box, with all the tools necessary to empower your application with any reporting requirement. From reusable, shared report components to flexible APIs, JReport is the most complete embedded reporting solution available.

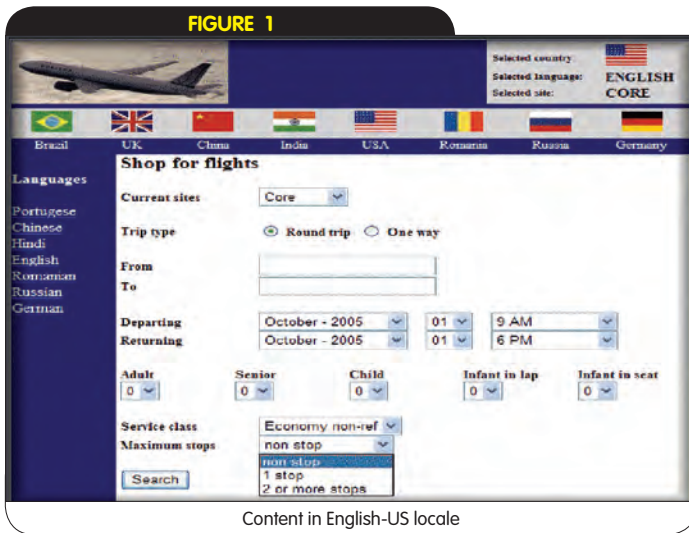
With the ability to scale to multi-CPU and server clusters, JReport is a perfect fit for any reporting workload. Load balancing and failover protection provide peak performance and uninterrupted access to critical business data. In addition, JReport integrates with any external security scheme for single sign-on.

JReport's ad hoc reporting lets users access and analyze data on demand, from any browser. And, with cascading parameters, embedded web controls for dynamic sorting and filtering, drill-down and pivot, JReport ensures users get the information they want, when they want it, and how they want it.

See for yourself why over half of the world's largest organizations have turned to JReport to enable their J2EE applications with actionable reporting.

When it comes to embedded Java reporting, JReport is the perfect fit. Download a FREE copy of JReport today at www.jinfont.com/wj4.





Content in English-US locale

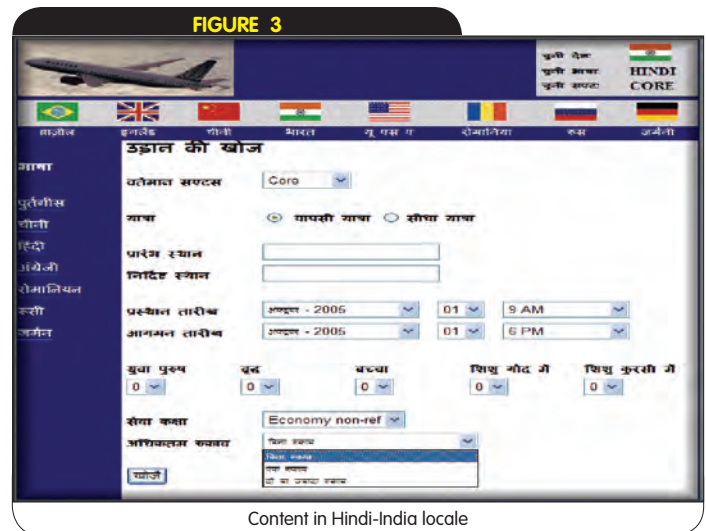


Content in German-Germany locale

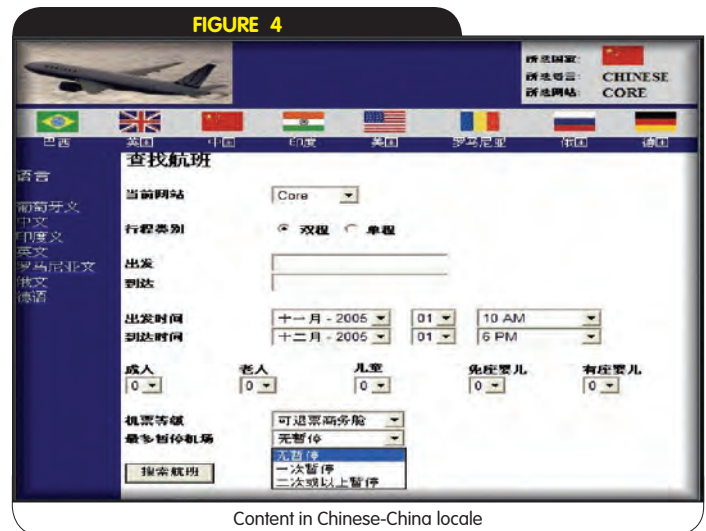
or formatted based on user's locale:

1. Elements that can be translated:
 - Application messages
 - Labels and headers on the form fields
 - Online help content
 - Images and icons
 - Personal titles and greetings
2. Elements that can be formatted:
 - Date and time
 - Numbers
 - Measurements (miles vs. km, etc.)
 - Currencies
 - Phone numbers
 - Mailing address

Java provides the ResourceBundle class in the java.util package to handle translatable data elements. A resource bundle in its simplest form is a collection of key-value pairs (a key is an index to a value that can be localized or translated). Property files are typically used to specify resource bundles. Custom resource



Content in Hindi-India locale



Content in Chinese-China locale

language and country combinations. However, while selecting a locale to be used in the application, it is not reliable to depend totally on the value of this HTTP header because a user may or may not specify the right locale supported by the application. One way is to provide the user with the ability to select language and country preferences dynamically while interacting with the application.

In the application described in this article, it was not too difficult to determine what constitutes a user locale. As mentioned earlier, the content to be displayed is based on the user's choice of language, POS country, and booking site. Obviously POS country and user's choice of language can be easily represented in terms of ISO codes. The question was about the booking site. To address this, a meaningful but constant name was assigned to each booking site such as "core," "discount," etc., and this site name was considered as a variant in a Locale object.

Handling Localized Content

The important step in internationalization is the isolation of data elements that are candidates for localization. The following are some of the displayable data elements that can either be translated

bundles can also be created as Java classes by extending from the `ResourceBundle` class. In either case, the naming of the properties file or the custom class matters most. Resource bundle names have two parts: a base name and a suffix. The base name is an identifier for the resource bundle. The suffix is an identifier for the locale. For example, a properties resource bundle “AppResources” for the default locale will have a name “AppResources.properties.” However, if the property values are translated to Chinese and French locales, then the new files will have names “AppResources_zh_CN.properties” and “AppResources_fr_FR.properties,” respectively. If the application uses a custom locale with some variant such as the one used in the example application “fr_US_core,” then the locale suffix should also contain the variant name, for example, “AppResources_fr_US_core.properties.” One note of caution: when properties’ resource bundles are being created, the right character set encoding should be used.

Resource bundles are loaded within the application by calling the “`getBundle`” static method on the `java.util.ResourceBundle` class and passing the base name of the bundle and the current locale. The loading of a resource bundle uses a locale fallback approach as shown in the flow diagram in Figure 5.

Creating property files with translated content is an approach typically used when the application is of medium size and the number of locales supported is small. However, in cases of large applications that need internationalization in many different locales, this approach may end up being a maintenance nightmare. For example, in the current application there are as many as 1200 possible custom locales (24 countries x 10 languages x 5 booking sites = 1200 custom locales), and there are thousands of data elements that need to be translated. Of course, most of the translated content in a particular language will be reused, but the idea of creating 1200 different property files with thousands of content entries is mind-boggling. However, the requirement to use CMA for localized content management and retrieval essentially eliminated any possibility of internationalization design based on locale-specific property files. It was inevitable to devise a strategy to pull the content without losing the internationalization features offered by Java and the Web frameworks such as Struts and JSTL.

Obviously, `Locale` and `ResourceBundle` are at the core of the internationalization features offered by Struts and JSTL. It was clear that a custom `ResourceBundle` instance should be created that can interact with the underlying CMA database to pull locale-specific content. The next question was whether there was a need to create a separate custom Resource bundle for each possible locale, particularly because of the way Resource bundles are looked up. If we took this approach, it meant that we had to create as many as 1200 unique Resource bundle implementations, one for each possible locale. This solution would not be any better than having hundreds of property files, so the conclusion was to create only one custom implementation and somehow incorporate it in the application.

The Solution

When the challenges were completely understood, it was decided that a custom `Locale` and a custom Resource bundle should be used while providing a solution. The custom Resource bundle will be more like a property resource bundle, in which case a key will be used to retrieve a corresponding value based on the custom `Locale`. Since this custom resource bundle was expected to

be CMA database-centric, essentially the key will be an index into the content in the CMA database supplemented by a custom locale.

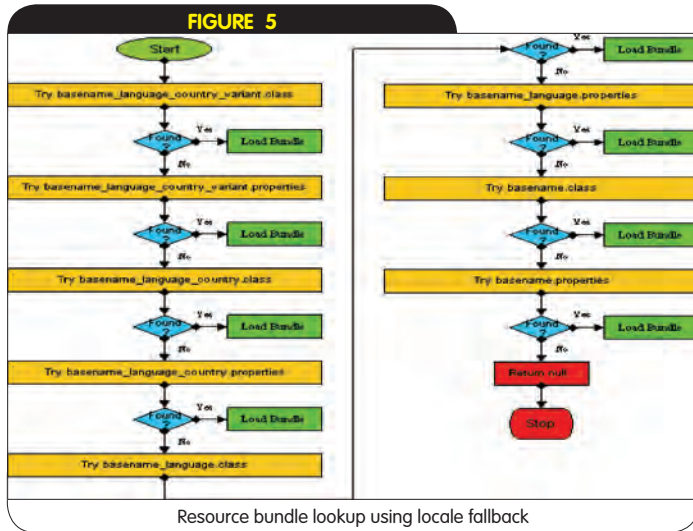
The foremost thing we ensured was that the underlying CMA database schema supported retrieval of content based on a content key and a custom locale. The existing CMA database schema did not fully support what we were looking for in terms of a content key. We had to come up with extensions without disturbing other parts of the schema so that localized content could be retrieved by using a unique content key. Based on these extensions, the CMA UI interface was also modified to provide the ability to create content with a unique content key in multiple different languages and the ability to target the content to a particular locale. Figure 6 is a simplified view of the extensions made to the schema with the existing design constraints in place.

The custom implementation of the database-centric Resource bundle was pretty straightforward. The custom Resource bundle class “`DynaResourceBundle`” was created by extending `java.util.ResourceBundle` and by overriding the “`handleGetObject`” method. The “`handleGetObject`” method is called internally when either the “`getString`” or the “`getObject`” method of the Resource bundle is invoked. An instance of this class will be created by passing the custom locale and a String array containing resource names. A constant resource name “`CMA`” was used to indicate to the bundle to pull content from the underlying CMA database. All other resource names were considered as non-CMA resources. For all non-CMA resources, the usual `ResourceBundle.getBundle` method was first used to load the bundle using the resource name and the current locale, and then the content value was retrieved using the key. The resources are tried out in the order they are found in the array while retrieving the content. If the current resource in the array returns null, then the next resource is used to retrieve the content. This is a simple failover approach in case content is not found in one resource. Listing 1 shows the concrete implementation of this method.

Please note that “`Label`” is a data object representing a label entry in the CMA database. The “`handleGetObject`” method internally uses a “`ContentBO`” instance while retrieving content from CMA. The “`ContentBO`” is a content retrieval business helper object that actually interacts with the data access objects to retrieve content from CMA. The “`ContentBO`” provides a “`getLabel`” method that looks up a “`DAOFactory`” class to get a reference to the DAO implementation for labels and invokes the “`getContent`” method on the DAO by passing the content key and the locale. The “`ContentBO`” also has a static method – “`getFallbackLocales`” – that implements the business requirement for the content look-up fallback logic approach as described earlier. It turns out that this requirement translates into the fallback of custom locales in our design and fits well with the approach we took. So ultimately our choice of the custom locale with “`site name`” as a variant paid rich dividends. Listing 2 is a code snippet for the “`getLabel`” method in the `ContentBO` object.

The “`LabelDAO`” class is extended out of an abstract DAO class and provides the concrete implementation for retrieving the content from the database. This class implements the “`getContent`” method; the content key and the custom locale are the arguments to this method.

```
class LabelDAO extends AbstractDAO {
    public CMAObject getContent( String contentKey, Locale locale ) throws
```



```
SQLException{
    .....
}
}
```

Note that this class has package-level access in the package “com.ual.i18n.content.dao” since an instance of this class is actually created by the DAOFactory that is part of the same package. The method “getContent” simply retrieves the language, country, and site name from the locale object, builds the SQL using the content key, and executes the SQL to retrieve the label from the CMA database. Listing 3 shows a code snippet from the method “getContent.”

Once the text of the label is retrieved from the database, this method invokes the inherited method “getEncodedText” by passing the retrieved text and “UTF-8” as the character set encoding. The following code snippet is for the “getEncodedText” method defined in the “AbstractDAO” class:

```
public String getEncodedText( String text, String encoding ) throws Un-
supportedEncodingException {
    if( text == null ){
        return null;
    }

    String encodedText = new String( text.getBytes( "8859_1" ),
encoding );

    return encodedText;
}
```

The “getEncodedText” method converts the text String retrieved from the database to a String encoded with UTF-8 charset. This conversion is required because by default a String is constructed from a sequence of bytes by using the underlying platform’s default character encoding. The specification of “8859_1” as a transit encoding to retrieve a byte array from the text String is only a precautionary measure, because a String object’s default “getBytes()” method encodes text into a sequence of bytes using the platform’s default charset. The behavior of this method when the

String cannot be encoded in the default charset is unspecified.

Figure 7 summarizes the method calls required to retrieve a localized label from the “DynaResourceBundle” class while retrieving the content from a CMA database.

Internationalization with Struts

As expected, Struts internationalization support is based on use of Locale class and Resource bundles. However, Struts slightly differs in terms of handling resource bundles. Struts uses the “MessageResources” class and its sub class “PropertyMessageResources” in the “org.apache.struts.util” package to represent resource bundles. Even though Struts web application developers normally do not directly work with an instance of “MessageResources,” it provides an API to retrieve localized messages. Struts uses this API internally to provide appropriate messages to the framework clients. One of the reasons why Struts uses message resources to represent resource bundles is because Java’s resource bundles are not serializable and Struts wanted to create its own serializable representations. Struts requires that any object it manages internally be serializable because a Struts-based Web application can be deployed in a clustered server environment where session data replication could be enabled.

The big question at that point was how to make our own custom resource bundle “DynaResourceBundle” work with Struts in a way that is transparent to the rest of the application. After much research, we could not find a direct way of incorporating “DynaResourceBundle” in Struts. We had to look into other options. Typically, Struts looks up the following configuration in the struts-config.xml to load an instance of the “MessageResources” class representing an “application.properties” file and retains it in memory for the life of the application.

```
<message-resources parameter="application" null="false" />
```

The promise of Struts has always been to enable the incorporation of custom “MessageResources” implementations that retrieve data from other sources such as a database. With this promise in mind, we moved forward with the idea of extending the “MessageResources” class in such a way that we could glue our custom resource bundle to the Struts framework. We created a custom class – “DynaMessageResources” – by extending “MessageResources” as shown in Listing 4.

The main methods that are overridden are the “getMessage” methods. The method that is internally called by the Struts framework is the overloaded “getMessage” method that takes locale, key, and an array of objects. The array of objects is typically passed to this method to replace indexed parameters in the actual messages. The indexed parameters {0}, {1} are placeholders in a message that can be replaced at run time with certain values.

TABLE 4

Tag Attribute	Tag Names
key	bean:message, html:option
formatKey	bean:write
altKey	html:button, html:checkbox, html:file, html:hidden, html:radio, html:select, html:submit, html:reset
titleKey	html:button, html:checkbox, html:file, html:frame, html:image, html:link, html:radio, html:select, html:submit, html:reset

Attributes of commonly used Struts locale-sensitive tags

THE MOST CUTTING EDGE DEVELOPMENT IN REPORTING SOFTWARE GIVES USERS SOME MUCH-DESIRED FREEDOM.



THE ONLY REPORTING SOFTWARE THAT LETS USERS DESIGN THEIR OWN REPORTS IN MSWORD. YES, REALLY.

Free Yourself from the Task of Report Design

At last! There's a way for developers to rid themselves of the daunting and time consuming task of trying to design report templates that satisfy business owners and managers. That's because Windward Reports enables them to design their own reports using popular, easy-to-use word processing programs like Microsoft Word — so there is no new software to learn.

Point. Click. Done.

Windward Reports utilizes Microsoft Word, the word processing software program that virtually everyone knows (and frankly, it works with just about every other word processing program, as well.) All a business owner needs to do is open up Word's deep, rich library of design templates and configure the design that they want. So designing a report is now as easy as creating any MSWord document. Your data just flows in and completes the picture, creating reports that will dazzle and impress. It's really that easy.

**Find Out for Yourself with a Free, No-Obligation Demo.
Go to www.windwardreports.com. You won't believe your eyes.**



The MessageResources class internally uses the “java.text.MessageFormat” class to format messages with run-time values. The MessageResources class provides a default implementation for the “getMessage” method with three arguments; however, one word of caution about this method: it caches the message format objects for each message in a local HashMap for the life of the application. This may prove to be undesirable if the custom message resources class is not supposed to cache because either there is no requirement to cache any messages or caching could be done outside of this class with much more control to refresh the cache on demand. This was the main reason why we had to override that method by eliminating the need to cache the message format objects in a local HashMap. However, the core function of the method was not taken away in the sense that it still performed formatting of the messages using the MessageFormat objects but without caching them. Note that this method internally calls the other overloaded method “getMessage” that takes two arguments: locale and the message key.

The “getMessage” method with two arguments is the method of interest. This is the method that binds our custom resource bundle with the Struts framework. When invoked, this method first looks up the “DynaResourceBundle” instance for the specified locale and calls the “getString” method on the bundle to retrieve the message for the specified key. This method uses a private method – “getBundle” – to look up an instance of the resource bundle for the specified locale. The private “getBundle” method manages a HashMap to store a locale-specific instance of “DynaResourceBundle” and creates a new instance of the bundle, if necessary. In this way, locale-specific instances of “DynaResourceBundle” are cached for the life of the application.

The sequence diagram shown in Figure 8 summarizes the main interactions.

In order to incorporate a custom “MessageResources” implementation in Struts, a custom “MessageResourcesFactory” should be created by extending the “org.apache.struts.util.MessageResourcesFactory” class. “MessageResourcesFactory” is an abstract class that declares an abstract method called “createResources.” In the current application, a factory, “DynaMessageResourcesFactory,” was created by providing implementation for the “createResources” method. The following is the code snippet for this class:

```
public class DynaMessageResourcesFactory extends MessageResourcesFactory{
    public DynaMessageResourcesFactory(){
        super();
    }

    public MessageResources createResources( String config ){
        return new DynaMessageResources(this, config, this.
returnNull);
    }
}
```

The custom factory is incorporated in the Struts framework by specifying the factory class in the Struts configuration file as follows:

```
<message-resources factory="com.ual.i18n.ui.core.DynaMessageResourcesFactory" parameter="CMAX, MessageResources" null="false"/>
```

Please note that the values specified for the attribute “parameter” are passed to the “createResources” method as config parameters. The Boolean value for the attribute “null” is set in the factory when the “setReturnNull” method is invoked on the factory at the time of its creation. The “createResources” method is where an instance of “DynaMessageResources” is created and glued into the Struts framework.

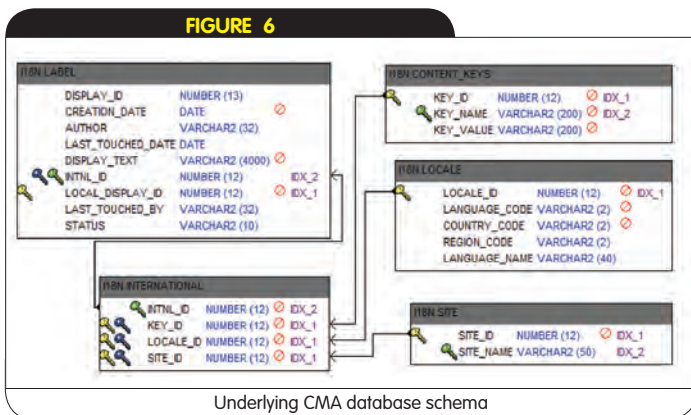
The major consumers of the messages retrieved from the “MessageResources” instance are the locale-sensitive JSP tags in the Struts framework. Table 4 shows some of the attributes of the most commonly used Struts locale-sensitive tags that retrieve messages internally from the underlying “MessageResources.”

Listing 5 is from the JSP page that uses some of these tags while retrieving content using content keys from the underlying “DynaMessageResources.”

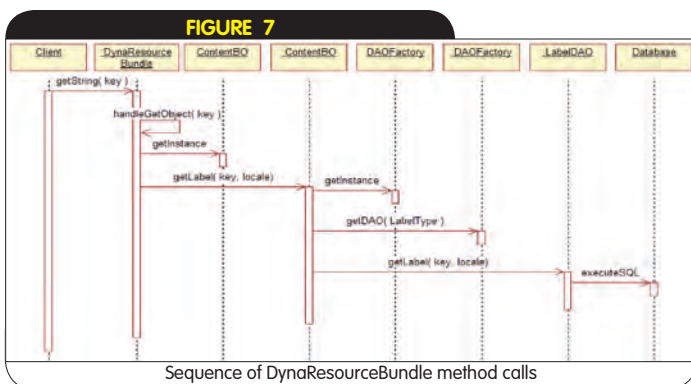
The most commonly used tag for writing out localized content is the Struts bean:message tag in the bean tag library. Please note that Struts by default sets up the user locale as an attribute in the user session using “Globals.LOCALE_KEY” as the key. Subsequently, the user locale can be retrieved at any point using the key. Figure 9 shows how this tag pulls the content from the underlying “DynaMessageResources” instance in the current example.

Localizing Other Struts Components

There are other components in Struts that participate in displaying localized messages while interacting with the underlying message resources instance. The important ones are the



Underlying CMA database schema



Sequence of DynaResourceBundle method calls

A man with a goatee and closed eyes is meditating in a server room. He is sitting cross-legged on the floor, wearing a light-colored polo shirt and khaki pants. The server racks are visible in the background, creating a sense of depth and perspective. The lighting is soft and focused on the man, with the server racks receding into the distance.

True application management starts from within.

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

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

 **motive**
www.motive.com

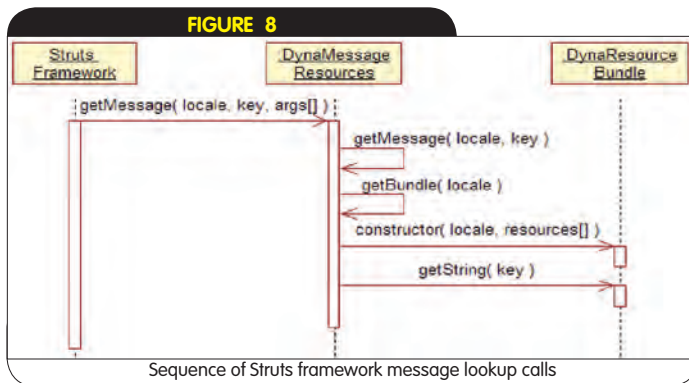
“ActionErrors” and the “ActionMessages” form validators and tiles configuration.

Typically, application-specific validations are performed within the Struts actions, and in many cases there will be a need to display correct error messages to the user. Instances of “ActionErrors” are used to relay the error messages to the JSP page. The following is a code snippet showing how an “ActionErrors” instance is constructed.

```
ActionErrors errors = new ActionErrors();
ActionError error = new ActionError( "ui.i18n.error.from.field.incorrect" );
errors.add( ActionErrors.GLOBAL_ERROR, error );
this.saveErrors( request, errors );
```

A content key such as “ui.i18n.error.from.field.incorrect” is used to represent the error message to be displayed to the user. On the JSP page, an “html:errors” tag is used to display all of the errors relayed to the page as shown in the following code snippet:

```
<html:errors id="error" >
    <bean:write name="error" /><BR>
</html:errors>
```



The important point to note is that internally, the html:errors tag uses the content key specified in the constructor of the “ActionError” and the current user locale to retrieve the localized error message from the underlying “MessageResources” instance.

The “ActionMessages” are similarly used to relay any localized messages to the JSP page. The html:messages tag is used to loop through each action message and retrieve the localized message from the underlying “MessageResources” instance.

One important feature of Struts is its inbuilt validation framework. The validation framework provides a declarative way of specifying validation settings for HTML form validations. The user input can be validated using settings specified in a XML configuration file. Each HTML form is represented as a <form> element and uses a name attribute corresponding to the form-bean name specified in the Struts configuration file. The validations for each HTML field can be declared under the <form> element using a <field> element. The validation settings include labels and message keys that will be used to display the messages in case the field validation fails. Listing 6 is an example declaration.

An interesting point is that the <form> element itself can be

specified per locale in case validation settings differ per locale as follows:

```
<form name="tripTemplateForm" locale="pt" country="BR" >
    .....
</form>
```

The configuration of tiles can be localized if the structure of the page layouts differs per locale. The localization of tiles configuration can be accomplished by following a naming convention similar to the one used for properties files. For example, if the Web application needs a separate tiles definition for a Brazilian Portuguese locale, then the naming of the tiles configuration should be “tiles_pt_BR.xml.”

Identification of the User Locale

Because our application was using a custom locale and not the browser locale, it was necessary to design a strategy to identify a user locale and use it for the life of the user session, unless a user explicitly selects a new locale by changing any of the custom locale attributes such as language, POS country, or booking site. It was decided that the first-time entry URL will be used to determine the initial user locale. Specifically, the user locale will be determined using the Servlet path of the URI as shown in the following example:

```
http://hostname/i18n/core/BR/pt/displayInput.do
```

In this example, the custom locale is determined from the Servlet path “/core/BR/pt,” indicating that the current request is for a core booking site with Brazil as POS country and Portuguese as the language in which to display content. This information should be retrieved at the entry point into the application in such a way that custom locale and resource bundle are populated even before any application processing occurs. The natural place to do this would be in the control Servlet. Because the current application is driven by Struts, the decision was made to create a control Servlet extending Struts’s ActionServlet class. The major responsibility of this class is to tap the user request and check for user locales before the request is mapped to a Struts action class. Listing 7 shows the class and its implementation of the “service” method.

Please note that, as a standard practice, a “SessionHelper” object is used by providing a standard-typed API to work with session attributes so that attribute names are not scattered across the application code, but rather centralized in a single object. The “SessionHelper” method that sets locales in a user session is the “setLocale” method. This method will be discussed in the next section.

Internationalization with JSTL

JSP standard tag libraries (JSTL) are built with inherent support for internationalization. The JSTL internationalization tags can be classified into the following two categories:

- Tags that support display of localized messages:
 - <fmt:setLocale>
 - <fmt:setBundle>
 - <fmt:bundle>
 - <fmt:message>

2. Tags that support locale-specific formatting

- <fmt:formatNumber>
- <fmt:formatDate>
- <fmt:parseNumber>
- <fmt:parseDate>

These internationalization tags work directly from the locale and current resource bundle settings in the JSTL configuration. JSTL provides the “javax.servlet.jsp.jstl.core.Config” class that governs these settings. Specifically, the current user locale is stored in the configuration setting using the “Config.FMT_LOCALE” key. The resource bundle to be used for the current locale is stored in the configuration setting using an instance of helper bean “javax.servlet.jsp.jstl.fmt.LocalizationContext.” This instance is configured using the “Config.FMT_LOCALIZATION_CONTEXT” key.

The <fmt:setLocale> tag enables explicit setting of the locale. It uses the “Config.FMT_LOCALE” key internally to store the locale object in the configuration so that other internationalization tags can use it. Optionally, the locale can be set in a particular scope using the “scope” attribute. The following is an example use of this tag:

```
<fmt:setLocale value="zh-CN" variant="discount" />
```

The <fmt:setBundle> tag enables explicit setting of a resource bundle for the current locale so that localized messages can be retrieved from that bundle. The mandatory “basename” attribute of this tag determines the resource bundle to load. Internally, this tag looks up the current locale using the “Config.FMT_LOCALE” key. It creates an instance of “LocalizationContext” to store the loaded bundle using the current locale, and stores that instance in the specified scope or by default in the page scope. The following is a typical use of this tag:

```
<fmt:setBundle basename="messages" scope="session" />
```

The <fmt:bundle> is a convenient tag to override the current localization context in such a way that a different resource bundle can be used within the nested scope of this tag. This tag uses the mandatory “basename” attribute to load the resource bundle using the current locale configuration setting. It then creates an instance of “LocalizationContext” to store the resource bundle and the current locale so that any other internationalization tags nested inside its body can avail themselves of that information. The following is the typical use of this tag:

```
<fmt:bundle basename="messages" />
```

The <fmt:message> tag is the major consumer of the underlying resource bundle and is the primary way of displaying localized messages in a JSP page. A stand-alone usage of this tag (shown in the following snippet) uses the underlying “LocalizationContext” instance to look up the resource bundle and retrieve messages using the value specified for the key attribute:

```
<fmt:message key="ui.i18n.label.shopForFlights" />
```

This tag has an optional “bundle” attribute that can be used to directly point to an instance of “LocalizationContext.”

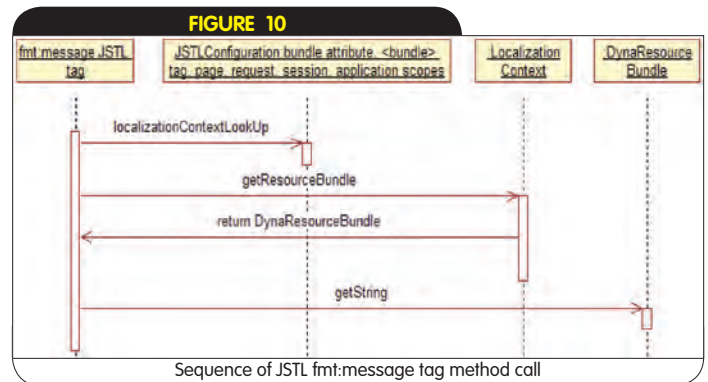
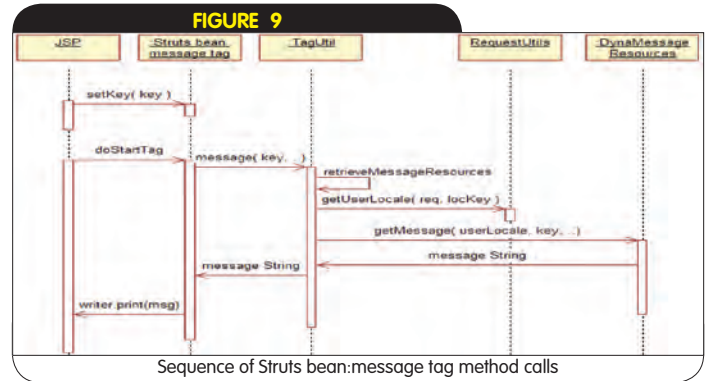
If this attribute is specified, this tag will not look up for the “LocalizationContext” in the JSTL configuration but will instead directly access the specified instance to retrieve the resource bundle. In general, the following is the look-up order used by the <fmt:message> tag to look up the resource bundle for localized messages:

1. The “LocalizationContext” instance specified by the “bundle” attribute
2. The “LocalizationContext” instance specified by <fmt:bundle> tag, if nested within the <fmt:bundle> tag
3. The “LocalizationContext” specified in the JSTL configuration either by <fmt:setBundle> or explicitly by custom Java code. The order of look up in this case is as follows:
 - a. Page scope
 - b. Request scope
 - c. Session scope
 - d. Application scope

Note that the key used to retrieve the message from the underlying resource bundle can be specified in the body of the tag. This helps in generating the key dynamically during run time, as shown in the following snippet:

```
<fmt:message>
<c:out value="ui.i18n.label.month.${month.month}" />
</fmt:message>
```

It is worth mentioning here that the default resource bundle to be used by the JSTL configuration can be specified in the “web.xml” file as follows so that there is no need to explicitly set the bundle either by using <fmt:setBundle> or by using custom Java code.



```

<web-app>
.....
  <context-param>
    <param-name>
      javax.servlet.jsp.jstl.fmt.localizationContext
    </param-name>

    <param-value>
      applicationMessages
    </param-value>
  </context-param>
.....
</web-app>

```

Tags that support locale-specific formatting use the current locale specified in the JSTL configuration through the “Config.FMT_LOCALE” key. The primary purpose of these tags is to format the data in a manner that is appropriate for the current locale. The `<fmt:formatNumber>` and `<fmt:formatDate>` are the most frequently used tags to display data in a locale-sensitive manner.

The `<fmt:formatNumber>` tag in its simplest form is used to display numbers formatted based on current locale. Optionally the “type” attribute can be specified to instruct the tag to print the numeric value as a regular number, currency, or percentage.

```
<fmt:formatNumber value="1000000.05"/>
```

Table 5 shows the formatting done by the above tag in different locales. The `<fmt:formatDate>` tag is used to format a date based on the current locale. The “value” attribute is used to specify the variable of `java.util.Date` to be formatted. The following code snippet shows an example:

```
<jsp:useBean id="today" class="java.util.Date"/>
```

Locale	Number format
en-US	1,000,000.05
de-DE	1.000.000,05
zh-CN	1,000,000.05
pt-BR	1.000.000,05
fr_FR	1 000 000,05

Formatting done in different locales

Locale	Date format
en-US	Oct 10, 2005
de-DE	10.10.2005
zh-CN	2005-10-10
pt-BR	10/10/2005
fr_FR	10 oct. 2005

Date formatting for different locales

Parameter name	Parameter value
NLS_CHARACTERSET	UTF8
NLS_NCHAR_CHARACTERSET	UTF8

An example of a character set encoding scheme used by the database

```

<font class="generalSmallLabel" >Today: </font>
<br/>
<font class="generalLabel"><fmt:formatDate value="{today}"/></font>

```

Table 6 shows the date formatting for different locales.

Incorporating “DynaResourceBundle” in JSTL

It was very easy to incorporate “DynaResourceBundle” into JSTL after understanding the internals of its internationalization support. Once the custom user locale was identified by the application based on the current user request, an instance of “DynaResourceBundle” was created and set explicitly in the JSTL configuration using the “setLocale” method in the “SessionHelper” object as shown in Listing 8.

Figure 10 shows how the `<fmt:message>` tag interacts with “DynaResourceBundle” to retrieve the localized messages.

Setting the Page Encoding Scheme

Setting the character set encoding is a final but critical step in making sure that the content is displayed correctly. For all JSP pages, the character set encoding was set using the “pageEncoding” attribute of the “page” directive. As discussed before, “UTF-8” was set as the character encoding scheme:

```
<%@ page pageEncoding="UTF-8" %>
```

If there were many JSP pages that use a common template JSP page, it would be easy to specify the page encoding in one single location in the common template JSP.

Encoding Scheme for the Database

If the localized content is stored in a database such as Oracle, it is important to make sure that the correct character set encoding scheme is specified for the database. Oracle gives options to specify the character set encoding to be used at the time of database creation. The character set encoding used by Oracle database can be verified by executing the following SQL:

```
select * from NLS_DATABASE_PARAMETERS
```

Among other things, the above SQL gives information about the character set property. The parameters “NLS_CHARACTERSET” and “NLS_NCHAR_CHARACTERSET” will indicate the character set encoding scheme used by the database. Table 7 shows an example.

Summary

The design for internationalization should be an up-front task and not an afterthought. The design and implementation of internationalization should be carried out after careful evaluation of the exact application internationalization requirements. The choice of locale plays an important role in the implementation strategy. The internationalization requirements in general will also help in determination of the implementation approach: a single set of JSPs for all locales or separate sets of JSPs per locale. Internationalization in a J2EE Web application can be accomplished by using many different methods and technologies. This article showed an example implementation using Struts framework and JSTL custom tags by extending the support provided by both in a manner suitable to the custom needs of

the application. In practice, it is easy to store and manage huge amounts of localized content in a database instead of storing it in application properties files. The right choice of character set encoding is a must for internationalizing a Web application.

Acknowledgements

We would like to thank our director Virgil Bistriceanu for his

continuous support and inspiration to write this article. He has monitored the design from day one and provided crucial feedback that helped us make necessary changes in the design. Because he is a hardcore technologist himself, he made sure that our design strategy aligned with the enterprise architectural direction. We also would like to thank our managers Helen Agulnik and Ken Kunz and all other team members who supported and participated in the implementation. 🍎

Listing 1

```
protected Object handleGetObject( String key ){
    if( key == null || key.trim().equals("") ) return "";

    Object result = null;
    for( int i = 0; i < _resources.length; i++ ) {
        String resourceName = _resources[i];
        if( CMA_RESOURCE.equals( resourceName ) ){
            try{
                Label label = ContentBO.getInstance().getLabel( key,
                _locale );
                if( label != null ){
                    result = label.getDisplayText();
                }
            }
            catch( Exception exception ){
                if( log.isErrorEnabled() ){
                    String errorMessage = "Error retrieving label from CMA
using key : " +
                    key + " : locale : " + _locale;
                    log.error( errorMessage, exception);
                }
            }
        }
        else{
            try{
                result = ResourceBundle.getBundle( resourceName, _locale
).getObject(key);
            }
            catch( Exception exception ){
                if( log.isErrorEnabled() ){
                    String errorMessage = "Error retrieving label from " +
resourceName +
                    " using key : " + key + " : locale : " + _locale;
                    log.error( errorMessage, exception);
                }
            }
        }

        if( result == null ){
            continue;
        }
        else{
            break;
        }
    }

    return result;
}
```

Listing 2

public Label getLabel(String contentKey, Locale locale) throws Excep-

```
tion {
    DAOInterface dao = DAOFactory.getInstance().getDAO( DAOFactory.
LABEL_DAO );
    Label label = (Label) dao.getContent( contentKey, locale );
    if( label == null ){
        ArrayList list = getFallBackLocales( locale );
        if( list != null ){
            int size = list.size();
            for( int i = 0; i < size; i++ ){
                Locale fallBackLocale = (Locale) list.get(i);
                label = (Label) dao.getContent( contentKey, fallBack-
Locale );
                if( label != null ){
                    break;
                }
            }
        }
        return label;
    }
}
```

Listing 3

```
String country = locale.getCountry();
String language = locale.getLanguage();
String siteName = locale.getVariant();

String sqlQuery = "SELECT * FROM LABEL label, INTERNATIONAL intl,"
+
" CONTENT_KEYS ckeys, LOCALE locale, SITE site" +
" WHERE intl.LOCALE_ID=locale.LOCALE_ID" +
" AND intl.KEY_ID=ckeys.KEY_ID" +
" AND intl.SITE_ID=site.SITE_ID" +
" AND label.INTNL_ID=intl.INTNL_ID" +
" AND ckeys.KEY_NAME='" + contentKey + "'" +
" AND lower(site.SITE_NAME)=lower( '" + siteName + "' )";

if( country.trim().length() > 0 ){
    sqlQuery = sqlQuery + " AND locale.COUNTRY_CODE='" + country +
"";
}

if( language.trim().length() > 0 ){
    sqlQuery = sqlQuery + " AND locale.LANGUAGE_CODE='" + language +
"";
}

public class DynaMessageResources extends MessageResources{

private static Log log = LogFactory.getLog("I18N_LOG");
private HashMap resourceBundleMap;

public DynaMessageResources( MessageResourcesFactory factory,
String config ){
```

```

        super( factory, config );
        resourceBundleMap = new HashMap();
    }

    public DynaMessageResources( MessageResourcesFactory factory,
String config, boolean returnNull ){
        this( factory, config );
        this.returnNull = returnNull;
    }

    public String getMessage(Locale locale, String key, Object
args[]) {
        if (locale == null){
            locale = defaultLocale;
        }
        String formatString = getMessage(locale, key);
        if (formatString == null) {
            return returnNull ? null : ("???" + key + "???");
        }
        MessageFormat format = new MessageFormat(escape(formatStr
ing));
        format.setLocale(locale);
        return format.format(args);
    }

    public String getMessage( Locale locale, String key ){
        if( key == null || key.trim().equals("") )
            return "";

        DynaResourceBundle bundle = getBundle( locale );
        String result = null;
        try{
            result = bundle.getString(key);
        }
        catch( MissingResourceException exception ){
            if( log.isEnabled() ){
                String errorMessage = "Error retrieving message for key :
+
                key + " : locale : " + locale;
                log.error( errorMessage, exception);
            }
        }
        return result;
    }

    private DynaResourceBundle getBundle( Locale locale ){
        String bundleKey = locale.getCountry() + locale.getLanguage()
+ locale.getVariant();
        DynaResourceBundle bundle = (DynaResourceBundle) resource-
BundleMap.get( bundleKey );
        if( bundle == null ){
            String [] resources = config.split( "," );
            bundle = new DynaResourceBundle( locale, resources );
            resourceBundleMap.put( bundleKey, bundle );
        }

        return bundle;
    }
}

```

Listing 4

public class DynaMessageResources extends MessageResources{

```

private static Log log = LogFactory.getLog("I18N_LOG");
private HashMap resourceBundleMap;

    public DynaMessageResources( MessageResourcesFactory factory,
String config ){
        super( factory, config );
        resourceBundleMap = new HashMap();
    }

    public DynaMessageResources( MessageResourcesFactory factory,
String config, boolean returnNull ){
        this( factory, config );
        this.returnNull = returnNull;
    }

    public String getMessage(Locale locale, String key, Object
args[]) {
        if (locale == null){
            locale = defaultLocale;
        }
        String formatString = getMessage(locale, key);
        if (formatString == null) {
            return returnNull ? null : ("???" + key + "???");
        }
        MessageFormat format = new MessageFormat(escape(formatStr
ing));
        format.setLocale(locale);
        return format.format(args);
    }

    public String getMessage( Locale locale, String key ){
        if( key == null || key.trim().equals("") )
            return "";

        DynaResourceBundle bundle = getBundle( locale );
        String result = null;
        try{
            result = bundle.getString(key);
        }
        catch( MissingResourceException exception ){
            if( log.isEnabled() ){
                String errorMessage = "Error retrieving message for key :
+
                key + " : locale : " + locale;
                log.error( errorMessage, exception);
            }
        }
        return result;
    }

    private DynaResourceBundle getBundle( Locale locale ){
        String bundleKey = locale.getCountry() + locale.getLanguage()
+ locale.getVariant();
        DynaResourceBundle bundle = (DynaResourceBundle) resource-
BundleMap.get( bundleKey );
        if( bundle == null ){
            String [] resources = config.split( "," );
            bundle = new DynaResourceBundle( locale, resources );
            resourceBundleMap.put( bundleKey, bundle );
        }

        return bundle;
    }
}

```


THE WORLD'S LEADING INDEPENDENT WEBLOGIC DEVELOPER RESOURCE

Helping you enable intercompany collaboration on a global scale

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



SAVE \$69.89 OFF
REGULAR NEWSSTAND PRICE
OFFER SUBJECT TO CHANGE WITHOUT NOTICE

***Only \$49.99 for 2 years (12 issues)**

Now in More Than 5,000 Bookstores Worldwide

Go Online & Subscribe **Today!** www.WLDJ.com

SYS-CON.COM/WEBLOGIC/

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



```

    }
}

```

Listing 5

```

<%@ taglib uri="http://struts.apache.org/tags-html" prefix="html" %>
<%@ taglib uri="http://struts.apache.org/tags-bean" prefix="bean" %>
<%@ taglib uri="http://struts.apache.org/tags-html-el" prefix="html-el"
%>
<%@ taglib uri="http://java.sun.com/jstl/core" prefix="c" %>
.....
<tr>
<td width="100" align="left" valign="middle">
    <label for="stops"><font class="formFieldLabel"><bean:message
key="ui.i18n.label.maxStops" /></font></label>
    </td>
    <td align="left" valign="middle">
        <html:select property="maxNumberOfStops" tabindex="17"
styleId="stops">
            <c:forEach items="${SessionHelper.stopsList}" var="stop" >
                <c:set var="stopsContentKey" value="ui.i18n.label.
stops.${stop.text}" />
                <html-el:option value="${stop.value}"
key="${stopsContentKey}" ></html-el:option>
            </c:forEach>
        </html:select>
    </td>
</tr>

```

Listing 6

```

<form name="tripTemplateForm">
    <field property="from" depends="required">
        <arg0 key="ui.text.o_and_d.from_label"/>
    </ field >
    < field property="to" depends="required">
        <arg0 key="ui.text.o_and_d.to_label_uppercase"/>
    </ field >
    < field property="departDay" depends="required">
        <arg0 key="ui.text.trip_template_edit.depart_la-
bel"/>
    </ field >
    < field property="departTime" depends="required">
        <arg0 key="ui.text.trip_template_edit.depart_la-
bel"/>
    </ field >
</form>

```

Listing 7

```

public class LocaleActionServlet extends ActionServlet{
    public static final String DEFAULT_LANGUAGE = "en";
    public static final String DEFAULT_COUNTRY = "us";
    public static final String DEFAULT_SITE = "core";

    public void service( HttpServletRequest request, HttpServletResponse
response ) throws ServletException, IOException{
        String language = DEFAULT_LANGUAGE;
        String country = DEFAULT_COUNTRY;
        String site = DEFAULT_SITE;

```

```

        MessageResources messageResources = (MessageResources) this.
getServletContext().getAttribute( Globals.MESSAGES_KEY );
        String configParam = messageResources.getConfig();
        HttpSession session = request.getSession(true);
        SessionHelper sessionHelper = SessionHelper.getSessionHelper(
session );

```

```

        String servletPath = request.getServletPath();
        servletPath = servletPath.substring( 1 );
        String [] pathValues = servletPath.split( "/" );

```

```

        if( pathValues.length <= 2 ) {
            Locale locale = ( Locale ) session.getAttribute( Globals.
LOCALE_KEY );

```

```

            if( locale == null ) {
                locale = new Locale( language, country, site );
                sessionHelper.setLocale(locale, configParam);
            }

```

```

            super.service( request, response );

```

```

        }

```

```

    else if( pathValues.length == 4 || pathValues.length == 5 ) {
        site = pathValues[0];
        country = pathValues[1];
        language = pathValues[2];

```

```

        Locale locale = new Locale(language, country, site);
        sessionHelper.setLocale(locale, configParam);

```

```

        String resourcePath = "";

```

```

        for( int i = 3; i < pathValues.length; i++ ) {
            resourcePath = resourcePath + "/" + pathValues[i];
        }

```

```

        RequestDispatcher dispatcher = request.getRequestDispatcher(
resourcePath );
        dispatcher.forward( request, response );
    }
    else{
        response.sendError( HttpServletResponse.SC_BAD_REQUEST );
    }
}
}

```

Listing 8

```

public void setLocale( Locale locale, String configParam ){
    //Setting the locale in session such that Struts can use it
    currentSession.setAttribute(Globals.LOCALE_KEY, locale);

```

```

    //Setting the locale in session such that JSTL can use it
    Config.set( currentSession, Config.FMT_LOCALE, locale );

```

```

    String[] resources = configParam.split(",");
    LocalizationContext jstlLocalizationContext = new LocalizationCon-
text( new DynaResourceBundle(locale, resources), locale );
    Config.set( currentSession, Config.FMT_LOCALIZATION_CONTEXT, jstlLo-
calizationContext );
}

```


eclipse) developer's journal

A DIGITAL PUBLICATION

Your One-Stop Guide to the Eclipse Ecosystem

The main purpose of *Eclipse Developer's Journal (EDJ)* is to educate and inform users of Eclipse and developers building plug-ins or Rich Client Platform (RCP) applications. With the help of great columnists, news you can use, and technical articles, *EDJ* is a great information source that you will be able to use to educate yourself on just about anything Eclipse related

Subscribe Today!

**FOR YOUR DIGITAL
PUBLICATION**

(AVAILABLE IN DIGITAL FORMAT ONLY)

6 Issues for \$19.99

Visit our site at www.eclipse.sys-con.com or call 1-888-303-5282 and subscribe today!

OFFER SUBJECT TO CHANGE WITHOUT NOTICE



<http://eclipse.sys-con.com>

SYS-CON.COM

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

Avoiding Middle-Aged Spread for Your Infrastructure

DON'T GET CRUSHED IN THE PIZZA-BOX RUSH



By Peter Holditch

I have been knocking around the computer industry for a while now, and I've noticed some changes in my contemporaries and myself... For one thing, the buttons around the stomach of those old shirts that have eluded capture by my wife are looking a bit more strained than they did in the shirts' heyday.

Another thing: I can't run down the road to the shops without spending five minutes glowing a vivid purple hue. I'm sure that kind of thing used to be much easier. I suppose that the increasing demands on my time – having a young family, more responsibility at work, etc., mean that all that time I used to spend playing badminton and going to the gym is being consumed by other activities, and what spare time I do get is generally used up exercising my right arm by lifting 568 g of brown liquid to my lips...

The other thing that I have noticed is that the same thing has happened to many computer systems I have known over the years. Those once-lithe applications so keenly tuned and lovingly slotted into production are getting a little tired too. Gone are the hours of tuning and tweaking that kept them at their svelte best; now, tired operations folk gaze blankly at a wall of CPU meters creeping ever upward, occasionally brushing off a cobweb or two to ring a bell when the utilization hits 90 percent for the umpteenth time.

The applications have gone the same way as the folk who created them. They too are doing different

workloads than they did in the early days of their lives – as workloads have increased, extra machines have been thrown at the applications in an attempt to keep them useful. Eventually one will be unable to consume any more machine resources due to some architectural constraint or another, at which point the unfortunate application will be taken out back and put out of its misery, and a new project will be spawned to create a new one, more fit for the purpose at hand. Maybe that's what they mean when they say that J2EE is a mature server-side platform...

It is lucky that application servers, with their clustering technology, make it easier to deploy new capacity than it was in the 2-tier client/server days, when the only option was to try to find an even bigger hunk of tin and hope that the application could make use of all of it. Application servers have thereby augmented this more traditional "scale up" option for adding more power to the middle tier with the "scale out" alternative – just rack up another Linux blade, and Bob's your uncle. The flavor-of-the-month solution to this is to scale out with blades, since the acquisition cost of a really big server machine is generally rather high. However, the lifetime cost of blades is also very high – they consume a lot of power – which in turn means they require a lot of cooling, and require a lot of management. "Will you just apply that security patch to the OS please?" is the tip of a pretty unpleasant iceberg if there are a few tens of machines to "just" upgrade.

Rather than continue to bark up this tree, which is feeling more and more wrong, it is worth taking a step back and considering what it is that drives the need for this incredible quantity of machines. The

"Since the advent of Storage Area Networks, when was the last time you worried how much spare disk capacity there was in your machine?"

Author Bio:

Peter Holditch is a senior presales engineer in the UK for Azul Systems. Prior to joining Azul he spent nine years at BEA systems, going from being one of their first Professional Services consultants in Europe and finishing up as a principal presales engineer. He has an R&D background (originally having worked on BEA's Tuxedo product) and his technical interests are in high throughput transaction systems. "Of the pitch" Peter likes to brew beer, build furniture, and undertake other ludicrously ambitious projects – but (generally) not all at the same time!

Contact:

pholditch@azulsystems.com

answer turns out to be – at least partially – the need to overprovision to provide headroom for demand spikes. That order-processing system that manages the sales of Christmas wrapping paper idles along utilizing 5 percent of a UNIX server for most of the year, just so it has 75 percent in its back pocket for the Christmas rush. And so it goes on, across the whole application estate, since each application generally has its own dedicated hardware on which to execute. It would be too difficult a configuration management job to have all of the applications on one machine – imagine if one of them needed an OS upgrade that the rest weren't compatible with. Therefore in an effort to make configuration management manageable, lots of CPU headroom is wasted, complete with the associated wasted power and cooling.

It is this issue that the Azul appliance is designed to address. The appliance is designed to run Virtual Machine-based applications *on behalf of* existing UNIX servers. An application that used to leave the UNIX CPU 15 percent idle under load will leave the same CPU 75 percent idle under the same load, when it is mounting the appliance's compute capacity. Suddenly, there is much more headroom on the UNIX box – maybe you can take the 10 you have already and replace them with just three to handle the same load and still provide for redundancy for failover. The appliances providing this compute capacity are deployed in a pool themselves, which means that they are redundant and highly available too. It also means that extra capacity can be brought online administratively, without making any changes at all to the UNIX application environments; the next time a Java application launches, any additional capacity added will be available to service its needs.

Finally, because the UNIX tier still exists to provide the Configuration Management control that it is very good for, the appliances can be shared among multiple applications, so one pool of appliances can be used for the Christmas wrapping paper application, the Easter Bunny application, and the Halloween gifts application. At any one time, the peaks and troughs will balance out, so the compute pool can be kept much more utilized than any one of the UNIX boxes ever could have been. In one fell swoop, the Network Attached Processing approach provided by the appliance pool has allowed consolidation of applications, and provided the flexibility to provide additional capacity on demand whilst preserving the existing configuration management systems.

This is clearly a recipe for teaching some of the “old dog,” middle-aged applications some new tricks, and thereby preserving the investment made in them for some years to come.

However for the hopeless cases, the appliance approach brings other benefits: new applications can be written to take advantage of a 96GB heap. Without fear of stop-the-world garbage collection pauses stretching into infinity, they can use easy-to-maintain, coarse-grained lock structures – since the appliance provides optimistic Java lock concurrency – and they can be written to take advantage of large numbers of active threads, because even the smallest appliance provides 96 CPU cores, each of which can run a thread in parallel.

All in all, the Network Attached Processing solution provides a powerful way to combat mid-tier bloat, while reducing management costs and providing new opportunities for innovative Java architectures. After all, since the advent of Storage Area Networks, when was the last time you worried how much spare disk capacity there was in your machine? Why shouldn't compute become commoditized in the same way? ●

Subscribe Today!

— INCLUDES —
FREE DIGITAL EDITION!
(WITH PAID SUBSCRIPTION)
GET YOUR ACCESS CODE INSTANTLY!



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

SAVE 50% OFF!

(REGULAR NEWSSTAND PRICE)

Only \$39⁹⁹ ONE YEAR 12 ISSUES

www.ISSJournal.com
or 1-888-303-5282

SYS-CON MEDIA

The World's Leading i-Technology Publisher



By Deepak Batra

Adding Self-Detection and Auto-Optimization to the WebLogic 8.1 Platform

A DIFFERENT APPROACH TO PROBLEM SOLVING AND OPTIMIZATION

Let me start by defining the current problem-detection process in most enterprises. An issue arises in the production environment for a J2EE application running on WebLogic, information is captured via logs, and WebLogic server is rebooted. From this point, application developers and administrators are tasked with analysis of the information and finding the root cause.

If they are skilled and fortunate enough they will have all of the information. However in most situations this isn't the outcome. You rarely find the details needed for root-cause analysis from production failures. You frequently have to reproduce the problem in a test environment to be able to get the proper data to determine "root cause." You get the picture: all of this can require significant time and effort.

After facing these problems in the field for many years as a WebLogic consultant and finding available tools lacking in this area, I decided to do something about it. Arcturus (www.arcturus.com) AutoPilot for WebLogic is the result of that decision.

AutoPilot addresses these issues with a unique approach. AutoPilot is an expert system-based tool with many person years of expert knowledge embedded within. AutoPilot automates extremely tedious tasks such as Proactive Monitoring, Instant Root Cause Analysis, and WebLogic Tuning, WebLogic environment review for best practices. AutoPilot results in significant savings in terms of man hours and hardware/software licenses.

AutoPilot proactively monitors WebLogic using its IntelliCheck Technology for patterns that can cause an outage and warns administrators before they

actually do. AutoPilot doesn't require any instrumentation to discover unusual patterns. AutoPilot uses information already available in WebLogic and has extremely low overhead.

AutoPilot completely automates the process of root-cause analysis. AutoPilot automatically detects WebLogic failure conditions in production and auto-analyzes WebLogic state (thousands of matrices) to generate a root-cause analysis report instantly. Additionally, AutoPilot will generate an e-mail that not only tells you that WebLogic server failed, but that also tells you why.

AutoPilot is an all JAVA and J2EE solution that plugs right into and seamlessly coexists with BEA WebLogic Server and Console (see Figure 1). You can get an evaluation version of AutoPilot at <http://support.arcturustech.com/downloadpage.do>.

The installation process for AutoPilot is straightforward. The AutoPilot installation wizard installs AutoPilot in fewer than 10 minutes. You can obtain more details on AutoPilot installation at <http://support.arcturustech.com/APHelp/installingautopilot.htm>.

Although WebLogic does provide all JMX information, it is not in an easy-to-use format. AutoPilot provides an easy-to-use interface for browsing MBeans information. With AutoPilot explorer, views are available for exploring configuration and run-time MBeans properties. AutoPilot explorers make JMX information accessible directly within WebLogic Console, without causing you to have to browse through many JSP pages to obtain the same.

One of the features that I missed with WebLogic was the lack of persistence capabilities for monitoring data. You do get a nice chart of heap and queue size in WebLogic, but once the information falls off the screen it is gone forever. What that means is if you are not in front of the Console when something bad happens,

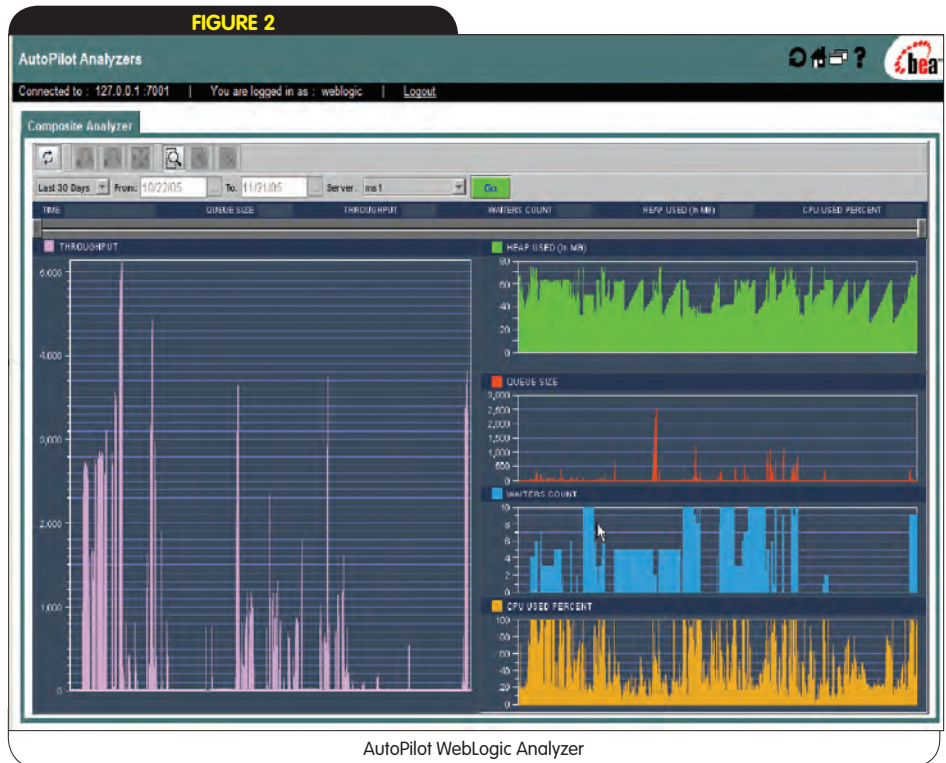
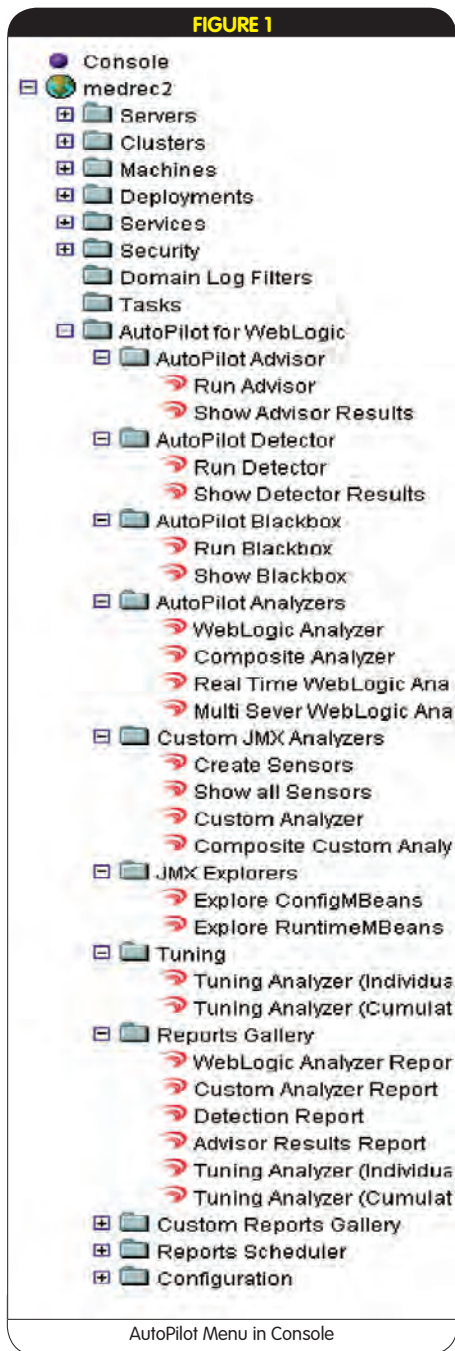
Author Bio:

Deepak Batra has more than 15 years of wide-range experience in the IT industry. He is the founder and CEO of Arcturus Technologies, Inc. Prior to co-founding Arcturus, Deepak worked as an architect with BEA for more than five years. In this capacity he led and managed various strategic initiatives at Fortune 500 companies and helped them craft enterprise solutions that leverage state-of-the-art technologies. Prior to BEA, he provided IT Advisory services to various fortune 500 companies for many years. He has worked closely with customers and understands their requirements first hand.

Contact:

deepak.batra@arcturustech.com

you are out of luck. AutoPilot automatically persists key information about WebLogic state for historical analysis. It provides WebLogic Administrators with extremely easy-to-use data-mining capabilities. For example, you can start with a 30-day view of WebLogic server's performance and drill down to a particular time frame in a matter of a few clicks. AutoPilot Analyzers help you spot problems & trends within the WebLogic Server (see Figure 2).



AutoPilot allows you to create sensors on any JMX property. A sensor is itself a JMX property whose value is persisted every time AutoPilot takes the pulse of WebLogic Server. AutoPilot provides an easy-to-use interface to graphically analyze data accumulated by AutoPilot sensors.

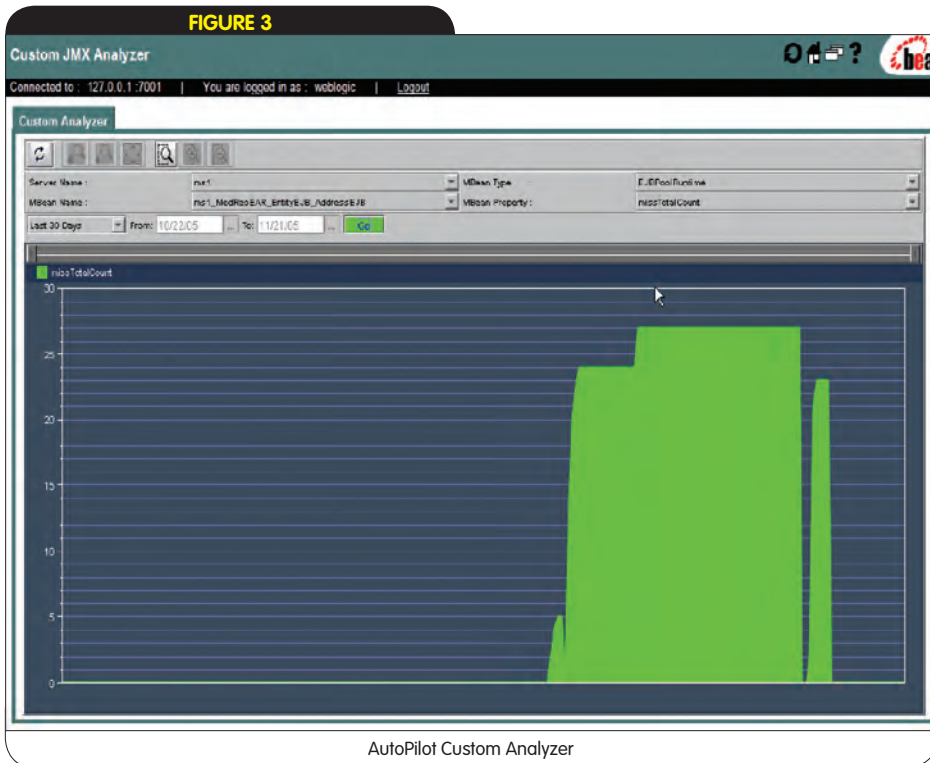
One of the unique and very useful features of AutoPilot is Blackbox. AutoPilot Blackbox (Figure 5) allows you to take a snapshot of the WebLogic Server state at any point in time. Blackbox is the view of the internal state of WebLogic Server and the applications running on it. Blackbox saves the state of the server at the time when it was created. Blackbox data can be analyzed with an easy-to-use explorer view, similar to the one used by AutoPilot Config and Run-Time MBeans Explorers. You can create a Blackbox at any time just with the click of a button. It comes in handy when you are in the middle of an analysis and want to preserve the state.

From my experience, it is much easier to find the root cause of the problem if you see it happening while you are there. Unfortunately however, you can't have an expert sitting in front of WebLogic Console waiting for a problem to occur. AutoPilot solves this problem. A key feature of AutoPilot is the auto-detection

of failure conditions in WebLogic Server. AutoPilot automatically detects when WebLogic Server gets into a bad state. Once such a condition is detected, AutoPilot automatically creates a Blackbox for WebLogic. In most cases once WebLogic gets into this bad state, administrators bounce the faulty instance of WebLogic Server quickly to bring the production environment back to the desired capacity. Hence even if an expert is available to analyze the server and find the cause for the issue, the person doesn't have enough time for troubleshooting. Blackbox provides these experts with the complete state of the server at that instance, even after the server is brought down. AutoPilot just doesn't stop there. It auto analyzes the state of the WebLogic Server to generate a "root cause" report that gets e-mailed to the administrator(s) automatically.

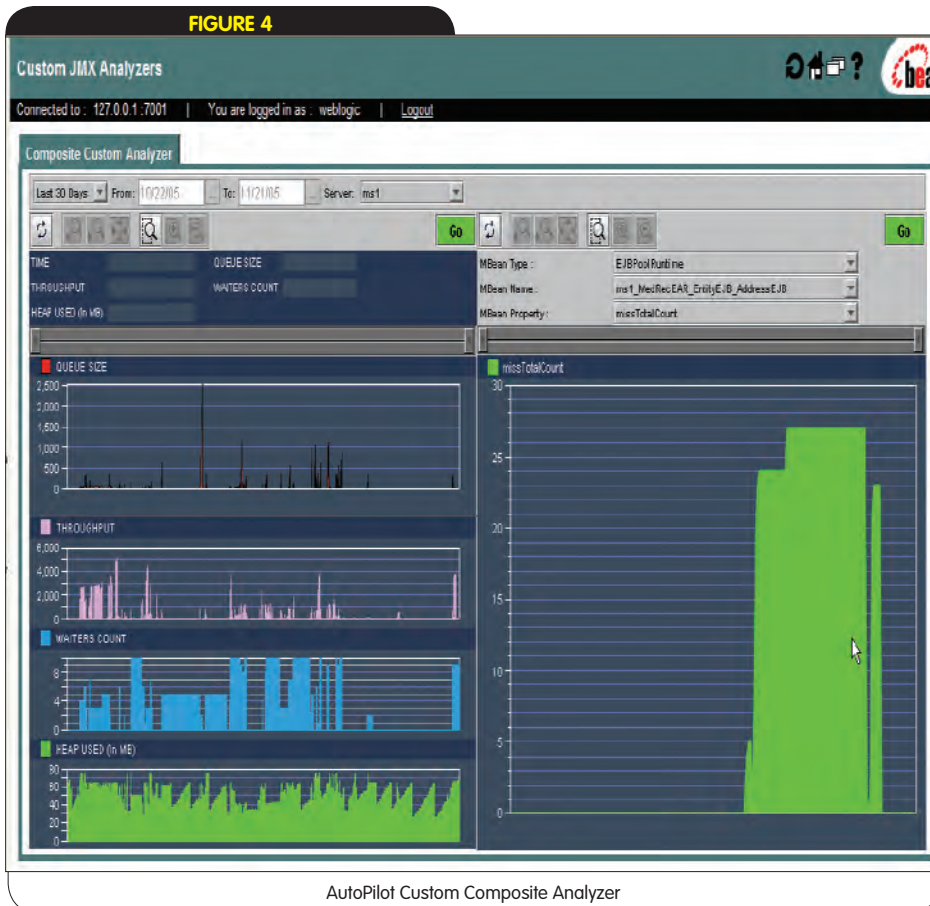
In this way experts get all of the information they need even though the incident may have happened in the middle of the night while they were catching up on their sleep. AutoPilot can analyze thousands of matrices almost instantly, versus an expert who would take significant time to analyze the same. AutoPilot drills down to transaction level in determining the root cause and identifies the culprit

FIGURE 3



AutoPilot Custom Analyzer

FIGURE 4



AutoPilot Custom Composite Analyzer

transaction(s). This is what makes AutoPilot for WebLogic an extremely useful tool.

The AutoPilot Detector that determines the root cause can also be manually executed to analyze the run-time state of WebLogic Servers at any point in time. It is extremely efficient and it's the least intrusive way of finding out the problematic patterns developing in your environment. AutoPilot detector doesn't make any changes to the environment and hence can be safely executed in a production environment.

AutoPilot adds sought-after self-detection capabilities to WebLogic 8.1. Using its IntelliCheck Technology, AutoPilot monitors and analyzes internal WebLogic state to detect conditions that can lead to an outage. Once such a condition is detected, AutoPilot raises alerts for administrators.

As a WebLogic consultant, every customer I visited asked me to review their environment and guide the on where there is room for improvement. It is a time-consuming task to certify that there are no issues with the environment and advise about best practices. With AutoPilot Advisor we have automated this task. AutoPilot Advisor analyzes your whole WebLogic environment and generates a report of recommendations. AutoPilot Advisor tells you where you are not leveraging your WebLogic right and guides you on how to resolve those issues. Advisor also guides by revealing best practices and where in your environment you can benefit from them. This is equivalent to having an expert evaluate your environment and provide feedback on how well you are leveraging your infrastructure. The AutoPilot advantage is that it applies many person years of expert knowledge at the touch of a button and produces results in a matter of minutes.

Many times you have a production outage and you work with BEA Support to find out that there is a patch (CR) available for that. Wouldn't it be good to know of these patches that are applicable to your environment before you actually encounter the issue in production? With AutoPilot you can. Advisor checks for all the applicable patches to your environment and provides you with details on each. It doesn't just blindly give you a list of patches that are available from BEA in newer service packs, it goes and checks for relevance. This way we are able to filter the list of patches to the ones you must have.

Another problem that enterprises are



Visit the *New*
www.SYS-CON.com
 Website Today!

The World's Leading *i*-Technology
 News and Information Source

24/7

FREE NEWSLETTERS

Stay ahead of the *i*-Technology curve with E-mail updates on what's happening in your industry

SYS-CON.TV

Watch video of breaking news, interviews with industry leaders, and how-to tutorials

BLOG-N-PLAY!

Read web logs from the movers and shakers or create your own blog to be read by millions

WEBCAST

Streaming video on today's *i*-Technology news, events, and webinars

EDUCATION

The world's leading online *i*-Technology university

RESEARCH

i-Technology data "and" analysis for business decision-makers

MAGAZINES

View the current issue and past archives of your favorite *i*-Technology journal

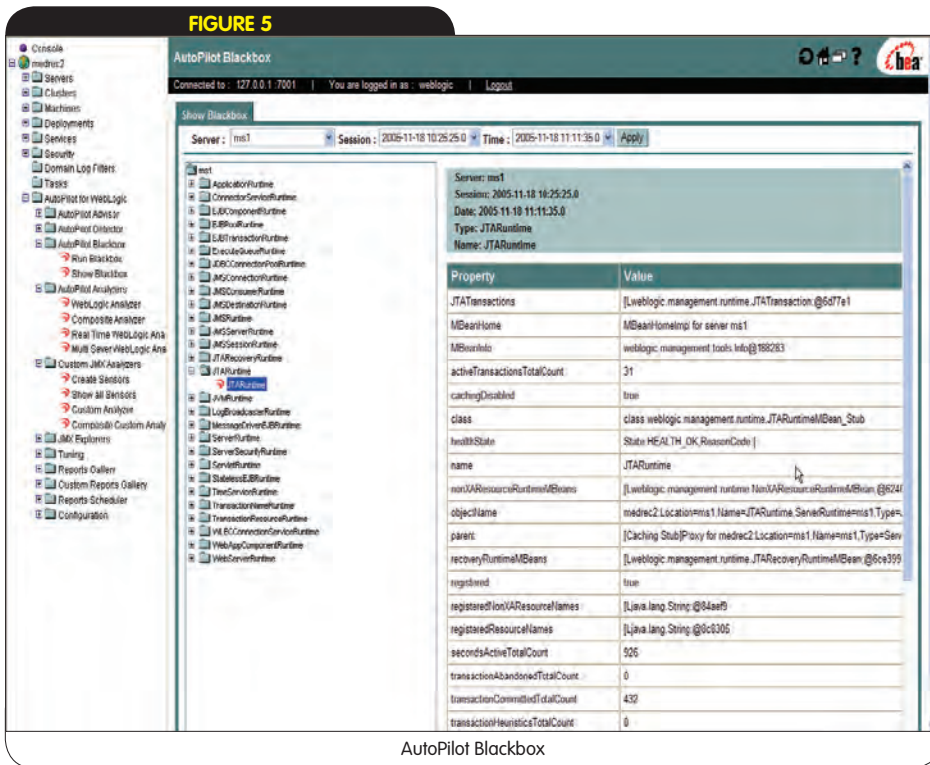
INTERNATIONAL SITES

Get all the news and information happening in other countries worldwide

**JUMP TO THE LEADING
i-TECHNOLOGY WEBSITES:**

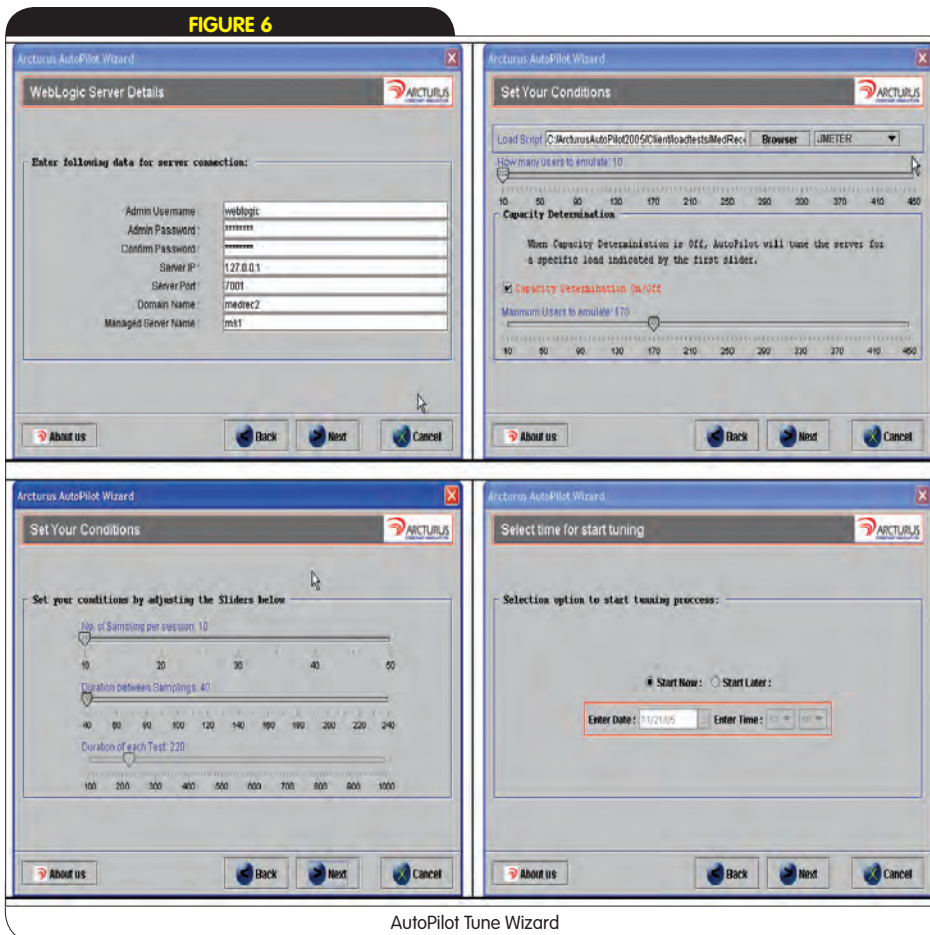
- IT Solutions Guide*
- Information Storage+Security Journal*
- JDJ*
- Web Services Journal*
- .NET Developer's Journal*
- LinuxWorld Magazine*
- Linux Business News*
- Eclipse Developer's Journal*
- MX Developer's Journal*
- ColdFusion Developer's Journal*
- XML Journal*
- Wireless Business & Technology*
- Symbian Developer's Journal*
- WebSphere Journal*
- WLDJ*
- PowerBuilder Developer's Journal*

FIGURE 5



AutoPilot Blackbox

FIGURE 6



AutoPilot Tune Wizard

facing is that extensive time and effort are required to align SOA/J2EE applications and WebLogic Servers for optimum performance. Currently, tuning WebLogic is considered an art and not science. Although more and more people are becoming familiar with it, the tuning process “tune one thing, see the impact, and repeat until balanced configuration is achieved,” still remains an extremely time-consuming and laborious task. Because performance tuning is so complex, many enterprises are running WebLogic with configurations that are not fully optimal. Another factor to keep in mind here is that WebLogic tuning is extremely fragile. Every time you change you application you can throw the whole environment completely off balance, and previously tuned configurations may not be optimum due to the recent change to the application. I compare WebLogic tuning to the analogy of car wheels. Every time you change car tires you need to do a few other things such as wheel alignment, balancing, etc. to get the best performance. Similarly in the case of WebLogic, every time there is a change in the application, WebLogic and the application both need to be realigned to get the optimum performance. During my experience in the field I often got this response: “We did tuning last year, it should still be good, right?” Well if nothing changed in the environment and application then that is true, but with today’s constantly changing business requirements, applications and environments change more often than once a year. With every change you should evaluate the performance impact. It may be that not all changes require change in WebLogic or application configuration, but at the very least every change should be looked at from a performance standpoint.

AutoPilot comes to the rescue for WebLogic tuning and capacity determination. AutoPilot changes WebLogic tuning from an extremely complex task to one that entails just a few clicks. AutoPilot automates the proven WebLogic tuning process using its IntelliTune technology built upon AutoPilot’s inherent knowledge and decision-making capabilities. With AutoPilot, the process of tuning WebLogic remains the same – it’s just that AutoPilot does all of the work of running loads, analyzing performance, making appropriate changes, and restarting WebLogic Server whenever needed. AutoPilot uses a user-provided load-generation script and

A LIMITED TIME SAVINGS OFFER FROM SYS-CON MEDIA

SUBSCRIBE TODAY TO MULTIPLE MAGAZINES

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

RECEIVE YOUR DIGITAL EDITION ACCESS CODE INSTANTLY WITH YOUR PAID SUBSCRIPTIONS



3-Pack

Pick any 3 of our magazines and save up to **\$210⁰⁰**
 Pay only \$99 for a 1 year subscription plus a FREE CD

- 2 Year – \$179.00
- Canada/Mexico – \$189.00
- International – \$199.00

6-Pack

Pick any 6 of our magazines and save up to **\$340⁰⁰**
 Pay only \$199 for a 1 year subscription plus 2 FREE CDs

- 2 Year – \$379.00
- Canada/Mexico – \$399.00
- International – \$449.00

9-Pack

Pick 9 of our magazines and save up to **\$270⁰⁰**
 Pay only \$399 for a 1 year subscription plus 3 FREE CDs

- 2 Year – \$699.00
- Canada/Mexico – \$749.00
- International – \$849.00

CALL TODAY! 888-303-5282

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

<input type="checkbox"/> 3-Pack	<input type="checkbox"/> 1YR	<input type="checkbox"/> 2YR	<input type="checkbox"/> U.S.	<input type="checkbox"/> Can/Mex	<input type="checkbox"/> Intl.
<input type="checkbox"/> 6-Pack	<input type="checkbox"/> 1YR	<input type="checkbox"/> 2YR	<input type="checkbox"/> U.S.	<input type="checkbox"/> Can/Mex	<input type="checkbox"/> Intl.
<input type="checkbox"/> 9-Pack	<input type="checkbox"/> 1YR	<input type="checkbox"/> 2YR	<input type="checkbox"/> U.S.	<input type="checkbox"/> Can/Mex	<input type="checkbox"/> Intl.

<input type="checkbox"/> LinuxWorld Magazine			
U.S. - Two Years (24) Cover: \$143	You Pay: \$79.99 /	Save: \$63 + FREE \$198 CD	
U.S. - One Year (12) Cover: \$72	You Pay: \$39.99 /	Save: \$32	
Can/Mex - Two Years (24) \$168	You Pay: \$119.99 /	Save: \$48 + FREE \$198 CD	
Can/Mex - One Year (12) \$84	You Pay: \$79.99 /	Save: \$4	
Intl' - Two Years (24) \$216	You Pay: \$176 /	Save: \$40 + FREE \$198 CD	
Intl' - One Year (12) \$108	You Pay: \$99.99 /	Save: \$8	

<input type="checkbox"/> JDJ			
U.S. - Two Years (24) Cover: \$144	You Pay: \$99.99 /	Save: \$45 + FREE \$198 CD	
U.S. - One Year (12) Cover: \$72	You Pay: \$69.99 /	Save: \$12	
Can/Mex - Two Years (24) \$168	You Pay: \$119.99 /	Save: \$48 + FREE \$198 CD	
Can/Mex - One Year (12) \$84	You Pay: \$89.99 /	Save: \$40	
Intl' - Two Years (24) \$216	You Pay: \$176 /	Save: \$40 + FREE \$198 CD	
Intl' - One Year (12) \$108	You Pay: \$99.99 /	Save: \$8	

<input type="checkbox"/> 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	
Intl' - Two Years (24) \$216	You Pay: \$170 /	Save: \$46 + FREE \$198 CD	
Intl' - One Year (12) \$108	You Pay: \$99.99 /	Save: \$8	

<input type="checkbox"/> .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	
Intl' - Two Years (24) \$216	You Pay: \$170 /	Save: \$46 + FREE \$198 CD	
Intl' - One Year (12) \$108	You Pay: \$99.99 /	Save: \$8	

<input type="checkbox"/> Information Storage + Security Journal			
U.S. - Two Years (24) Cover: \$143	You Pay: \$49.99 /	Save: \$93 + FREE \$198 CD	
U.S. - One Year (12) Cover: \$72	You Pay: \$39.99 /	Save: \$39	
Can/Mex - Two Years (24) \$168	You Pay: \$79.99 /	Save: \$88 + FREE \$198 CD	
Can/Mex - One Year (12) \$84	You Pay: \$49.99 /	Save: \$34	
Intl' - Two Years (24) \$216	You Pay: \$89.99 /	Save: \$126 + FREE \$198 CD	
Intl' - One Year (12) \$108	You Pay: \$59.99 /	Save: \$48	

<input type="checkbox"/> Wireless Business & Technology			
U.S. - Two Years (12) Cover: \$120	You Pay: \$49.00 /	Save: \$71 + FREE \$198 CD	
U.S. - One Year (6) Cover: \$60	You Pay: \$29.99 /	Save: \$30	
Can/Mex - Two Years (12) \$120	You Pay: \$69.99 /	Save: \$51 + FREE \$198 CD	
Can/Mex - One Year (6) \$60	You Pay: \$49.99 /	Save: \$10	
Intl' - Two Years (12) \$120	You Pay: \$99.99 /	Save: \$20 + FREE \$198 CD	
Intl' - One Year (6) \$72	You Pay: \$69.99 /	Save: \$2	

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.

<input type="checkbox"/> MX Developer's Journal			
U.S. - Two Years (24) Cover: \$143	You Pay: \$49.99 /	Save: \$93 + FREE \$198 CD	
U.S. - One Year (12) Cover: \$72	You Pay: \$39.99 /	Save: \$32	
Can/Mex - Two Years (24) \$168	You Pay: \$79.99 /	Save: \$88 + FREE \$198 CD	
Can/Mex - One Year (12) \$84	You Pay: \$49.99 /	Save: \$34	
Intl' - Two Years (24) \$216	You Pay: \$89.99 /	Save: \$126 + FREE \$198 CD	
Intl' - One Year (12) \$108	You Pay: \$59.99 /	Save: \$48	

<input type="checkbox"/> 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	
Intl' - Two Years (24) \$264	You Pay: \$189 /	Save: \$75 + FREE \$198 CD	
Intl' - One Year (12) \$132	You Pay: \$129.99 /	Save: \$2	

<input type="checkbox"/> WebSphere Journal			
U.S. - Two Years (24) Cover: \$216	You Pay: \$129.00 /	Save: \$87 + FREE \$198 CD	
U.S. - One Year (12) Cover: \$108	You Pay: \$89.99 /	Save: \$18	
Can/Mex - Two Years (24) \$240	You Pay: \$159.99 /	Save: \$80 + FREE \$198 CD	
Can/Mex - One Year (12) \$120	You Pay: \$99.99 /	Save: \$20	
Intl' - Two Years (24) \$264	You Pay: \$189.00 /	Save: \$75	
Intl' - One Year (12) \$132	You Pay: \$129.99 /	Save: \$2	

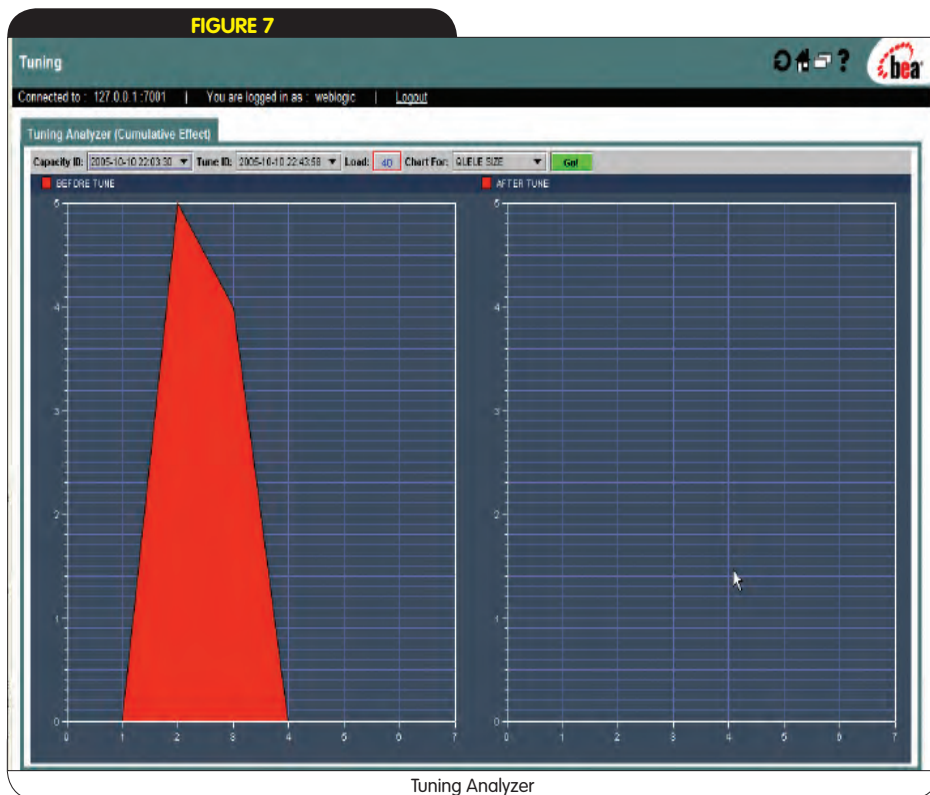
<input type="checkbox"/> PowerBuilder Developer's Journal			
U.S. - Two Years (24) Cover: \$360	You Pay: \$169.99 /	Save: \$190 + FREE \$198 CD	
U.S. - One Year (12) Cover: \$180	You Pay: \$149 /	Save: \$31	
Can/Mex - Two Years (24) \$360	You Pay: \$179.99 /	Save: \$180 + FREE \$198 CD	
Can/Mex - One Year (12) \$180	You Pay: \$169 /	Save: \$11	
Intl' - Two Years (24) \$360	You Pay: \$189.99 /	Save: \$170 + FREE \$198 CD	
Intl' - One Year (12) \$180	You Pay: \$179 /	Save: \$1	

<input type="checkbox"/> WLDJ			
U.S. - Four Years (24) Cover: \$240	You Pay: \$99.99 /	Save: \$140 + FREE \$198 CD	
U.S. - Two Year (12) Cover: \$120	You Pay: \$49.99 /	Save: \$70	
Can/Mex - Four Years (24) \$240	You Pay: \$99.99 /	Save: \$140 + FREE \$198 CD	
Can/Mex - Two Year (12) \$120	You Pay: \$69.99 /	Save: \$50	
Intl' - Four Years (24) \$240	You Pay: \$120 /	Save: \$120 + FREE \$198 CD	
Intl' - Two Year (12) \$120	You Pay: \$79.99 /	Save: \$40	

*WHILE SUPPLIES LAST. OFFER SUBJECT TO CHANGE WITHOUT NOTICE

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





runs various scenarios on the WebLogic application. It analyzes the application and WebLogic behavior during each load. At the end of each load and analysis, AutoPilot makes appropriate adjustments to the right tunables. After adjustments it reruns the load and compares. It follows this process until a perfect balance is achieved. During the tuning process AutoPilot identifies any application bottlenecks. You can start the WebLogic tuning process before you leave for the evening and the next morning you get a perfectly tuned WebLogic. Your involvement in the whole tuning process takes place only during the kickoff. AutoPilot Tune Wizard simplifies the kickoff process to just a few clicks. You can also schedule the kickoff to start tuning at nighttime (or any other time) if needed.

At the end of the tuning, you can view the performance impact using AutoPilot Tuning Analyzers. You get the before and after picture to compare the results of each tuning. Since AutoPilot tunes and detects problem at the WebLogic-engine level, it helps the whole suite of products based on WebLogic Server.

AutoPilot compliments self-tuning capabilities offered by WebLogic 9.0. In my opinion, self-tuning is a step in the right direction, but it will take time to mature. Dynamically changing the settings

in production can cause unforeseen conditions and can result in outages. Until enterprises get comfortable with dynamically changing the settings in production, AutoPilot will help auto-tune applications and WebLogic in preproduction environments. This way you are pushing a well-tuned configuration to production and not doing tuning in production. Even in the future when self-tuning matures and enterprises become comfortable with self-tuning, it would make sense to use self-tuning features only as "traction control" or "stability assist," i.e., go to production with a well-tuned configuration and let self-tuning help in certain conditions not accounted for in preproduction tuning.

In a nutshell, AutoPilot compliments WebLogic 8.1, the most widely used version of WebLogic in production, with well-sought-after capabilities. Key differentiators for AutoPilot are its intelligence that is built upon many person years worth of embedded expert knowledge, the way AutoPilot leverages that to streamline the root-cause analysis and WebLogic tuning process, proactive guidance, and how AutoPilot results in significant reduction in total cost of ownership for the BEA WebLogic Platform. 🍅

THREE REASONS TO
blog-n-play.com

1 Get instantly published to 2 million+ readers per month!

blog-n-play™ is the only **FREE** custom blog address you can own that comes with instant access to the entire i-technology community. Have your blog read alongside the world's leading authorities, makers and shakers of the industry, including well-known and highly respected i-technology writers and editors.

2 Own a most prestigious blog address!

blog-n-play™ gives you the most prestigious blog address. There is no other blog community in the world that offers such a targeted address, and comes with an instant targeted readership.

3 Best blog engine in the world...

blog-n-play™ is powered by **Blog-City™**, the most feature rich and bleeding-edge blog engine in the world, designed by Alan Williamson, the legendary editor of **JDJ**. Alan kept the i-technology community bloggers' demanding needs in mind and integrated your blog page to your favorite magazine's Web site.

 www.TAMI.linuxworld.com
 "Many blogs to choose from"

PICK YOUR MOST PRESTIGIOUS ADDRESS

IT Solutions Guide	MX Dev. Journal
Storage+Security Journal	ColdFusion Dev. Journal
JDJ: Java	XML-Journal
Web Services Journal	Wireless Business & Tech.
.NET Dev. Journal	WebSphere Journal
LinuxWorld Magazine	WLDJ: WebLogic
LinuxBusinessWeek	PowerBuilder Dev. Journal
Eclipse Dev. Journal	

3 MINUTE SETUP

Sign up for your FREE blog Today!

blog-n-play.com™
 i-Technology Blogs Read by Millions *beta*
www.blog-n-play.com

— This site will go beta **February 15, 2005!**

REPRINT IT!

Once
you're in
it...



...reprint it!

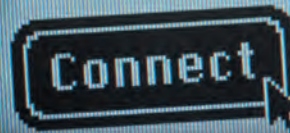
- ColdFusion Developer's Journal
- Java Developer's Journal
- Web Services Developer's Journal
- Wireless Business & Technology
- XML Journal
- PowerBuilder Developer's Journal
- .NET Developer's Journal

Contact Dorothy Gil
201 802-3024
dorothy@sys-con.com

REprints



www.linuxworld.com



Subscribe
Today!

Connect online
for fastest service...
don't miss another
issue of LWM!



SAVE 30%
OFF!

REGULAR ANNUAL COVER PRICE \$71.76

YOU PAY ONLY

\$49⁹⁹
12 ISSUES/YR

WITH SALES TAX WHERE APPLICABLE

LOG ON
TO



The World's Leading i-Technology Publisher



www.linuxworld.com

WLDJ ADVERTISER INDEX

ADVERTISER	URL	PHONE	PAGE
Arcturus	www.arcturus.com	866-769-8166	52
Eclipse	http://eclipse.sys-con.com		37
Hp	www.hp.com	800-752-0900	19
Intersperse	www.intersperse.com	800-340-4553	2
Issj	http://issj.sys-con.com/	888-303-5282	39
Jinfonet	www.jinfonet.com/jp12	301-838-5560	23
Motive	www.motive.com/within1	512-531-2527	29
Netiq	www.netiq.com/solutions/web	408-856-3000	51
Parasoft	www.parasoft.com/	888-305-0041	3
Quest Software	www.quest.com/wldj	949-754-8633	7
Reporting Engines	www.reportingengines.com	888-884-8665	11
Sandcherry	www.sandcherry.com	866-383-4500	15
Tangosol	www.tangosol.com	617-623-5782	13
Wily Technology	www.wilytech.com	888-get-wily	5
Windward	www.windwardreports.com	303-499-2544	27
Wldj	http://weblogic.sys-con.com/		35

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



Design for Production Meets the Application Delivery Process

LESSONS FROM THE WORLD OF MANUFACTURING



By Raman Sud

Author Bio:

Bio: Raman Sud is the vice president of engineering for mValent, developer of the Infrastructure Automation Suite. Sud has 20 years of experience delivering mission-critical software for enterprises and telecommunication service providers leveraging distributed development and building integrated teams in the US and India

rsud@mvalent.com

An anecdote that an engineer shared with me recently reminded me of a long-standing concept in manufacturing, design for production (DFP).

The concept has to do with evaluating how a given operation – production line, supply chain, or an entire factory – is performing. DFP ratings are supposed to help product development teams pinpoint bottlenecks and decide what steps might be needed to raise productivity and profitability. DSP isn't something you hear very often in the application delivery space, but it is interesting to imagine what criteria would provide a holistic assessment of how the application life cycle is working, and help determine what tools or processes might improve the process.

The anecdote concerned a large shipping/logistics company, where the engineer had been involved in the final delivery of new and re-worked WebLogic applications to production. The company had a farm of 600 Linux machines, and his role was to automate new WebLogic deployments as much as possible and troubleshoot any issues that

arose. The automation portion of the job was fairly routine: back up the config.xml, stop the admin server, zero out the file, regenerate a new config.xml for use as a template on a bare bones administrative server install, and finish by restarting the server and configuring all of the components necessary to run the clusters in a given domain. Due to the existence of all those hosts, the need to constantly revise the scripts to automate new WebLogic deployments was no trivial job; however, the main challenges he faced were always on the troubleshooting side.

One day he received a page around 6:30 p.m. and returned to the office to find that one of the company's primary shipping acknowledgement systems had completely crashed. The fact that it happened late in the evening was helpful from a severity standpoint, but going over the entire infrastructure stack line-by-line, hunting for configuration errors, made for a very long evening – and represented a task he had encountered many times previously. What was the issue? One of the engineers working on a major upgrade to the application went in and accidentally modified loads of configurations on the wrong host. He applied the changes to production instead of staging. It's easy to see what happened – the standard way for developers to compare their staging environment



to production is to bring up two administration consoles side-by-side, and scan line-by-line for discrepancies. While reviewing the settings, this developer simply got confused and made his edits in the wrong browser window.

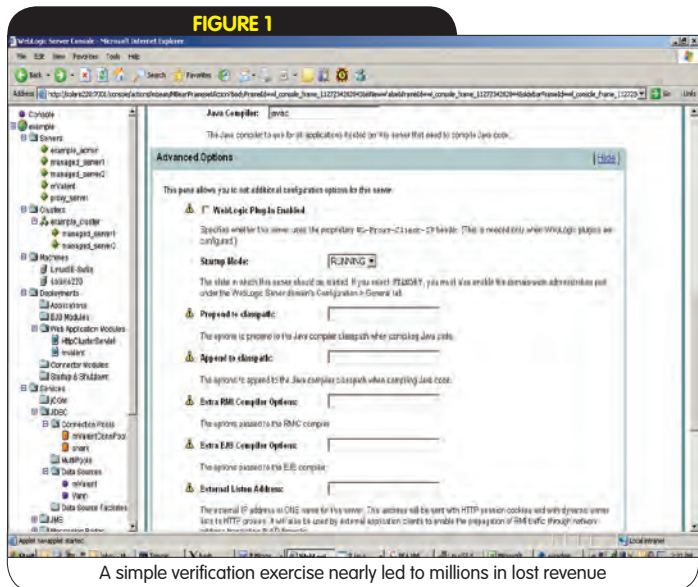
It is an easy mistake to make. The only way to tell the browser windows apart is to look up at the host names of the preproduction and production machines at the top. This is the kind of problem that often is easily dismissed with, “Sorry, I messed up,” until the loss of revenue and business impact results in a painful escalation. The fact – and often the challenge – remains that developers have to access real-time production settings. Even if access were barred to live servers, errors such as this are common, even in IT. Who hasn’t heard of “fat fingering” while manually updating configuration settings in production?

All this leads me to muse about what sort of criteria might be useful to assess the comparative health of an application delivery system, similar to “design for production” standards. My goal is to help

managers assess weaknesses in the overall application delivery chain, and provide areas to target as they go about the work of implementing improvements. Here are five areas for consideration at a high level:

- 1. How good are you at centralizing all of your configuration artifacts? In my own experience, this is an extremely tall order.** The design of a repository is not so difficult, but the actual will to stick to using it on an ongoing basis is next to impossible. Configuration artifacts come from all over the stack – Web servers, application servers, clusters, middleware, LDAP servers, databases. You might be able to assemble everything into a single place for a week or so, but how do you make the metadata usable? For this you need some sort of UI that can expose the settings in a normalized way, thereby allowing the Database administrator, the System administrator, the WebLogic administrator, etc., to make sense of the configuration metadata and feel comfortable using it themselves.
- 2. How well can you gauge what has changed as an application progresses across the life cycle?** This includes detecting out-of-band changes to running application configurations, and monitoring the steady stream of changes that occur along the application’s path to production. By enhancing the ability to compare environments up and down the infrastructure stack – and determining good and bad changes when an application moves from QA out to the staging environment – companies can gain more confidence when it comes time to unleash a new application on the production environment. More often than not, the inability to model the impact of changes before committing them to production leads IT to simply implement the change and “watch for smoke” – which has always struck me as a pretty risky way to run a business.

- 3. The third set of criteria that I think would be valuable in a DFP assessment – and one that consistently comes up among managers of IT infrastructure – is the extent to which standards are defined and enforced around changes to the IT infrastructure stack.** Standards have a general appeal, especially now that change management and reporting have become such hot topics. However enforcing standards around changes to the IT infrastructure stack represents more than just trying to set up a common way of reviewing or reporting on the changes. It requires setting a reliable, consistent process for how to make changes, and ideally, a set of rules that declare which changes are considered valid or invalid. If applied effectively, standards can be enormously useful in helping streamline tasks, as they establish procedures for reusing processes that are known to work. In the long run, standards also lay the foundation for a much-desired state within IT – true automation of time-consuming tasks and processes.
- 4. The fourth DFP yardstick, in my opinion, should be evaluating how well the organization eliminates wasted effort. This is an area that probably has the biggest impact on the bottom line, and seems quite hard to remedy.** I have a hypothesis about why this is so: it’s the “blow up the data center” mentality. For reasons I’ve never been clear about, architects and other senior IT people seem to always conclude the worst about how inefficient business areas need to be addressed. “Our processes are so inefficient, we should just start over” is often the response. It is of course optimistic to plan to solve these problems in one big push, but it just doesn’t seem realistic to me. Instead of ripping everything apart and rebuilding, how about defining a solution that solves 80 percent of the pain, and trying to execute on that as a starting point? One way to begin is to look across silos of responsibility and identify tasks that are constantly being repeated. By addressing these repetitive tasks, managers can increase application quality and throughput, and also lay the groundwork for further savings through automation.
- 5. My final candidate for judging an organization’s DFP state of health has to do with relative consistency.** In a real-world



A simple verification exercise nearly led to millions in lost revenue

environment of hot-fixes, patches, upgrades, and new releases, the IT infrastructure team is typically very hard-pressed to combat configuration “drift.” It is commonly accepted that as a particular server in one area of the staging environment or the data center undergoes a series of consecutive changes, it will naturally drift out of alignment with the proper standard or policy. To make a lasting impact on the overall health of the application delivery process and achieve a high level of consistency and reliability across all stages of the application life cycle – from development, QA, stress/performance testing, to staging and production – the life-cycle environments have to remain in sync and be verifiable within the correct constraints set up by the IT department.

The anecdote related by the engineer that prompted this line of thought wasn’t unusual. It was merely a symptom of an IT infrastructure approach that is significantly handicapped. In order to push higher-quality applications out faster, IT has to expose the underlying configuration settings that power the infrastructure stack in a transparent way, and provide the right degree of access to these same artifacts across teams of stakeholders who are not in a position to share this type of access today. The WebLogic administration console is very well suited to managing

configuration changes to the application server layer, but it cannot single-handedly automate repetitive processes across the entire application life cycle. Handling that burden requires introducing tools and technologies capable of normalizing metadata from a broad range of target systems and allowing individual administrators to manage the data more efficiently.

Once new approaches to standards are implemented and repetitive tasks have been rolled into automated actions, it is reasonable to expect that IT will have a solid foundation from which to easily push out new applications as fast as developers can deliver them. The point is not that mistakes should never happen in a highly automated IT organization; rather, the best way to guard against human error is to put in place a system of checks and balances that can take into account the broad base of specialized, granular system knowledge that exists across the entire application life cycle. In agreeing on such a foundation, it is vitally important to apply the same policies across the entire IT infrastructure, from development to QA to staging and out to production. In this way, the foundation for automation is laid far in advance of the production environment, and the whole application delivery mechanism improves steadily over time. After all, just as in a manufacturing line, the IT infrastructure stack is never better than the sum of its parts, and when one silo of technology gets something wrong, the entire value chain suffers. ●

– continued from page 9

Assured Delivery of Audit Data

In the JMS Topic, the message is persisted again and broadcasted to durable subscribers. We have only one subscriber in the system – the Message Bridge. The latter starts transaction, retrieves message from the JMS Topic, and sends it to the receiver’s distributed Queue. If Message Bridge is down or receiver’s distributed Queue is unavailable, the message stays with the JMS Topic until next attempt. When the message is successfully delivered to the receiver’s distributed Queue, the acknowledgement is sent and message leaves from the JMS Topic persistent storage.

If more than one receiver is permitted to get the message, a new Message Bridge and receiver’s distributed Queue are configured in the system. In this case, the JMS Topic sends the message to as many durable subscribers (Message Bridges) as permitted and configured in the system. Figure 3 does not reflect if a receiver is deployed in a cluster, though it is highly recommended.

Finally, a receiver – Audit Service Provider – deploys a pool of MDB to get the messages from the receiver’s distributed Queue. Each MDB starts its own transaction and Audit Service Provider persists an audit message into the Audit database in the context of this transaction. If transmission to the database

results in an exception, the transaction is rolled back and the message stays with the receiver’s distributed Queue until the next MDB processes it. It is important to notice that MDB uses Bean-Managed Transaction (BMT). The BMT allows Audit Service Provider to roll back the transaction if Provider experiences any problems with, for example, data transformation and throws an application exception.

The discussed system looks relatively complex. One can ask – what is the big deal here? Any Enterprise Service Bus (ESB) can solve this problem and hide all details! Of course, you are right. However, even if an ESB provides guaranteed delivery, what do you do if your organization cannot afford ESB, or it takes too long to buy the product?

Conclusion

We have described two solutions for transmitting audit or any sensitive data from the application to the database with guaranteed delivery. Both solutions are scalable and can effectively work in a distributed environment. Both solutions may be easily converted into services to be used in the service-oriented architecture. The most important aspect of the solutions is that they do not require expensive

products and can be implemented on a regular WebLogic platform.

Acknowledgements

Many thanks go to Bruce Horner and Manmeet Ahluwalia for their significant contributions into the reliability design.

References

- OASIS Web Services Reliable Messaging specification: <http://specs.xmlsoap.org/ws/2005/02/rm/ws-reliablemessaging.pdf>
- OASIS Committees by Category Web Services: www.oasis-open.org/committees/tc_cat.php?cat=ws
- Orchard, D. “Making Sense of Web Services Standards”: http://dev2dev.bea.com/pub/a/2004/01/ws_orchard.html
- BEA AquaLogic Service Bus: www.bea.com/framework.jsp?CNT=index.htm&FP=/content/products/aqualogic/service_bus/
- National Security Institute’s Information, Security Resource: <http://nsi.org/NSIProducts/SECURITYsense/SECURITYsense.html>
- Poulin, M. “How to Deal with Security When Building Application Architecture.” IDJ. SYS-CON Publications, Inc. Vol.10, issue 9: <http://java.sys-con.com/author/poulin.htm> ●



Forget something?

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

The wild blue yonder of operational monitoring and management is extremely unforgiving.

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

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

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



IT CAN THINK
AND DECIDE..
On its own.



keep your J2EE problems...at bay. Get freedom for more in life



**Eliminate endless hours of resolving J2ee problems.
Let AutoPilot™ from Arcturus Technologies, do that for you.**

It's an affordable new product that eliminates worries about WebLogic upkeep and maintenance. AutoPilot™ is so powerful and efficient that not only does it monitor, analyze and optimize your WebLogic for maximum performance, it also advises you about how to cure problems and avoid costly failures. It does this all automatically, giving you the freedom to put back what you have been missing in life

- Automatically optimizes WebLogic server, saving you time and effort.
- Detects problems automatically.
- Gives advice from a dynamic knowledge base.
- Provides WebLogic system state for analysis in the event of a catastrophic failure.
- Optimizes existing J2EE resources.
- Increases the life, effectiveness, and quality of existing resources by reducing or eliminating problem areas.
- Improves resource communication and response time.
- Enhances server capacity to handle more loads and transactions.
- Provides advanced system monitoring.
- Furnishes customizable reports.
- Generates email alerts.

www.arcturustech.com
866/769-8166
sales@arcturustech.com



AUTOPILOT™
AUTO OPTIMIZER