SYS-CON

THE INTERNET'S NEXT GREAT PROTOCOL PG 18

SEPT/OCT 2005 VOL:4 ISSUE:5

wldj™

THE LEADING INDEPENDENT MAGAZINE FOR WEBLOGIC™ PROFESSIONALS

WWW.WLDJ.COM

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

bea 8.1 VALIDATED
BEA WEBLOGIC PLATFORM

Parasoft SOAPtest®

Keep Your Skills Ahead of the Crowd

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

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

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

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

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

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

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

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

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

For Downloads go to www.parasoft.com/soaptest_WSD PARASOFT®

Email: soaptest@parasoft.com Call: 888-305-0041 x1209

Features

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

Benefits

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

Platforms

- Windows 2000/XP
- Linux
- Solaris

Protocol Support

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

Contact Info:

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

www.parasoft.com

Mom, Apple Pie, and the Bottom Line

By James Fenner

Faster than you can say XML, a whole cottage industry has developed to standardize the mechanics of Web services to add to them protocols for things like security and routing and workflow, and even to develop standard XML schemas for business.

For too many, though, Web services are synonymous with service-oriented architecture; many believe that you can't have the latter without the former. While it is cool – this ability for my Visual Basic application to call a Java EJB – it is still just a tactical, marginal improvement in our ability to deliver systems. The real strategy – the big deal that is service orientation – is the systematic eradication of duplication in our systems. For many, this duplication of code and effort is reaching pandemic proportions.

Building enterprise systems is such a daunting task that we usually don't do it. Smaller, departmental solutions are easier, both politically and technically. These stove pipes are usually built soup-to-nuts with their own presentation, business, and data layers. We integrate them when necessary with EAI technology, an enterprise service bus, or with using Web services. We build up an army of point-to-point integrations and never really develop a concept of the enterprise. When it all works we are rewarded and move on. However as good as they might appear, they tend to duplicate each other with impunity. Too often I see things of the enterprise such as customers, products, and agreements built separately, on different platforms and – perhaps the scariest part – even with different rules. It is a level of technical debt that invariably bites you back.

I worked a while ago with a client that had a serious problem because of this type of technical debt. In the face of mounting competitive pressure they needed to drastically change their business model. The very systems that had helped build this successful corporate empire were quickly becoming a burden because they were inflexible and built to enable only the current way of doing business. Adding features was becoming impossibly expensive and their data was out of control. One of their directors estimated that just to adjust their prices for inflation meant that 22 different databases and files needed to be synchronously changed.

Their transformation required service-oriented architecture, but not glitzy things like Web services. It began with a carefully crafted business object model – a finite set of *things* of their business – that could be built as enterprise-wide reusable components for accessing and storing data. In front of these domain components they planned reusable services to handle enterprise processes like registrations, account management, and billing. These could be shared, extended, and localized by their organizations in different countries. Behind, they planned to implement an EII program to rationalize and modernize their data storage. The plan represents years of incremental change to their systems, but it forms a common vocabulary and a flexible infrastructure with which all of their future systems can be built.

– continued on page 21

Author Bio:
Jim is a senior systems architect at CSC Consulting. He specializes in architecture and delivery of business systems and more recently in forensic software analysis and remediation.

Contact:
jfenner@gmail.com

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

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

Run-Time Management of WebLogic Messaging Services

By John-Axel Stråhlman

REINFORCE YOUR ENTERPRISE-MESSAGING INFRASTRUCTURE

As Internet services have evolved and gradually become more and more distributed in nature, enterprise messaging has grown into one of the most important parts of Web application infrastructure. Applications can transfer an enormous amount of messages in a short amount of time, and the data being transferred is often very essential to the underlying business processes.

However important the data might be, many enterprise applications aren't fully equipped to recover from common messaging problems. Sure, messages might be persisted to a data store, but systems often lack reliable and adequately flexible mechanisms for monitoring and administration – something as principal as proper handling of failed messages might be overlooked.

In this article I will discuss WebLogic messaging and the Java Message Service (JMS) from an administrative point of view – we will take a look at how to give business-critical messages the attention they deserve and how to implement management tools for ensuring a smooth message flow for your application.

The Enterprise Application Middleman

Enterprise messaging is all about the exchange of data between disparate systems, a form of intercomputer communication that invariably requires special measures for successful operation. Process-to-process communication over networks without some form of intermediary would be extremely difficult to maintain, so we use message-oriented-middleware systems (MOMs), such as WebLogic JMS, to offload the responsibilities of guaranteed delivery, message notifications, and all other complicated inherent issues. Think of MOMs as the postal service of enterprise applications.

WebLogic JMS is a highly reliable service and if it acknowledges a sent message, you can indeed be sure that the message has been received. That is, however, where the guarantees end; the client application (message producer) has no way of knowing if the receiving application (message consumer) has successfully processed the message, not unless you implement your own acknowledgement system.

Problems, What Problems?

Whenever you have two computers communicating with each other, sooner or later something is bound to go wrong. Unavailable subsystems, network failures, message overflows, and deadlocks are just a few common predicaments that you might stumble into. Instead of trying to make sure that nothing ever goes wrong, be sure to prepare for situations when something does go wrong.

The corrective actions for each type of processing failure need to be evaluated separately. Some problems (such as busy subsystems) are easily fixed by automatic retries, while others require manual intervention. Many services also don't support any form of reprocessing or error handling whatsoever, which makes it the responsibility of the client application to recover from failures.

Let's assume that your business model obligates you to take every effort to process a message before giving up and sending a negative acknowledgement back to the client application. If that were the case, then you should probably retry processing automatically a couple

Author Bio:
John-Axel Stråhlman is the founder and CEO of Sanda Interactive Ltd (www.stc-interactive.com), a software consulting company based in Espoo, Finland. He is a distributed systems specialist and has been working as a consultant for his clients' projects for more than five years.

Contact:
john-axel.strahlman@stc-interactive.com

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

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

GROUP PUBLISHER
Jeremy Geelan jeremy@sys-con.com

ADVERTISING

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

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

Advertising Director
Robyn Forma robyn@sys-con.com

Advertising Manager
Megan Mussa megan@sys-con.com

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

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

SYS-CON EVENTS

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

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

CUSTOMER RELATIONS

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

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

SYS-CON.COM

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

Web Designers
Stephen Kilmurray stephen@sys-con.com
Percy Yip percy@sys-con.com
Vincent Santaiti vincent@sys-con.com

Online Editor
Roger Strukhoff roger@sys-con.com

ACCOUNTING

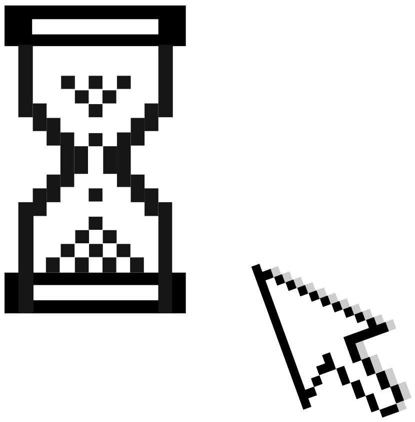
Financial Analyst
Joan LaRose joan@sys-con.com

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 S&H)



World Wide Wait.

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

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

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

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

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

Application Management | Database Management | Windows Management

QUEST SOFTWARE®

of times, and when that doesn't help, someone should be able to manually examine the failing message. Let's take a look at a common way to achieve this in practice.

Message Management, Take One

A very robust message-administration technique is to redirect all failing messages to an error queue and have a consumer save the embodied data and properties to a database. Later, the messages can be investigated by an administrator and optionally re-created and resent. Using a database to store messages gives you very high data integrity and convenience. Here are the main steps required for the implementation:

1. Configure a persistent error queue for your destinations in your config.xml file. Error queues are also known as dead message queues and are regular JMS destinations where failing messages will be sent automatically. Messages are directed to the error queue by adding the ErrorDestination parameter to your active destinations.
2. Configure a consumer for your error queue and make it read the stored data and properties of failed messages by using the accessor methods of the javax.jms.Message class.
3. Save the recovered data together with the name of the original destination of the message to a database table. The original destination can be obtained by the getJMSDestination() method of the Message class.
4. Write methods that read the data from your database, display it, and give the option to resend by re-creating messages from the saved data and sending them to the original destination.

Having done all that, you have a very flexible system for managing messaging problems. The implementation requires quite a lot of work, but is fairly straightforward if you are familiar with JMS and know the basics. This is a very good and often-used solution, but there is another approach that is simpler to implement, doesn't require using a database, and gives you the added benefit of allowing you to peek into the content of any queue at run time.

QueueBrowser for Queue Browsing

The JMS API includes tools that enable full access to queued messages. The tools I'm talking about are the javax.jms.QueueBrowser interface and message selectors. The QueueBrowser is an interface that can retrieve messages from a given queue without consuming them. It can return all messages or, with the help of message selectors, only a subset of the messages.

Our alternative queue-management approach is to use passive error queues for storing failed messages and to use the QueueBrowser for listing them. Single messages can then be selected and resent, copied, or deleted.

Here are the step-by-step instructions for developing a simple queue-management system that handles failed messages:

1. Configure a persistent error queue for your destinations in your config.xml file. You don't need to configure or write any consumers for the error queue.
2. Use the QueueBrowser interface to display all of the messages in your error queue. Browsing through queued messages is very simple and only requires a couple lines of code (see Listing 1).
3. When the user selects a message, you can retrieve it by using its message ID as a criterion for the message selector parameter (see Listing 2). After retrieving it, you can easily redeliver it to its original destination or even to some other passive queue for safekeeping. You can also delete the message by using a QueueReceiver with the same message selector parameter (see Listing 3). You can find a detailed introduction to JMS message selectors in the Sun J2EE API documentation of the javax.jms.Message interface.

In this approach, the application container takes care of persisting the messages for you, which saves you from doing some redundant work. You also avoid a lot of tedious data conversion because you are working with JMS Message objects the whole time.

To ensure that everything works optimally, remember to configure appropriate redelivery limits and redelivery delays in your config.xml file by using the RedeliveryLimit and RedeliveryDelayOverride parameters, respectively.

In addition to developing a message-management system, you should also consider building suitable monitoring tools that keep an eye on your destinations for you. The WebLogic console gives you most of the statistics you need, but there are many good reasons to make that data available from your own management system as well – you have more control that way and access to the console of a production system is usually restricted.

Reading server statistics is remarkably straightforward through the Java Management Extensions (JMX) framework. By using JMX, you will always know what's going on inside your server, and JMS is just one of the many services that are easily monitored with it.

Using JMX for monitoring JMS destinations

JMX is fully supported by the WebLogic Server, which means that almost all WebLogic functionality can be monitored through it. If you haven't done so yet, print a listing of all the MBeans (JMX objects representing managed resources) residing in your server and you will be amazed by the wealth of information available to you. It is a good idea to have a separate process polling all of the most critical MBeans and comparing the values against predefined warning levels.

For retrieving JMS statistics, you first need to acquire JMS MBeans such as the weblogic.management.configuration.JMSQueueMBean and the weblogic.management.runtime.JMSDestinationRuntimeMBean for the destination you want to monitor. These two classes contain several accessor methods for looking up JMS statistics from the server. For instance, you can get the total number of bytes received by a destination or current number of bytes queued, just to name a few things. See the WebLogic documentation for details on how to acquire and use MBeans.

Conclusion

Enterprise messaging has become ubiquitous; whatever kind of service you are building, it is very likely that before long, you will need to share data between other services or simply between processes within your cluster. JMS is the de facto industry standard for the task, and if you still aren't convinced of all its benefits, take a look at my previous article in the Jan/Feb edition of WLDJ entitled "Distributing Tasks in a Clustered Environment Using JMS," and you will see that JMS is about much more than asynchronous data transfer. It is available on the WLDJ Web site under the Archives section (Vol: 4 Iss: 1).

– continued on page 50

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

Managing WebLogic Portal Content Management

AN IMPORT/EXPORT TOOL

By John Graves

One of the nice features of the WebLogic Portal 8.1 release is a fairly extensive content management system. It does not, nor was it ever intended to, compete with the large content management vendors on the market today, but for many applications it works quite well. There are some problems however with using the content management system. This article will address several of them and provide details of a dev2dev tool that will help get around them.

Problem #1: Migration

When content is created outside of a production system, there is no easy way of moving this content into production. In many cases production systems have very strict rules on changing content and ad hoc changes are not allowed, therefore migration is needed.

Problem #2: Backup/Restore

If the default repository is used, all data in the content management system is stored in a database. There are many ways to backup and restore this data using standard database tools, but a good knowledge of the database schema is needed to allow for a proper restore. Often the entire database is backed up, which only allows for a restore to a particular point in time.

Problem #3: Data Type Changes

Data types defined in the content management system cannot be changed if there is any content associated with that type. Therefore, all data must be deleted before a change can be made. This can be frustrating when a lot of data has been created and a simple property change is needed.

Problem #4: Bulk Changes

Sometimes bulk changes are needed in a content management system. The WebLogic Portal Admin User Interface is nice for maintenance of the content and for small changes, but for large ones it can be difficult to use.

Most of these issues are addressed today by fancy SQL scripts written by very advanced database gurus, or that require a large amount of manual work that leads to high maintenance costs.

After facing this issue while building a complex content management–based portal, I decided to build something to help. I developed a Content Management Import/Export tool. I shared this with several of my colleagues and soon discovered this was something that was needed in the public domain. You can find the tool, including source, on dev2dev: http://cmimportexport.projects.dev2dev.bea.com.

I will cover the operation of this tool and how it can be used to address the aforementioned issues.

Installation

You can find an installation guide to help with this process here: https://cmimportexport.projects.dev2dev.bea.com/files/documents/261/192/InstallationGuide.doc. For the purposes of this article, it is assumed that the CM Import/Export portal was installed; however, the most common installation will be the portlet within an existing administration portal.

To install the CM Import/Export portal, simply deploy the appropriate EAR file from here: https://cmimportexport.projects.dev2dev.bea.com/servlets/ProjectDocumentList?folderID=96&expandFolder=96&folderID=96. Once this is done,

Author Bio:
John Graves has been with BEA for more than seven years and is currently working as an internal trainer for Education Services specializing in WebLogic Portal. He spent the first five years of his time with BEA in the Advanced Service Group doing full life-cycle project work..

Contact:
john.graves@bea.com

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.

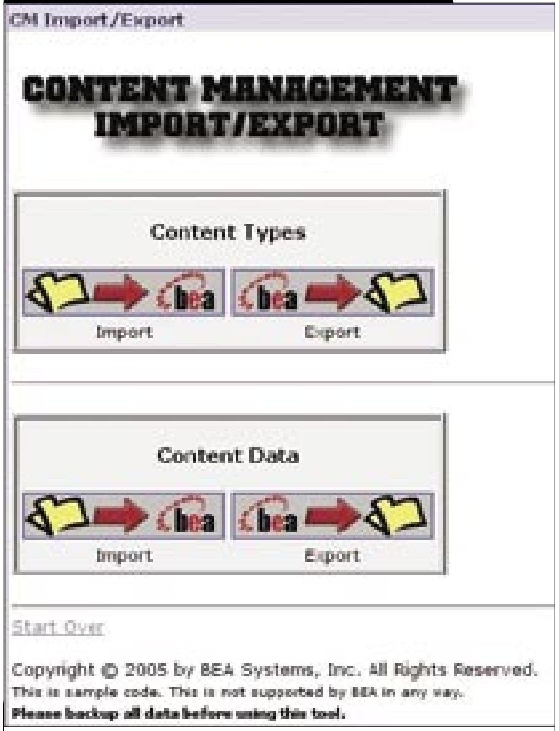
Try before you buy.
Download a free evaluation license at

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.

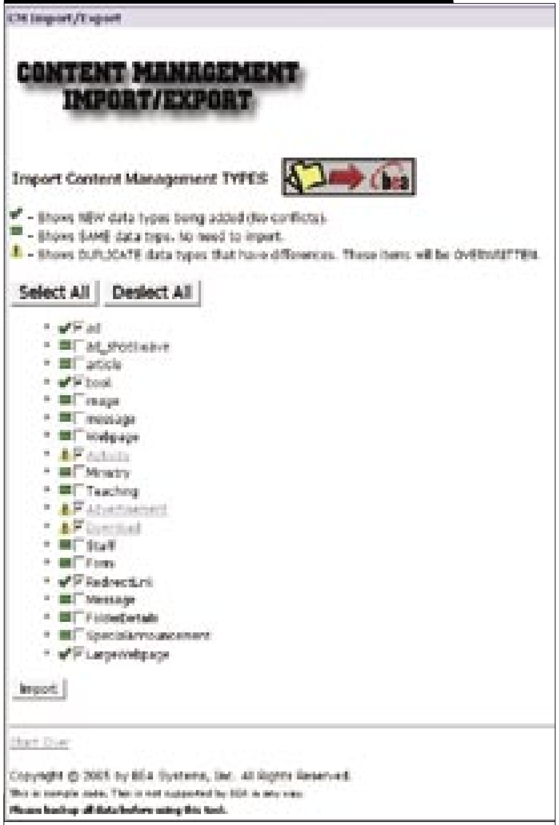
FIGURE 1



Copyright © 2005 by BEA Systems, Inc. All Rights Reserved.
This is sample code. This is not supported by BEA in any way.
Please backup all data before using this tool.

Main screen

FIGURE 2



✓ - Shows NEW data types being added (No conflicts).
▬ - Shows SAME data types. No need to import.
⚠ - Shows DUPLICATE data types that have differences. These items will be OVERWRITTEN.

Copyright © 2005 by BEA Systems, Inc. All Rights Reserved.
This is sample code. This is not supported by BEA in any way.
Please backup all data before using this tool.

Import types

the tool can be accessed by going to the following URL: http://host:port/cmPortal/cmImportExport.portal. Simply replace the host and port that is appropriate for your installation.

Use

After logging into the portal, you are presented with four options (see Figure 1). Two options deal with content types and the other two deal with content data. The tool assumes you have some understanding of WLP content management and the fact that data types must be defined before content data can be added. However, the tool does not force you to do things in a particular order. Therefore, it is possible to try to import data with no type definitions. This, of course, would result in errors.

Content Types Export

Let's start with content types. If you have an existing repository with type definitions, you can export them to an XML file using the Content Types Export button. Once this is selected, you are able to select which repository to use for the export. When the View Export Tree button is pressed, you will be presented with a list of content types and checkboxes. All items that are selected will be exported to the XML file.

The XML file will be created in the server's temp directory. The final screen will show you the full path to the file.

Content Type Schema

There is no official schema definition for WLP content management types. So, I created one (cmTypes.xsd). The tool uses this schema to create XML beans for the import and export operations. The schema is a simple flat structure that supports all data types and properties except for the "explicit" flag (see Listing 1). The WLP content management APIs do not expose this property. Consequently, if you have types with the explicit flag set, this information will not be exported and the corresponding content data will not be exported. Since this is a user-defined database table, typically a separate process is used to maintain this information anyway.

Note: A future release of WLP may adopt portions of the JSR-170 standard for content management import/export. If this is the case, I will adopt this standard to make it easier to move content data from one version of WLP to another.

Content Types Import

The import operations allow you to upload a local XML file into an existing content management system. The import will look for differences between the XML file and the existing content management types. If there are any conflicts, they will be displayed BEFORE the import (see Figure 2). This allows you to intelligently select which content types should be imported into the new system. Note that in some cases, changes to data types that have associated content data may cause a failure. The existing data may need to be exported, deleted, and then reimported. If the data type structure changes, then the exported XML file may need to be changed to reflect the new structure. There are many good XML manipulation tools available that can do this.

Content Data Export

Content data presents more complexity. The structure can be hierarchical, can contain many combinations of different data types, and can have multiple values. Therefore, the JavaScript tree presented during the export can be very extensive. The interface allows you to select and deselect entire branches or individual items. Only the selected branches will be included in the exported XML file, so you could modify the root node in the XML file to reposition the entire set of content to another location in the tree. This would be one way to move or copy an entire branch of a content management system.

Content Data Schema

Once again, there is no official schema definition for the content data, and so I had to create one (cmContent.xsd). There are two types of nodes within WLP content management, content and hierarchy. You may notice that the schema only

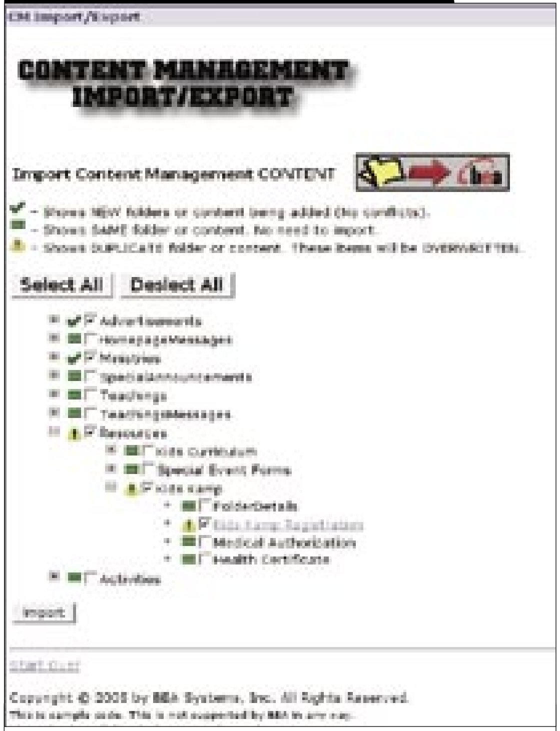
Want to spend more time developing software and less time supporting it? Spend some time discovering hp OpenView—a suite of software management tools that enable you to build manageability right into the applications and Web services you're designing.

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

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

hp
invent

FIGURE 3


CM Import/Export

CONTENT MANAGEMENT IMPORT/EXPORT

Import Content Management CONTENT

✓ - Shows NEW folders or content being added (No conflicts).
🟩 - Shows SAME folder or content. No need to import.
⚠ - Shows DUPLICATE folder or content. These items will be OVERWRITTEN.

Select All | Deselect All

- ✓ Advertisements
- 🟩 HomepageMessages
- ✓ Ministries
- 🟩 SpecialAnnouncements
- 🟩 Teachings
- 🟩 TeachingsMessages
- ⚠ Resources
  - 🟩 Kids Curriculum
  - 🟩 Special Event Forms
  - ⚠ Kids Camp
    - 🟩 FolderDetails
    - ⚠ Kids Camp Registration
    - 🟩 Medical Authorization
    - 🟩 Health Certificate
- 🟩 Activities

Import

Start Over

Copyright © 2005 by BEA Systems, Inc. All Rights Reserved.
This is sample code. This is not supported by BEA in any way.

Import content

differentiates between these two based on using a node attribute (see Listing 2). This is because these two types are actually identical. That's right, you can have content on a hierarchy node. Most users of WLP content management don't realize this feature exists. In the WLP Portal Admin tool, you can create a hierarchy node and then define a data type and content for that node.

Content Data Import

The content data import is the most interesting. There are many situations to consider when importing data into an existing content management system. When there are data conflicts, it is important to identify them and allow the operator to make a logical choice between importing and ignoring certain content. When using the import tool, items that are in conflict are shown with an exclamation point (see Figure 3) and details can be displayed showing what properties are different (see Figure 4). The user can then decide whether or not to import this particular node. It is important to note that the tool uses content hierarchy and name for matching, not ID numbers. If a node is renamed, no conflict will be seen. IDs were not used because it is very possible that ID numbers will be different from one system to another, but the hierarchy and names would be the same.

Solving the Initial Problems

The overall purpose of this article is to show how this tool can be used to solve the initial problems.

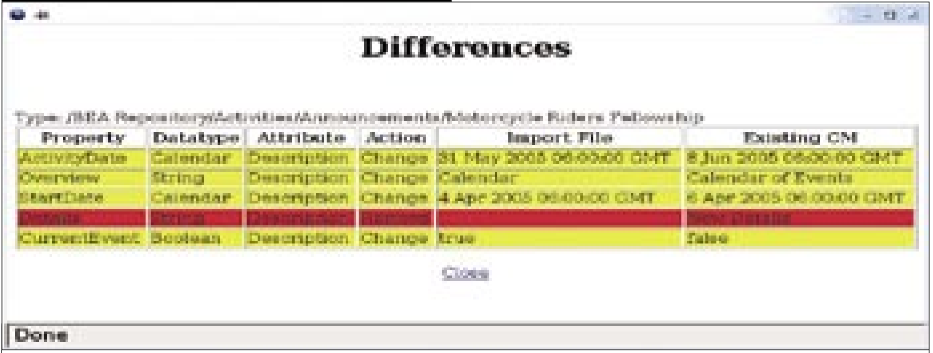
Problem #1: Migration

This was the main purpose for developing this tool. Content types and data can be migrated from one system to another through XML files. It is important to note that the two systems do not need to be accessed at the same time. The exported XML file can be created and imported at a later date (deployment).

Problem #2: Backup/Restore

Migration of data is not the only use for such a tool. You can easily take snapshots of a production system for backup and recovery purposes. The graphical tool or automated script can be used to perform the backup. The XML file name contains the date to make it easy to do periodic backups. Recovery is very easy because the latest backup files can be used and specific data can be included or excluded from the recovery process.

FIGURE 4


Differences

Type: /BEA Repository/Activities/Announcements/Motorcycle Riders Fellowship

Property	Datatype	Attribute	Action	Import File	Existing CM
ActivityDate	Calendar	Description	Change	31 May 2005 06:00:00 GMT	8 Jun 2005 06:00:00 GMT
Overview	String	Description	Change	Calendar	Calendar of Events
StartDate	Calendar	Description	Change	4 Apr 2005 06:00:00 GMT	6 Apr 2005 06:00:00 GMT
Details	String	Description	Remove		New Details
CurrentEvent	Boolean	Description	Change	true	false

Close

Done

Import content differences

Problem #3: Datatype Changes

Solving this problem is a bit more difficult. Data can be exported to an XML file and then it can be deleted from the repository, which allows the data type to be modified. However, it may be necessary to modify the XML file to reflect the datatype changes. There are many good XML editing tools available to do this, or a simple Perl script can be written. The import tool is very forgiving of datatype changes and so it may not be necessary to change the XML file at all. The import will always attempt to add as much data as possible. For example, if a particular property is deleted from the content management system, then the import will simply ignore this data in the XML file and add the rest of the properties. If a property is added, the import will leave this new property blank.

Problem #4: Bulk Changes

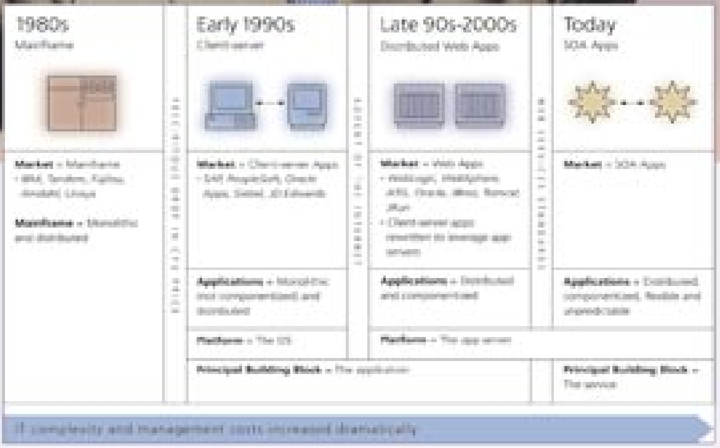
WebLogic Portal provides a mechanism for doing bulk loading, however, it is sometimes more convenient to do this by creating an XML file. For example, for testing purposes, I created a content management type that reflected properties of a file: name, size, permissions. I then wrote a simple Perl script that scanned my filesystem and created a matching XML hierarchy with the file properties. After I imported this into my content management system, I had a full representation of every file on my computer.

Conclusion

The content management system in 8.1 is very powerful and usable, but lacks the important content management migration tools. In WebLogic Portal 9.0, many of these features will be added, but until then, this tool can help make building enterprise-quality applications on top of WebLogic Portal 8.1's content management systems much easier to manage.

PAID EDITORIAL

Intersperse Manages the Risk in SOA Deployments


1980s
Mainframe
Market – Mainframe
• IBM, Tandem, Fujitsu, Amdahl, Unisys
Mainframe – Monolithic and distributed
Early 1990s
Client-server
Market – Client-server Apps
• SAP, PeopleSoft, Oracle Apps, Siebel, JD Edwards
Applications – Monolithic (not componentized) and distributed
Platform – The OS
Principal Building Block – The application
Late 90s-2000s
Distributed Web Apps
Market – Web Apps
• WebLogic, WebSphere, ATG, Oracle, iPlanet, Tomcat, JRun
• Client-server apps rewritten to leverage app servers
Applications – Distributed and componentized
Platform – The app server
Today
SOA Apps
Market – SOA Apps
Applications – Distributed, componentized, flexible and unpredictable
Principal Building Block – The service
IT complexity and management costs increased dramatically

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

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

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

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

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

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

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

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

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

intersperse.

Listing 1

```
<?xml version="1.0" encoding="UTF-8"?>
<cmt:cmTypes xmlns:cmt="http://www.bea.com/CMTypes">
  <cmt:cmDatatype name="ad_shockwave">
        <cmt:cmProperty name="content">
        <cmt:datatype>BINARY</cmt:datatype>
        <cmt:description>Content binary file</cmt:description>
        <cmt:isMandatory>false</cmt:isMandatory>
        <cmt:isReadOnly>false</cmt:isReadOnly>
        <cmt:isPrimary>true</cmt:isPrimary>
        <cmt:isMultiValued>false</cmt:isMultiValued>
        <cmt:isRestricted>false</cmt:isRestricted>
        <cmt:cmBinaryValue default="true"/>
        </cmt:cmProperty>
        .
        . // All properties here
        .
        <cmt:cmProperty name="adWinTarget">
        <cmt:datatype>STRING</cmt:datatype>
        <cmt:description>If set, render will be in popup window with this
name</cmt:description>
        <cmt:isMandatory>false</cmt:isMandatory>
        <cmt:isReadOnly>false</cmt:isReadOnly>
        <cmt:isPrimary>false</cmt:isPrimary>
        <cmt:isMultiValued>false</cmt:isMultiValued>
        <cmt:isRestricted>false</cmt:isRestricted>
        </cmt:cmProperty>
        </cmt:cmDatatype>
        .
        . // All datatypes here
        .
</cmt:cmTypes>
```

Listing 2

```
<cmc:cmContent xmlns:cmc="http://www.bea.com/CMContent">
  <cmc:cmNode name="Activities" nodeType="Hierarchy">
        <cmc:createdBy>weblogic</cmc:createdBy>
        <cmc:creationDate>3 May 2005 21:10:57 GMT</cmc:creationDate>
        <cmc:modifiedBy>weblogic</cmc:modifiedBy>
        <cmc:modifiedDate>3 May 2005 21:10:57 GMT</cmc:modifiedDate>
        <cmc:cmNode name="Bookstore" nodeType="Hierarchy">
        <cmc:createdBy>weblogic</cmc:createdBy>
        <cmc:creationDate>3 May 2005 21:11:03 GMT</cmc:creationDate>
        <cmc:modifiedBy>weblogic</cmc:modifiedBy>
        <cmc:modifiedDate>3 May 2005 21:11:03 GMT</cmc:modifiedDate>
        <cmc:cmNode name="Gene Wards" nodeType="Content">
        <cmc:objectClass>Staff</cmc:objectClass>
        <cmc:createdBy>weblogic</cmc:createdBy>
        <cmc:creationDate>3 May 2005 21:11:03 GMT</cmc:creationDate>
        <cmc:modifiedBy>weblogic</cmc:modifiedBy>
        <cmc:modifiedDate>3 May 2005 21:11:04 GMT</cmc:modifiedDate>
        <cmc:cmContentProperty name="EMail">
        <cmc:datatype>STRING</cmc:datatype>
        <cmc:value>gene@home.org</cmc:value>
        </cmc:cmContentProperty>
        <cmc:cmContentProperty name="Title">
        <cmc:datatype>STRING</cmc:datatype>
        <cmc:value>Bookstore Assistant</cmc:value>
        </cmc:cmContentProperty>
        <cmc:cmContentProperty name="DisplayOrder">
        <cmc:datatype>LONG</cmc:datatype>
        <cmc:value>20</cmc:value>
        </cmc:cmContentProperty>
        <cmc:cmContentProperty name="Name">
        <cmc:datatype>STRING</cmc:datatype>
        <cmc:value>Gene Wards</cmc:value>
        </cmc:cmContentProperty>
        <cmc:cmContentProperty name="Phone">
        <cmc:datatype>STRING</cmc:datatype>
        <cmc:value>(917) 555-1212 Ext. 111</cmc:value>
        </cmc:cmContentProperty>
        </cmc:cmNode>
        </cmc:cmNode>
        .
        . // More nodes
        .
  </cmc:cmNode>
</cmc:cmContent>
```

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.

Download
30 Day Free Trial
www.sandcherry.com

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

SandCherry

SIP
THE INTERNET'S NEXT GREAT PROTOCOL

By Pat Shepherd

This article will demonstrate the value of a communication platform not just for telecoms, but also for any company developing enterprise-wide applications.

BEA has entered the communication platform space in a very big way with its release of the BEA WebLogic Communication Platform (WLCP). BEA's new platform brings capabilities that used to be strictly in the domain of telecom network programmers up to the J2EE developer community. Doing this provides a number of advantages such as drastically lowering the time and cost required to build telephony services and affording the ability to create new types of telephony-related services that were never before possible – all with Java, J2EE, and Web service technology.

It is also important to understand that the user experience from a user interface (UI) standpoint is quickly changing. As Figure 1 shows, UI experience arguably moved backwards from an era of client-server applications with their rich, fat-client, front-end applications to more simplistic HTML pages. HTML-based applications were easier to build and maintain through the model-view-control architecture of servlets and JSPs, but they were fairly flat and did little to integrate the richer collaboration that some Windows-based applications took advantage of (e.g., various VB controls that could integrate with audio- and video-based media applications).

More and more, you can expect Web applications to meld traditional HTML screens with such capabilities as instant messaging, voice (through regular PSTN phones, VoIP, or soft phones), video, etc. SIP (session initiation protocol) is a protocol that lets all this happen. Many people feel that SIP is the next great protocol of the Internet because it brings an entirely new set of media and collaboration capabilities to applications. These new applications are called "converged applications" because they integrate traditional telecom-type capabilities within enterprise applications. They can now include various media (e.g., voice, video and chat) and application logic (e.g., UI logic with JSPs and business logic with EJBs).

Here are some examples of the new types of converged applications that SIP enables (this list is by no means exhaustive):

- Real-time video sharing
- Voice instant messaging
- Voice-video telephony
- Video conferencing
- Rich-media enterprise collaboration
- Click-to-call
- Autoinitiated conference calls
- Converged call-center communication

Anyone who has been involved in telephony knows that the field is currently an alphabet soup of acronyms, including IMS, SIP, PARLY, PARLYX, SMS, PSTN, RBOC, VoIP (Voice Over Internet Protocol) – and the list goes on. There are also some important elements such as registration and presence servers, which provide for registering users and tracking their availability through various channels that I cannot cover here. This article cannot possibly be a "Telecom 101" review, but I will try to give you a bigger-picture, lay-of-the-land overview so you may better understand how things fit together.

The good news is BEA's WLCP can help you become productive without necessarily knowing all

Author Bio:
Pat Shepherd has been an active member of the Java revolution since its inception in 1995. He has written for several magazines and was a technical editor for the Sams book BEA WebLogic Platform 7. He has worked at BEA for more than five years and is currently a global account architect who spreads the Java/SOA message everywhere he goes

Contact:
pat.shepherd@bea.com

the myriad of terms and technologies. This brings up an important point I would like to make based on a number of discussions I have had with architects and developers. Many people are under the impression that SIP servers only have a place in a telecom and specifically only for VoIP initiatives (see VoIP section below for definition). This is absolutely not the case. SIP is agnostic when it comes to working with the traditional Public Switched Telephone Network (PSTN), VoIP, or wireless networks. There are gateways that can take traffic between these various delivery technologies.

Current State of the Telecom Industry

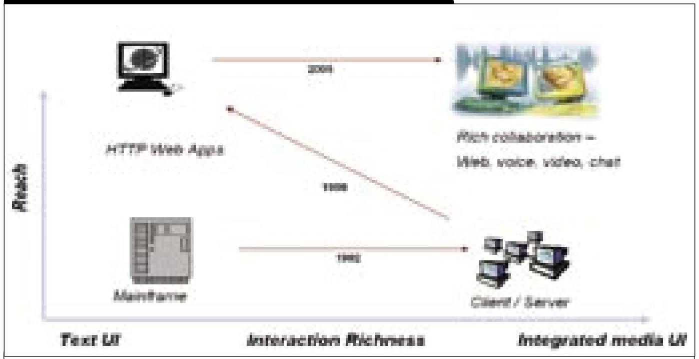
The telecom industry is undergoing a number of radical changes. There are two distinct trends going on. One is the need for carriers to create new revenue opportunities by creating converged video, voice, and data applications – often referred to as a "triple play." Online real-time multimedia and two-way interactive services (such as games) are some examples. The other trend is the continuing effort to lower the cost of running traditional PSTN networks and operations and eventually move to the more cost-effective VoIP networks.

Race to the Home

Fiber optic cable is being run to homes and businesses very rapidly. A few months ago there were guys stringing fiber on the poles outside my house along the road – but not yet directly to my house. Fiber promises to revolutionize telecommunications and networking by bringing unprecedented network speeds both at work and at home. Verizon is, for example, now offering speeds of 15 megabytes per second through their FiOS product. The economies of scale tip dramatically when these types of speeds are available for new types of applications. Indeed, the "race to the home" is on in a big way.

To continue the racing metaphor, the need for telecoms to create new services to drive differentiation is pushing them to embark on strategies to make their services better than their competitors. If they were to continue doing this with the old class 5 switches and other proprietary network elements, it would be like pressing one foot on the accelerator while the other foot mashes on the breaks – not very effective for winning the race. This is where the BEA WLCP comes in. It replaces much of the need for these older technologies and brings the power that was originally only in the domain of telecoms and the highly specialized skills of the "Network Guys" to the J2EE (or JEE) developer.

FIGURE 1

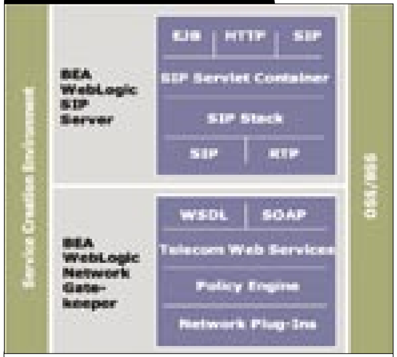


User experience

VoIP

VoIP is hardware and software that allow media calls – voice and otherwise – to be placed over IP-based networks (the Internet) using packet switching instead of the traditional PSTN. The key advantage that VoIP provides is that companies avoid the tolls and tariffs associated with PSTN. Also, the infrastructure required to host VoIP networks is based on commoditized hardware and software, whereas PSTN is built on largely closed hardware and software systems such as the class 5 switches. Generally, telecom will have both PSTN and VoIP networks that can interoperate for the foreseeable future, but will be moving to IP-only networks in the long run.

FIGURE 2



Components of the WLCP

Why Is a Communication Platform Needed

The world has changed because of the driving need to bring telecom capability out of the hands of the pure network shops and give common developers the power to create new and converged applications. As noted, this is generally termed the convergence between traditional telecom and enterprise application development. The BEA WLCP provides developers with a set of industrial-strength tools that enable them to program in a well-known environment of Java, J2EE, and Web services.

As of this writing, the WLCP has two components: the WebLogic SIP Server, and the WebLogic Network Gatekeeper, as shown in Figure 2. Though this article focuses on the WebLogic SIP Server, its sister WebLogic Network Gatekeeper is an important piece of the puzzle. When you need to provide a layer that protects and arbitrates networks and network traffic, the Network Gatekeeper steps in to provide much lower level network control such as protocol adapters, as well as support for ParlayX, a specification for opening up telecom network capabilities to applications.

So, when you tie these new core telecom capabilities with the rest of the BEA Platform (WebLogic Server, WebLogic Portal and WebLogic Integration) you have all of the tools to drive out converged services that are:

- Integrated with existing legacy applications
- Fully a part of a service-oriented architecture (SOA)
- Represented to any type of browser or device; otherwise called multichannel delivery
- Able to provide personalized views of the applications through WebLogic Portal

These combined capabilities represent something that the industry is generally terming a Service Delivery Platform (SDP). In this paradigm, the distinctions among wireless, wireline, VoIP, and Web-based delivery channels are blurred. In fact, these channels now can work together in ways they never could before.

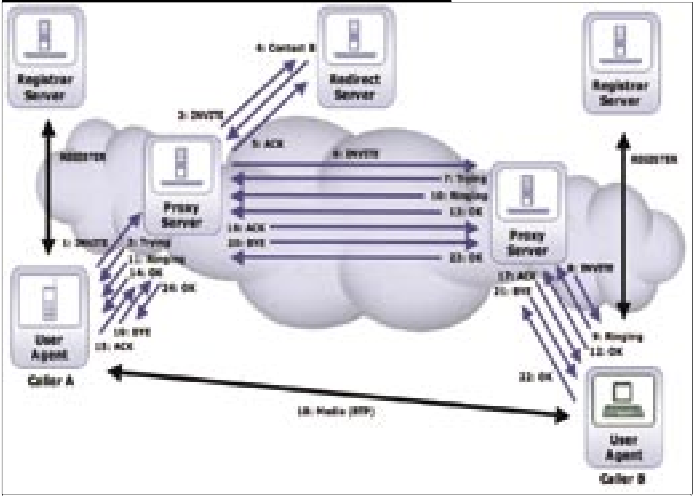
What Is SIP

Simply stated, SIP is a protocol provided by IETF in RFC 3261 that provides signaling capabilities that were once in the purview of protocols such as Signaling System 7 (SS7). SIP has been incorporated into the Java language under JSR-116 (the "SIP Servlet API" – see: www.jcp.org/en/jsr/detail?id=116). WebLogic SIP Server is an application server that supports the SIP Servlet API. There are marked similarities between SIP Servlets and HTTP Servlets because they share some of the same design and coding principles. In fact, they both extend javax.servlet.GenericServlet. The SIP Servlet specification has a relatively small number of classes, interfaces, and methods, which makes it fairly easy to learn the basics quickly.

SIP's job is to "ping" other devices and essentially do an INVITE message requesting that a conversation be set up between the devices. This is typically called call control – setting up and tearing down media streams such as phone calls. The conversation can be of almost any type and is more a function of the devices than of the SIP protocol. This conversation can be one phone inviting another to open up a voice media stream or it could be a video-enabled application requesting a video media stream. The media stream, known as Real-time Transport Protocol (RTP), is a standardized packet format for delivering audio and video (and other media) over the Internet and is independent of the SIP signaling.

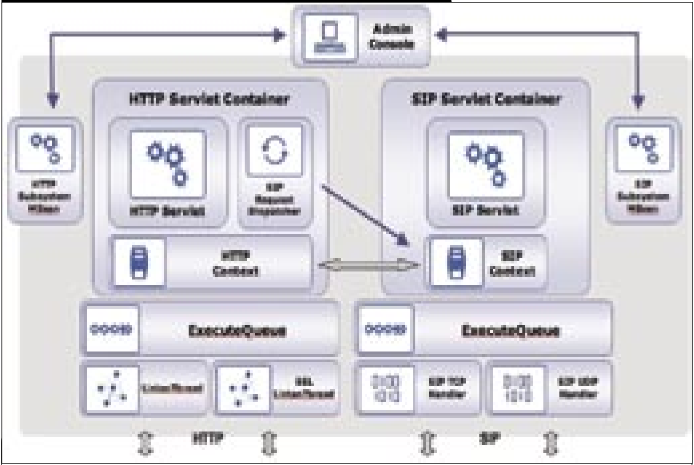
As an example of a call flow, Figure 3 shows the signaling steps that might be used in the normal course of one phone (User Agent in SIP parlance) attempting to establish communication with another phone. Caller A is registered with a registration server and sends an invite to an intermediary proxy server that looks up Caller B's location and uses that information to contact yet another proxy server that will send the request to the physical device with which Caller B is registered. The acknowledgements are passed back and forth and ultimately in step 18 an RTP media stream is set up between the two callers.

FIGURE 3



An example of a call flow

FIGURE 4



WebLogic SIP Server architecture

SIP Code Example

Listing 1 is an example of a SIP Servlet that sends back a 200 OK response to the SIP MESSAGE request. This example shows the similarities between a SIP Servlet and an HTTP Servlet:

- Servlets must inherit the base class provided by the API.
- HTTP Servlets must inherit HttpServlet and SIP Servlets must inherit SipServlet.
- Methods such as doInvite(), doBye(), and doRegister() must be overridden and implemented.
- HTTP Servlets have doGet/doPost methods that correspond to GET/POST methods. Similarly, SIP Servlets have do<MessageName> methods that correspond to the method name (in the aforementioned example, the MESSAGE method). Application developers override and implement necessary methods.

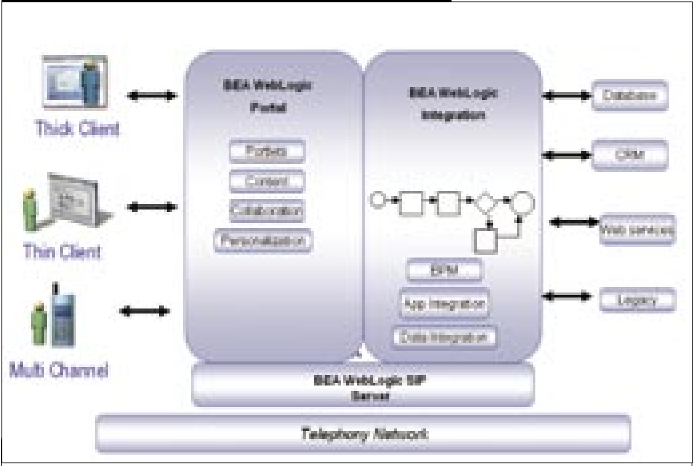
The life cycle and management methods (init, destroy) of SIP Servlet are exactly the same as HTTP Servlet. Manipulation of sessions and attributes is also the same. Although this example does not show it, there is a deployment descriptor called sip.xml for a SIP Servlet, which corresponds to web.xml in HTTP Servlets. Application developers and service managers can edit this file to configure applications.

WebLogic SIP Server

Because the WebLogic SIP Server is converged with the WebLogic Application Server, developers and administrators will already be familiar with the scalability, manageability, and programmability of the SIP Server. They can leverage all of the power of Java and J2EE and manage production applications in the same manner that traditional J2EE applications are managed. Figure 4 shows that WebLogic Server hosts the HTTP Servlet container and the SIP Servlet container. Both have full access to all of the core Java and J2EE capabilities implicit to WebLogic Server. An additional benefit is that the latency between operations performed in the SIP Servlet container are minimized because they are in-process with the rest of the application artifacts such as EJBs that might lend business-logic support to the application.

Indeed, without a converged container there is far more complexity because of the additional effort in programming and operating separate environments for signaling functionality and business logic. If you imagine one of the new services, such as "click-for-service-rep," there is a need to integrate with existing operational support systems (OSS), and possibly with billing support systems (BSS). Without the BEA converged container, you would likely need to create three separate SIP, Web, and integration tiers, each having its own set of disparate technologies to learn and manage.

FIGURE 5



SIP-powered enterprise applications

Figure 5 shows how BEA WebLogic Portal, Integration, and SIP Server all work in concert with each other to provide a powerful and unified framework for supporting UI, integration, and next-generation signaling, respectively.

Summary

This article discussed the importance of the SIP protocol in developing new types of applications that were never before practical when traditional telecom capabilities were not within reach of enterprise developers. Because it opens up so many possibilities for next-generation collaborative applications, many believe that SIP is truly the next great Internet protocol. The Java community has an API – the SIP Servlet API (JSR-116) – that provides a blueprint for developing SIP-based applications. Now, BEA provides an enterprise-grade SIP Server that is unique in its scalability and manageability. Also, because of the converged SIP + J2EE container, BEA offers the power and flexibility of both SIP and J2EE to enterprise developers. Finally, BEA's SIP Server ties into the entire BEA WebLogic Platform, thus providing developers with a unified application development framework for easier, faster, and more cost-effective development. For more detail on the BEA WLCP release, please see www.bea.com/framework.jsp?CNT=index.htm&FP=/content/products/wlcom/.

Listing 1: SimpleSIPServlet.java

```
package com.bea.example.simple;

import java.io.IOException;
import javax.servlet.*;
import javax.servlet.sip.*;

public class SimpleSIPServlet extends SipServlet {

    protected void doMessage(SipServletRequest req)
        throws ServletException, IOException
    {
        SipServletResponse res = req.createResponse(200);
        res.send();
    }
}
```

Mom, Apple Pie, and the Bottom Line

– *continued from page 5*

As it is deployed, customers, account reps, and other users will still have their customized experiences, but behind the screen sharing of the services is expected to increasingly improve the company's bottom line. For example, instead of a yearlong effort when one organization wants to sign customers up differently, it's only a few weeks or months. They don't have to build and test new code; they have to use existing code components in a different order. Moreover, its flexibility and component-based design should keep them from accumulating new technical debt.

Next to the more immediate gratification of Web services, service orientation may seem very mom and apple pie, but articles I've read suggest that as much as 75 to 85 percent of corporate IT budgets are spent just running what they have. Individual technologies can at best deliver only marginal improvements in ROI. Our arsenal of tools and technologies and techniques need the guidance of architecture to do better. Done well – and I realize that we may not have yet done this – mom and apple pie means delivering more value and doing it faster with that remaining 15 percent of the budget. That capability becomes strategic to the enterprise.

With the combination of standards and robust platforms like BEA we have it in our grasp more than ever to build *enterprise* systems. It may be less glitzy than technology, but in the end architecture pays better.

WEEKEND WITH EXPERTS

Get a Java injection on the go!

Spend a weekend and have a lunch with experts in an intimate learning environment in the city near you.

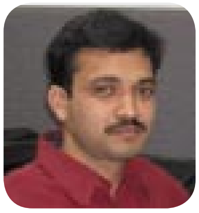
No salesmen allowed. Join our technical discussions and hands-on workshops.

The next Roadshow is in Philadelphia on December 10 and 11.

www.weekendwithexperts.com

Effective EJB

MAKE EJBS WORK FOR YOU



By Shankar Itchapurapu

Java development is at a crossroads. The open standards have done lot of good for the Java platform and language, but they have brought in some problems too. Developers are often drenched in the complexities that surround Java development. Worse yet, these complexities are so overwhelming that the actual business problems take a back seat.

The J2EE specification provides a lot of APIs, standards, and open ends that allow architects, designers, and developers to build superior enterprise systems. Care must be taken to engage in the balancing act of choosing the right technology.

Technology development has created more confusion for the developers (unless developers are versatile) rather than helping them to resolve the issues. Often, architects and developers spend most of their time supporting their chosen framework instead of concentrating on the business problem in hand.

This article discusses the techniques involved in designing and developing superior J2EE applications using the Enterprise Java Beans (EJB) specification. While the article does not intend to show how to use EJB, I'll discuss the various pitfalls in EJB development and essentially focus on the real-world antipatterns that sneak into your development.

Learning things the hard way is nice, but due to the ever-shortening development cycles, it is smarter to learn from mistakes made by other people. Having a comprehensive understanding of where the dangers are will help one to take a proactive strategy, which is precisely the theme of this article. We will start with the EJB Context in J2EE applications and then discuss some potential dangers that exist in EJB development.

Effective EJB Decision

Software is an engineered art and the engineering is continuous. With each exciting technology such as the EJB specification, weary engineering minds tend to adopt the exciting new technology in hurry. This is one reason most EJB projects fail. Often they fail so badly that they finish at point of no return. It would be wiser to engineer the choice of EJB rather than to embrace it just because it's a hot, sexy, and promising technology.

With regard to component architecture for building distributed, transactional, and persistent business solutions, the decision to use EJB in software projects demands a careful analysis and proactive planning with sound engineering practices.

Author Bio:
Shankar Itchapurapu is a software engineer at BEA Systems, India. He holds a Master's degree in Computer Applications. You can e-mail Shankar at shankar.itchapurapu@gmail.com. This document was created with Win2PDF, which is available at www.daneprairie.com. The unregistered version of Win2PDF is for evaluation or noncommercial use only.

Contact:
shankar.itchapurapu@gmail.com

Choosing Unwisely (A Golden Hammer for a Fly)

"When you have a hammer in your hand, everything looks a nail." This phrase befits most EJB choices. EJB may not be the best suited for most projects simply because the following are true:

- You have already paid for the server and have an EJB container ready for use
- You are doing enterprise Java development (is your business really an enterprise?)
- You want a fully portable architecture (is EJB really portable?)
- You have a container and EJB allows you to delegate most of the jobs to the container (this would reduce the development cost/ time and ensure fast time to market)

Solution (Choose Wisely)

Figure 1 shows the trade-off between project size and cost for a simple POJO (Plain Old Java Objects) solution and an EJB solution.

Choosing an EJB Solution

- Strive to achieve the break-even point quickly
- Choose EJB if your project is complex (as seen in the graph, EJB projects start complex and expensive, but ramp up more slowly with project size)
- Choose EJB if the services provided the EJB container are needed for the business; in other words, choose EJB if the answer is YES to the following questions:
  - Do your components need to be distributed?
  - Do you need to handle global transactions?
  - Do you have security requirements at the business-component level?
  - Do you need a persistence framework that rund inside a managed environment?
  - Does your application need high scalability?

Not Everything Is an EJB

The most often seen misinterpretation of EJB among junior- and intermediate-level users is that when you use EJB, every component is an EJB.

No. This is rarely true. A component or a subsystem may be a true EJB candidate while others can be just POJOs. This mixed-mode design is tough, but engineering the design decisions at each component/subsystem level makes life easier during the roll out.

Coarse-Grained and Fine-Grained Services

One key aspect of choosing EJB at the subsystem level is to understand the EJB services. EJB components come in two flavors: a set of work horse components called *session beans* and a set of persistence components called *entity beans.* Session beans are generally coarse-grained services that take advantage of a container's ability to provide distribution, transaction management, and security. For a non-EJB component, the implementation of these services is the developer's responsibility. Components that make heavy use of these services are often the right candidates for session EJBs.

The EJB specification mandates support for fine-grained persistence services through entity beans. Each entity bean can itself be again distributed, transaction aware, and secure and hence there is a complex mixture of fine-grained and coarse-grained services. Often, this mixture makes entity bean components very difficult to manage. If a component needs to be just persistent, there is seldom a need for the component to be an EJB. Other popular, lightweight persistence frameworks exist (e.g., JDO, Hibernate etc.) that are easier to maintain.

Effective EJB Interfaces

One of the key aspects of EJB design is creating interfaces. Interfaces are the lingua franca of the EJB components. They serve as a vehicle to expose the services provided by the EJB component to the external world. Poor interface design would lead to EJBs that are hard to maintain and change.

Considerations for Interface Design

What is the size of your network pipe? At the end of the day, EJB is RMI. It involves remote procedure calls. The location transparency will just add some more network complexity. The EJB calls involve significant marshalling and unmarshalling of parameters over the network. Care should be taken when designing the remote interfaces. If you have a big pipe and can stream large data quickly through it, then the interfaces can be coarse grained. On the other hand, if you have a smaller bandwidth, then you're probably better off having finer-grained interfaces and lightweight parameter marshalling.

Do we façade? The decision to have a façade as a gateway for your other EJB components should be considered well ahead of time. It cannot be an afterthought. Once the decision has been made to have a façade, you can design the façade to return objects that are aggregations of values. The fine-grained access will be handled by the façade and is effective because it is generally within the container. This way we can reduce the number of round trips to the remote server.

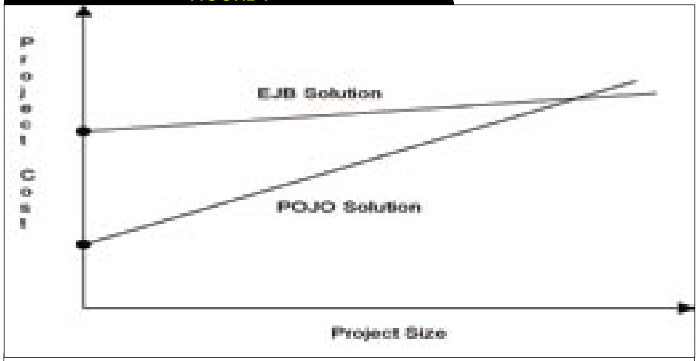
Ok. We façade. But how good is our facade? When designing EJB interfaces the crucial points are that, for the purposes of designing remote interfaces, state should be avoided where possible and remote interfaces should be sufficiently coarse grained. Coarse graining interface design does not necessarily mean bundling all of the methods of your EJB layer inside one façade. Most often it is good to have multiple facades serving as gateways to your EJB components. When designing EJB-based architecture the safest approach is to find a balance between an object-oriented domain model and the procedural remote service layer.

Effective Exception Handling

Exception handling is another of the most muddled areas of EJB development. Exceptions are basically a breakaway from the expected behavior. These breaks are very common in distributed architectures because there are too many heterogeneous environments involved that are all connected by the network. A single network failure can trigger a catastrophe. Handling exceptions in EJB is particularly complex, and EJB involves two types of exceptions. The following are some guidelines for handling exceptions.

- ***Refractor Logic out of Exception Handling Blocks.*** Having application logic inside exception handling code is a bad practice that often leads to code that is difficult to understand, change, and debug. The particular case in EJB programming where logic is deliberately pushed into exception handling code is transaction rollbacks. Rolling back transactions when there is an exception confuses program flow. Also, rolling back a container-managed

FIGURE 1



The trade-off between project size and cost for a simple POJO solution and an EJB solution

transaction is not a straightforward task. EJB specification recommends use of EJBContext.setRollBackOnly() instead. Second, transaction rollback during an exception is not always appropriate. Third, transaction rollback falls outside of the transaction implementation. If an application throws an exception when running inside an EJB with notSupported transaction attribute, there is confusion as to whether to roll back the suspended enclosing transaction or not.

- ***Don't Swallow Exceptions.*** Exception logging is an important part of any component design. One of the most common mistakes made by developers is to just print the exception and continue as if nothing has happened. Whenever there is an exception, it is an indication that something has gone wrong. The error could be on part of the container or in your application code itself. Simply logging exceptions without much detail makes debugging very difficult. Java provides an elegant way of logging the complete exception trace. In most cases, the exception trace would provide all of the information that is required to debug a problem. It is also better to have a custom verbose attached to the exception trace to help make understanding the situation easier.
- ***Throw the Right Exception. Always.*** One bad practice in many EJB implementations is the tendency to wrap application exceptions into Container exceptions and throw them at the client. This practice stems from the misconception that the container handles all things for you when you are developing an EJB. However, container exceptions are specialized cases where there is a failure on part of the container, while the application exceptions are a run-time application failure. An example of application exception is invalid credit card number. This should not be wrapped inside a container exception like EJBException. The application exceptions are to be gracefully handled as in any other normal Java class.
- ***Look for Hot Potatoes.*** A hot potato is a particular situation where a MDB repeatedly receives a same message and throws an exception while handling the message. When a JMS server does not receive an acknowledgement for a message delivered to a consumer, the server's only recourse is to resend the message. This sets the stage for repeated delivery of the same message indefinitely when the consumer that is processing the message does not handle an exception condition properly.

When developing MDBs care should be taken to identify potential areas of code that cause the hot potato situation. The solution for this is to break the acknowledgement and message processing into two execution paths. Acknowledge the message immediately once received. When an exception occurs during message processing, you may possibly write the exception to an error queue.

Effective EJB Persistence

The EJB specification addresses the persistence services through a special class of beans called entity beans. Entity beans are aimed at abstracting the domain data model. As such, they represent rows in a database table if the underlying datasource is an RDBMS. Initial specifications sparked a lot of controversy about entity beans and their usage. Henceforth, the persistence API has been rewritten twice in 1.1 and 2.0 specifications.

Because persistence is a fine-grained service it does not fit naturally into the coarse-grained nature of EJB services. This brings in a lot of confusion about how best to use entity beans. Before the decision to use entity beans has been made, a careful evaluation of alternate persistence models is advised. If a decision has been made to use entity beans, the following problems should be carefully considered and evaluated.

Exposing Entity Beans Directly to the Client Through Remote Interfaces

Due to their fine-grained nature, entity beans provide fertile grounds for antipatterns when exposed directly to the client. One of the most popular problems that is likely to be encountered is the n + 1 problem. This means you will require n + 1 remote calls to retrieve n attributes of a business entity. The extra call is to obtain the remote stub from the EJB container. Addressing the same retrieval function through a plain POJO DAO is much easier. All you will require is a JDBC call to retrieve the rows.

A more subtle problem associated with exposing entity beans is the loss of transaction integrity. Consider an entity bean with three mutator methods. Further assume that a client needs to update two columns of the entity represented by two mutator methods in a transaction. How does the client ensure that transaction integrity is maintained between two successive update methods on the entity bean? The only way out is to use a session façade to wrap the entity bean.

"Before the decision to use entity beans has been made, a careful evaluation of alternate persistence models is advised"

Attend the Second Annual

Register by September 9 to take advantage of the Super Early Bird Rate!

Software Test & Performance CONFERENCE

More than 60 keynotes, tutorials and classes!

SEE WHAT ATTENDEES SAID ABOUT LAST YEAR'S CONFERENCE:

"This was the most practical conference I have been to in 18 years."

Mary Schafrik, B2B Manager/QA & Defect Management
Fifth Third Bank

"Good for hands-on test managers and testers."

Steve Boykin, Deputy Program Manager
Citizenship and Immigration Services
Department of Homeland Security

"Extremely informative, and talking to colleagues in the field was incredibly enlightening."

Shari Pagley, Testing Supervisor
Sunrise Senior Living

"Definitely go for it. It's worth every minute and second of your time."

Felix Choy, Software Engineer
Avaya Inc.

"New tips and best practices in software testing were presented by excellent speakers and professionals."

Ella Naydorf, QA/Test Manager
Booz Allen Hamilton

"Very diverse set of sessions, passing on great and door-opening information and techniques at a level that many can comprehend. The speakers were excellent and approachable. The conference was very well organized!"

Jessica Navarette
Software Engineer
Alion Science & Technology

November 1-3, 2005
at the Roosevelt Hotel
New York City

www.stpcon.com • www.stpcon.com • www.stpcon.com • www.stpcon.com

Produced by

BZ Media   SD Times — Software Development — The Industry Newspaper for Software Development Managers   Software Test & Performance

For information on exhibiting at or sponsoring the STP Conference, contact Donna Esposito at 415-785-3419 or desposito@bzmedia.com

Avoid Application Joins

Managing entity relations within the application code will cause serious performance issues. Java is not optimized for database lookups. It is a bad programming practice to emulate application joins inside Java code. Databases use sophisticated technologies to optimize query access plans and consequently minimize the number of data comparisons. For example, the act of looking up for a Person entity when given an address entity within the application does not scale. These types of relations are best expressed in terms of a CMR field (from EJB 2.0). The number of comparisons that an application makes inside a loop construct in an attempt to simulate a relational join will have a severe impact on application performance.

Do Not Use Long Primary Keys

Designing entity beans with long primary keys results in performance degradation when the data in the underlying data store increases. Primary keys are used by the database for lookup. Database indexes make heavy use of primary keys to build hashes. Long primary keys will require more computations to hash and compare than shorter keys. Databases cache index fields to optimize performance on relational joins. Long primary keys tend to eat lot of space in the cache area, thereby opening the gates for a potentially thrashing situation.

However, how long is long? There is no bench mark for how long a primary key can be. This length depends on the volume of data anticipated in the target database table.

Some Guidelines for Effective EJB Performance Tuning

Your system needs some tuning irrespective of any good design, coding, and engineering practices adopted during development. This is particularly true of EJB-based systems as they are complex enough to warrant high testing. The following are some guidelines for performance tuning.

- ***What is performance?*** Performance is the measure of an application's ability to live up to the expectations within the environmental constraints of an enterprise.
- ***Performance tuning the software engineering way – know the limitations.*** Expecting throughput increase by a rate of 10X with only 32MB RAM and DEC PDP 11 machine is impossible.
- ***Measure, don't guess!*** Define what is expected in absolute terms. Simply iterating that performance needs to be improved will not help. Come up with an absolute figure; for example, method mytrans() should complete in three seconds. Most times it is advisable for the end users to come up with some expected performance numbers.
- ***Use the container.*** EJB containers are designed to provide a higher level of scalability through clustering, load balancing, and other means. It is important to learn the capabilities before you start tuning your performance.
- ***Good coding is second to none.*** Use design patterns wherever and whenever applicable to enable a high level of maintainability. This will help by enabling easier tuning of applications.
- ***Collect statistics***. Before and after each step of tweaking and tuning, collect statistics. These numbers are invaluable to further planning. Compare the numbers, analyze the improvements achieved, and then plan forward if necessary.
- ***Take one step at a time – Rome was not built in a day.*** Don't do everything at step 1. Go step by step and measure at the end of each step. This will help to isolate the performance bottleneck.
- ***Know when to stop. Knowing when to stop is the key to tuning.*** Overtuning will have dire consequences. Well-documented performance criteria will help identify when to stop further turning.
- ***Be among good workmen – choose your tool.*** Use tools to guide you through the tuning process. For example, JUnitPerf will help you find the response times in various scenarios. Profilers are great tools for isolating the problem areas.
- ***Do not PLAN, do planning.*** Do not document a tuning master plan and stick to it. Performance tuning is an iterative and repeatable process. At the end of each step, measure and replan the process and define new horizons. Master plans are not practical for software turning.

Some Loose Ends

Using XML as a Silver Bullet

Filling JMS messages with XML in the name of flexibility and scalability is no panacea. Most designers/developers tend to overuse XML because it's new and hot. Overuse of XML when smart alternatives exist will severely impact application scalability.

XML is a great technology for bringing together heterogeneous environments, but it is best used judiciously. Bear in mind that XML brings in some costs too. Since XML is typeless (everything is a string), there is a type conversion and checking that need to be done. Also, parsing huge XML messages is a resource-consuming process. The overuse of XML is generally seen in JMS implementations. In this case, the MDB spends lot of time type checking and parsing the incoming XML message.

Do Not Think Too Much About Portability

One of the goals of EJB specification is to allow creation of portable components. However in reality, it is necessary to depend on extensions and enhancements that are provided by your container provider. For example, every vendor has his own proprietary deployment descriptor that allows for tuning the behavior of EJB instances. It is important to recognize that portability, though a desired feature, is not necessarily a requirement. Organizations seldom change application servers, and it is likely that you'll deploy your EJB in the same server. Thus it is advisable to take full advantage of additional features provided by your container provider.

Use Entity Beans Sparingly

As already discussed, entity beans do not naturally fit into the coarse-grained nature of distributed services. As such, they are very resource consuming and hard to change and maintain. The original goal of entity bean use is that we be able to deploy a domain model on Machine A and another domain model on Machine B, and they interoperate seamlessly through location transparency. This is seldom true because of the connecting glue called network. This resulted in developers trying to tweak the entity bean model only to make things worse.

As the lightweight container moment gains popularity, we have very good alternatives for providing transparent persistence. Hibernate, for example, comes from the open source world. It provides the transparent persistence services to POJOs. It is easily configurable and does not need a full-blown J2EE container. Sun has a new standard for transparent persistence through JDO.

WHY 'ECLIPSE DEVELOPER'S JOURNAL' AND WHY NOW? PG 2

eclipse) developer's journal

PREMIER 2005 VOL 1 ISSUE 1

http://eclipse.sys-con.com

What's New in Eclipse 3.1

Polishing the apple

PLUS...

WEB TOOLS PLATFORM: J2EE DEVELOPMENT THE ECLIPSE WAY

"BEING A BETTER PLATFORM" INTERVIEW WITH MIKE MILINKOVICH, EXECUTIVE DIRECTOR, ECLIPSE FOUNDATION

MYECLIPSE 4.0 UP ON DECK: EXCLUSIVE INTERVIEW WITH MAHER MASRI, PRESIDENT, GENUITEC

ECLIPSE IDE FOR STUDENTS: USEFUL TIPS AND TRICKS

Although JDO is in its infancy and has not been fully embraced by J2EE vendors yet, it provides a lot of promise and is a fairly simple API.

Test, Test, and Test

Distributed components are long-term investments and they play a very vital role in a company's IT infrastructure. They are heavily used and a single failure can cause huge losses. Care must be taken to ensure that the middleware components perform very well in production. This can be ensured by rigorous testing in each phase of the development life cycle. EJB components need specialized testing as they live inside a managed environment (container). From the design to coding to integration and deployment, EJB components should be tested with all real-world use cases to ensure smoother production transition.

EJB 3.0 – The Road Ahead

The EJB architecture is probably the only J2EE component that has failed so miserably in delivering J2EE's promise of increased developer productivity thorough ease of development. EJB 3.0 makes another attempt at delivering that promise by reducing EJB's complexity for developers.

EJB 3.0 decreases the number of programming artifacts for developers to provide, eliminate, or minimize the callback methods that are required to be implemented, and reduces the complexity of the entity bean programming model and O/R mapping model. The following are the key new features of the EJB 3.0 specification.

- ***An annotation-based EJB programming model:*** EJB 3.0 takes full advantage of the annotation-based programming model introduced in Java 5 (aka Tiger). In EJB 3.0, all kinds of enterprise beans are just POJOs. Annonations are used to define references, call backs, remote interfaces, create methods, etc. This makes programming much easier and the model is neat. Furthermore, it relieves the developer from the constrained framework implementation.
- ***Support for POJO-like EJB development:*** EJB 3.0 supports POJO-like EJB. This means the EJB classes need not implement the corresponding interfaces. All that is required is to set an annotation for the compiler to understand the behavior of the class. This makes converting existing POJO services such as EJBs much easier.
  One of the problems with the earlier programming model was that EJB requires a whole lot of classes and descriptors. Starting EJB 3.0, there is no need for interfaces or deployment descriptors. All that is required is an EJB class (essentially a POJO) annotated adequately to provide the run-time information to the container.
- ***The new persistence model for entity beans:*** In line with a new session bean model, entity beans in the EJB 3.0 specification are simple POJOs. Also, the EJBQL has been significantly changed to add much more functionality. Entity bean classes can be just like a plain Java Bean class, and all fields that are not marked with @Transient annotation are assumed to be persistent. The object relational mapping has changed from the abstract persistence schema model in earlier specifications to the Hibernate-inspired persistence model.
- ***Testability outside the container:*** One of the main drawbacks of EJB architecture is that it cannot be tested outside of the managed environment. Initial drafts of EJB 3.0 specification had some thoughts about testability outside the container. However, this is still in draft stage and will be a very important addition to the specification if it comes through the final version.

Summary

EJB remains one of the most important and ambitious technologies in the J2EE world. The main goal of EJB is to standardize the enterprise middleware programming. As with all complex technologies, it is very prone to misuse and misunderstanding. The key to using EJB effectively and to its best potential is to identify the business context for EJB, evaluate the design decision, apply sound engineering principles, and finally, create effective EJB. Once care is taken to use EJB properly, it delivers the promise. If not handled properly, it can become a nightmare for you.

From the decision to use EJB to the production deployment, every step requires careful consideration, analysis, and discipline. This article has provided some guidelines for and has shown some potential gray areas in EJB development. However, time and again new antipatterns crop up and it is up to the architects, engineers, and developers to tame the problems.

EJB 3.0 is definitely a very good step towards easing EJB development, and not just on the coding part of it. Being able to write simple EJBs will help facilitate maintenance and lower risks in production. J2EE, and EJB by extension, is now the default development platform for enterprise applications.

Effective use of EJB lies in understanding that EJB is just an extension to J2EE capabilities. A J2EE application does not necessarily use EJB. However, when there is a need, EJB can provide significant advantages to the application. The key to developing effective EJB is to move away from the religious feelings you may have about EJB and evaluate the technology in its own capacity without the hype surrounding it. This will help you make better choices, use EJB cleverly, and finally, to build *Effective EJB*.

Hope this helps!

References

- *Bitter EJB* by Bruce Tate, Manning Publications.
- *EJB 3.0 in a Nut Shell:* www.javaworld.com/javaworld/jw-08-2004/jw-0809-ejb_p.html
- *Hibernate:* www.hibernate.org
- *Enterprise Java Beans* by O'Reilly Publications
- *Sun Microsystems, Enterprise Java Beans page:* http://java.sun.com/products/ejb/
- *Building a better exception-handling framework:* www-106.ibm.com/developerworks/java/library/j-ejb01283.html
- *Organizing your EJB development environment:* www-128.ibm.com/developerworks/java/library/co-tipejbo.html
- *Performance tuning EJB applications:* http://dev2dev.bea.com/pub/a/2005/02/perf_tune_session_beans.html
- *Best practices in EJB exception handling*: www-128.ibm.com/developerworks/java/library/j-ejbexcept.html

Running ASP.NET Applications on WebLogic!

EASIER THAN YOU MAY THINK

By Laurence Moroney

Author Bio:
Laurence Moroney is the director of Technology Evangelism at Mainsoft and the author of several books on .NET and Web services, as well as several dozen articles that span the technology realm. A strong belief in interoperability at the human level has made him a keen enthusiast of the Visual MainWin for J2EE product, because it allows developers to work together more effectively to deliver systems that operate on diverse platforms.

Contact:
ljpm@sportstalk-ny.com

When WLDJ wanted someone to take a First Look at Visual MainWin for J2EE, we turned to interoperability expert Laurence Moroney – coauthor of a forthcoming book on Web services security and a senior architect in a major financial services house in New York City. In the course of assessing the product, Laurence in fact became more and more involved – in the end, on a staff basis – with the company behind it, Mainsoft. So this First Look should be read with that basic journalistic disclosure in mind.

When WebLogic workshop was first conceived and presented to the public, it was done so as an alternative to the hugely popular Visual Studio.NET development environment – one that was just as easy to use, and equally if not more powerful, because it would allow you to build J2EE applications, including EJBs, in as easy a manner as a VB programmer could put together a Windows-based object.

However, many shops that have existing assets in .NET and build in C# would have to port all that code into Java using Workshop, and effectively throw away their existing investments should they want to run on the WebLogic J2EE platform. This became, and still is, a huge hurdle in migrating between the platforms. In addition to this, a real strength of WebLogic Workshop 8.1 is in its ability to easily construct EJBs and to easily construct workflow applications and expose them as services. On the front end, it has the very useful pageflow technology that can be used to construct UIs that consume these, but many would argue that there is no better productivity alternative for front-end Web applications than ASP.NET with Visual Studio.NET. It would be nice if you and your team could use WebLogic for what it does best – middleware EJBs and Workflows, and ASP.NET for what it does best – front ends.

Now there is a way: Visual MainWin for J2EE is a product from Mainsoft (dev.mainsoft.com) that offers a unique and innovative solution to this. In a nutshell, this product, nicknamed Grasshopper, takes the Microsoft Intermediate Language (MSIL) generated by .NET and converts it into Java Byte Code, supported by a Java port of the Mono libraries. This way your C# code can be compiled and run on a J2EE application server such as WebLogic on Windows, Linux, or anywhere else it is supported. It's a wonderful way of capitalizing on the usability of Visual Studio.NET with the run-time reliability and characteristics of WebLogic.

In this article you will take a look at what it takes to deploy your existing ASP.NET front ends to WebLogic 8.1, and to interop with assets such as EJBs that are already running on that platform.

How It Works

The concept behind this product is very simple – but very effective. Mainsoft is a major contributor to the Mono project, and they have used their tool to port the Mono source code to Java. This provides the namespace support so that the ASP.NET namespaces can be called from Java. It then takes your C# or VB.NET code and cross-compiles the MSIL generated by the .NET framework compilers into Java Byte code. This is then a pure Java solution that will run on your J2EE application server. They also add support to consume Java References and EJBs.

When developing an application you simply create an ASP.NET application and use their wizard to convert it to J2EE, or you can create a C#/VB.NET for J2EE application using the new project types in Visual Studio.NET after installing.
The rest – developing, debugging, code completion, intellisense, etc. – all work cleanly within Visual Studio.NET. It's very compelling, and if you or any member of your team is comfortable in this development environment, you'll be amazed at how cleanly it integrates. There's something very cool about writing C# and running and debugging it on WebLogic as cleanly as you would on IIS.

Getting Started – A Very Simple Example

This first example takes a standard C# Web form, adds an EJB reference to it to allow it to consume an EJB off a WebLogic 8.1 server, and then compiles the Web form to run on WebLogic.

To get started, use WebLogic Workshop to create a simple EJB that implements the following method:

It is very simple and just returns "Hello World" and the caller's name.

You'll need to install the enterprise edition of Visual MainWin for J2EE to continue, because only the enterprise edition supports BEA WebLogic. You can find this at dev.mainsoft.com. They also have a free edition of the tool that supports Tomcat. Once you have downloaded and installed the tool, you can launch Visual Studio. NET and create a simple Web form like the one shown in Figure 1.

Adding a reference to an EJB is very straightforward – and will look very familiar if you are used to consuming Web services in Visual Studio.NET. If you right click on the "References" node in the solution explorer, you will notice two new entries – "Add Java Reference" and "Add EJB Reference." The former allows you to consume JAR files and use them from within ASP.NET, and the latter does the same but for EJBs. If your EJB (from above) is deployed to WebLogic and the Application Server is running you can use the Add EJB reference, and you'll get the dialog that is shown in Figure 2.

You can add the EJB reference using either a path to the JAR file that contains it, or by using JNDI.

Once the EJB reference is made, Visual Studio.NET and Visual MainWin for J2EE create a proxy class that allows you to talk to it. This then allows you to code to the J2EE using all of the productivity features of Visual Studio.NET, such as autocomplete.

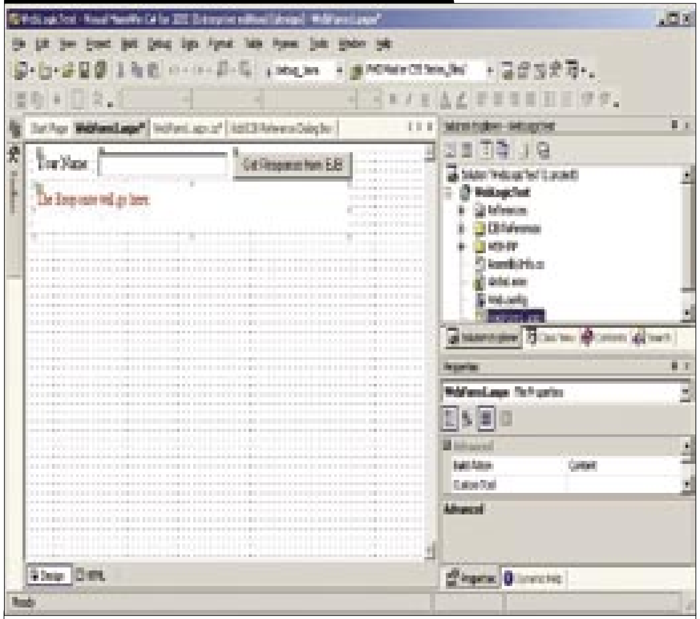
On the simple ASP.NET UI that you created earlier, you can now add the following code (the textbox should be called "txtName" and the label should be called "lblResp" for this code to work):

```
localhost.ejbhwRemote myEjb = new localhost.ejbhwRemote();
string strTest = myEjb.echoHelloWorld(txtName.Text);
lblResp.Text = strTest;
```

Using a WebLogic Workflow

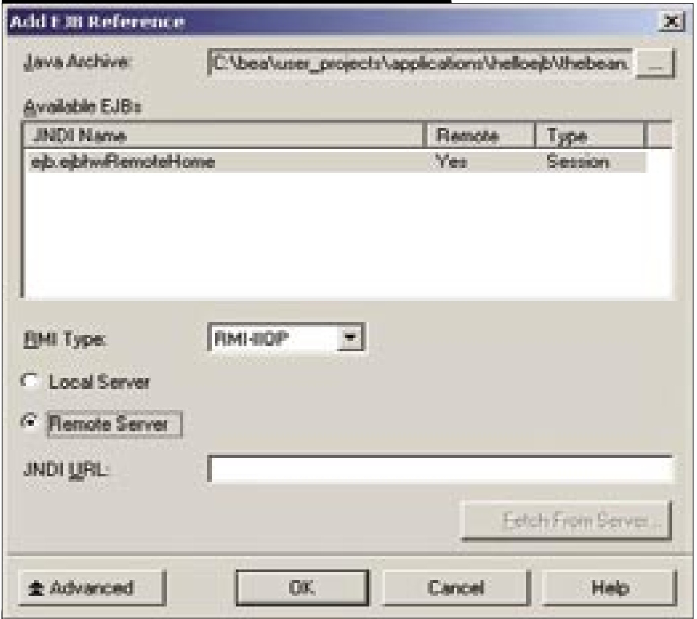
One of the massive strengths of the WebLogic Workshop 8.1 is the impressive workflow engine. This allows you to graphically design workflows and processes, and to interface them with

FIGURE 1



A simple ASP.NET Web application

FIGURE 2


Add EJB Reference

Java Archive: C:\bea\user_projects\applications\helloejb\thebean...

Available EJBs:

JNDI Name	Remote	Type
ejb.ejbhwRemoteHome	Yes	Session

RMI Type: RMI-IIOP

Local Server [✓ Remote Server]

JNDI URL:

Adding an EJB reference

external components such as Web services or EJBs. In many ways it is the foundation of an enterprise service bus. In this section you will look at how you can tie your ASP.NET user interfaces to these workflows using Web References, and deploy both the workflow and the UI to WebLogic.

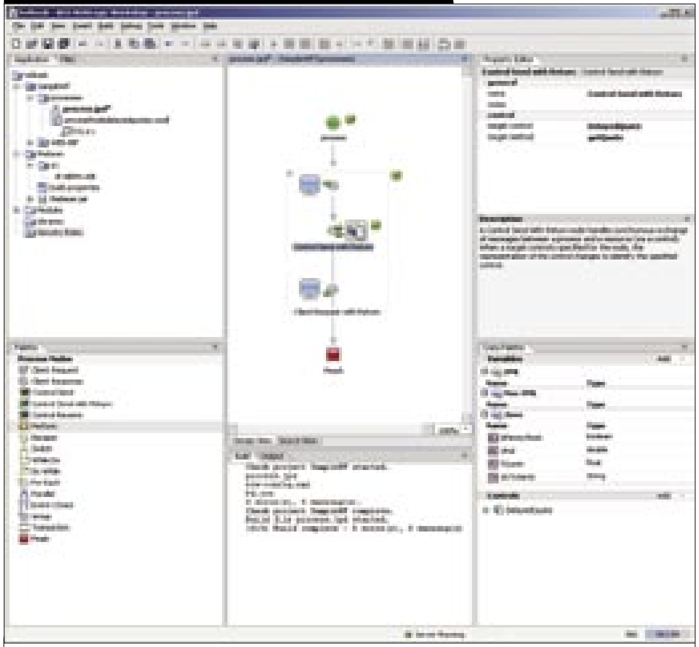
First you'll need to create a simple workflow. It is shown in Figure 3. Your application needs to be set up to run on an integration domain in WebLogic in order for this to work.

This example is very simple and uses a single control, which is a reference to the XMethods delayed stock quote Web service. The process takes an input parameter from the user and dispatches it to the service. When the service replies, the reply value is sent to the client. The WebLogic process engine allows for very complex interactions, including interfacing to Web Services, EJBs, and other processes, as well as decision trees, transactions, and much more. If you aren't familiar with it, it is well worth looking at, and it's a very powerful engine.

In this example the process is a linear one that uses a "Client Request with Response" node. The Request accepts a string, the Response issues a float. In between these a Control Send with Return (synchronous) is used. This consumes the publicly accessible delayed stock quote Web service from XMethods, whose WSDL is available at: http://services.xmethods.net/soap/urn:xmethods-delayed-quotes.wsdl. When you execute this process you get the BEA test harness, which you can see in Figure 4.

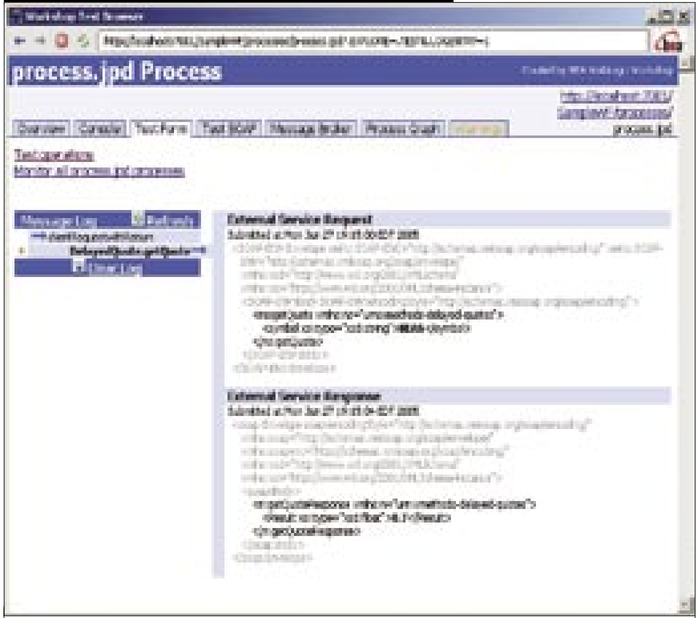
Because this process exposes a WSDL endpoint you can now consume the entire process as a Web Reference within Visual Studio.NET. Should you desire your runtime environment to be Java-based, and you want to capitalize on existing ASP.NET skills or assets, this is very useful.

FIGURE 3



A sample workflow/process project

FIGURE 4



Running the workflow/process

Conclusion

Many companies, both small and large, have development teams that specialize in both .NET and Java – with a typical deployment environment scenario of mission-critical applications running on J2EE applications servers and front-end GUI applications running on .NET. The sheer productivity characteristics of Visual Studio.NET make for a compelling argument for developing these Web applications. However, you may face a challenge when you want to use EJBs in such a scenario, because you have to build a wrapper around them, perhaps as a Web Service, so that the GUI tier can handle it. In this article you took a quick look at the Visual MainWin for J2EE product from Mainsoft that allows you to take a "best of both worlds" approach to this. By using the tool you can have your .NET developers run their code on WebLogic 8.1, as well as directly consume EJBs and WebLogic process applications. It helps you to effectively use your developers as well as streamline and simplify your deployment – instead of mixed data centers, you can have a uniform WebLogic-based data center that runs all of your applications – your ASP.NET, your Java, and your J2EE applications, which makes for much easier management. It's a compelling option that sometimes might seem to be too good to be true. From experience I have found that it isn't – it manages the automatic port of the bulk of your code, including data access, with little or no modification. It's well worth checking out – and the evaluation edition can be downloaded from dev.mainsoft.com.

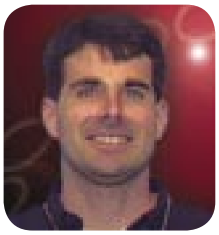
WLDJ ADVERTISER INDEX

ADVERTISER	URL	PHONE	PAGE
Blog-n-Play.com	www.blog-n-play.com	201-802-3000	36
Eclipse Developer's Journal	http://eclipse.sys-con.com	888-303-5282	27
EV1 Servers	www.ev1servers.net	800-504-SURF	9
HP	www.hp.com	800-752-0900	13
Information Storage & Security	www.issjournal.com	888-303-5282	47
Intersperse	www.intersperse.com	800-340-4553	2, 15
IT Solutions Guide	www.sys-con.com	201-802-3021	45
JDJ	www.sys-con.com/jdj	888-303-5282	39
LinuxWorld Magazine	www.linuxworld.com	888-303-5282	33
Motive	www.motive.com/within1	512-531-2527	52
NetIQ	www.netiq.com/solutions/web	408-856-3000	51
Parasoft	www.parasoft.com/soaptest_wdj	626-256-3680	3
Quest Software	www.quest.com/wldj	949-754-8633	7
SandCherry	www.sandcherry.com	866-383-4500	15
STP Conference	www.stp.com	415-785-3419	25
Smart Data Processing Inc	www.weekendwithexpects.com		21
SYS-CON e-newsletters	www.sys-con.com	888-303-5282	31
SYS-CON Publications	www.sys-con.com/2001/sub.cfm	888-303-5282	37
SYS-CON Reprints	www.sys-con.com	201-802-3026	33
Tangosol	www.tangosol.com	617-623-5782	11
Wily Technology	www.wilytech.com	888-GET-WILY	5
WLDJ	www.sys-con.com/weblogic/	888-303-5282	35

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

Rating WLI 8.1 on Process Patterns

WLI AND BPEL DON'T ALWAYS FLY IN FORMATION IN THE PATTERNS AIR SHOW



By Michael Havey

Every aircraft can take off, fly straight, and land, but few are capable of the dazzling rolls and loops displayed at air shows. When judged on aerobatics, some airplanes are superior to others.

Every BPM process language, analogously, can implement basic sequential control flow, but most languages struggle to support the most advanced splits, joins, loops, and synchronizations. These process maneuvers are known as patterns; like an aerial maneuver through thin air, a process pattern is a kind of *movement* through a business process, modeled as a particular arrangement of activities.

The Web site www.workflowpatterns.com, created by computer scientists specializing in BPM, is the foremost source of material on process patterns. The two most distinctive features of this site are a catalog of 20 patterns and ratings of numerous standard and vendor-specific process languages on their support for the catalog. Business Process Execution Language (BPEL), the most popular process language, gets a very high score, supporting 14 of the 20 patterns according to the site's analysts. (If you look at the other scorecards, you will see that 70 percent is a comparatively good grade!)

BPEL's glowing scorecard bodes well for WebLogic Integration (WLI), which, starting with version 8.5, will adopt BPEL as its process language. However, what about the large installed base of processes deployed on WLI 8.1, whose process language, Process Definition for Java (PD4J), is not rated on the site? This article compares the pattern capabilities of BPEL and WLI 8.1. Four patterns are considered in detail:

- Cancel Case, supported by both
- Milestone, supported by neither
- Discriminator, supported only by WLI 8.1
- Interleaved Parallel Routing, supported only by BPEL

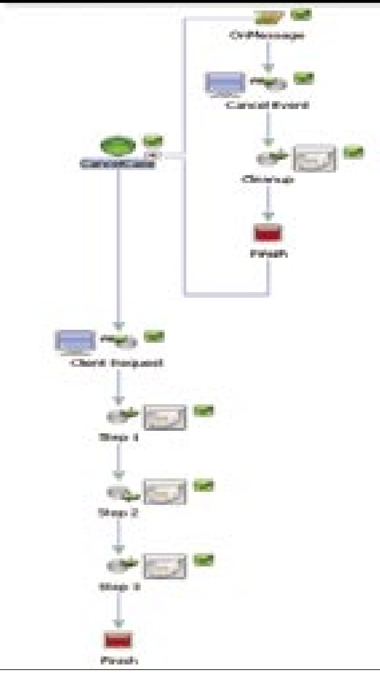
This article also presents a complete scorecard for WLI 8.1.

Cancel Case: Supported by Both

The intent of the Cancel Case pattern is to stop a running process, no matter where it is in its execution, on receipt of a cancellation event (e.g., rescind the processing of an insurance claim while it is under investigation). Most BPM vendor implementations offer a system-management function that allows administrators to abort errant processes, but this pattern requires that the cancellation logic, which may well have crucial business significance, be embedded into the process model itself.

For a process language to support Cancel Case, it must have the ability to listen for events in a path that is separate from the mainline path. WLI 8.1 supports this by allowing a message path to be associated with a process or activity. The process shown in Figure 1, for example, is always prepared to respond to a Cancel Event, regardless of how far (Step 1, Step 2, or Step 3) the main logic

FIGURE 1



WLI 8.1 Cancel Case implementation

Author Bio:
Michael Havey is an IBM consultant with 10 years of industry experience, mostly with application integration. Michael is currently writing a book on BPM, to be published by O'Reilly later this year.

Contact:
haveym@ca.ibm.com

THE WORLD'S LEADING INDEPENDENT WEBLOGIC DEVELOPER RESOURCE

ARE YOU A WEBLOGIC EXPERT? PAGE 4
NOV/DEC 2004
wldj
THE LEADING INDEPENDENT MAGAZINE FOR WEBLOGIC PROFESSIONALS
WWW.WLDJ.COM
Database Controls
Best Practices
AVOIDING POTENTIAL PITFALLS DURING THE DEVELOPMENT LIFE CYCLE ...8
PLUS...
Open Source Technologies
Enabling Next-Generation Portals
Security Best Practices
Strategies for WebLogic Domain Configuration
Creating WebLogic Domains in Silent Mode
A Real-World Business Process Model
TRANSACTIONS:
PRODUCT REVIEW:
PORTAL:

Helping you enable intercompany collaboration on a global scale

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

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

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

Now in More Than 5,000 Bookstores Worldwide

Go Online & Subscribe Today! www.WLDJ.com

SYS-CON.COM/WEBLOGIC/

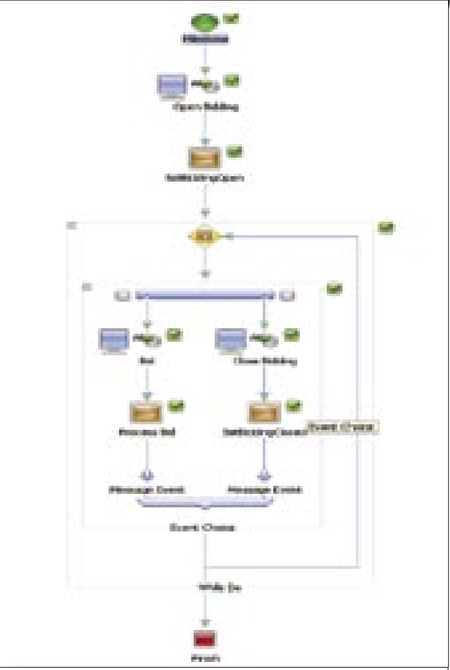
SYS-CON Media, the world's leading publisher of *i*-technology magazines for developers, software architects, and e-commerce professionals, brings you the most comprehensive coverage of WebLogic.

SYS-CON MEDIA

has progressed. When it receives the cancellation event, the process cleans up (Cleanup) and exits (Finish).

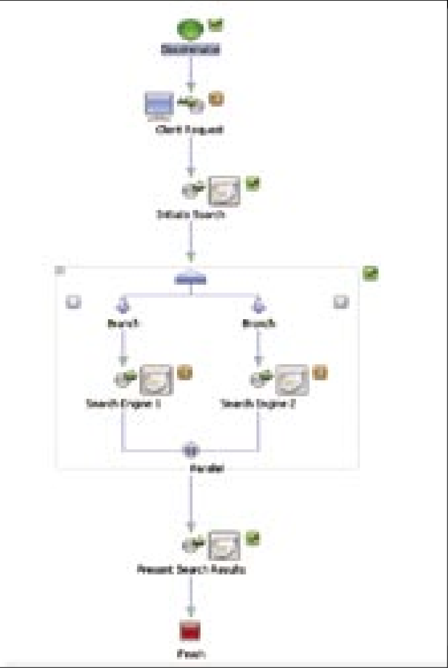
BPEL, which permits an event handler to be associated with a process or a child activity, has a similar implementation.

FIGURE 2



WLI 8.1 Milestone implementation

FIGURE 3



WLI 8.1 Discriminator implementation

In Listing 1, the process registers an event handler (lines 2-6) that on receipt of cancelEvent stops the process by terminating it. The handler can interrupt with main logic (lines 7-9) at any time.

(*Note:* In this article, BPEL samples are shown as XML-based source code, whereas WLI 8.1 examples are shown in the graphical representation adopted by WebLogic Workshop. BPEL does not have a standard graphical representation. WLI 8.1's PD4J does have an XML encoding, but most readers are more familiar and comfortable with Workshop's graphical view.)

Milestone: Supported by Neither

In the Milestone pattern an activity can be performed only when a certain milestone is reached and cannot be performed after the milestone expires. Put differently, an activity can run any number of times between the occurrences of enabling and disabling events. For example, in an auction, a bidder can place a bid any number of times between the start and end of bidding.

Few process languages support this distinctive pattern directly. In WLI 8.1, elaborate design is required, demonstrated by the bidding process shown in Figure 2. The process begins with the enabling event Open Bidding, after which it sets a Boolean flag to true (in the Perform node Set Bidding Open) to indicate that bidding is enabled. The heart of the process is a complicated while loop whose condition checks the Boolean flag to determine whether to continue; the loop runs for as long as the flag is set to true. The while loop contains an Event Choice with two event paths: one representing a bid (Bid), the other representing the disabling event Close Bidding. When Close Bidding occurs, the process calls a Perform node (Set Bidding Closed) to clear the flag, which will break the loop before its next iteration.

The logic successfully satisfies the requirement that the Bid event can occur any number of times until the Close Bidding event occurs. However the implementation is far too complicated to get a passing grade on the scorecard; the language lacks features that would make the implementation easy.

BPEL's implementation of this example, described in the paper "Pattern Based Analysis of BPEL4WS" (Reference #5), is

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

SUBSCRIBE TODAY TO MULTIPLE MAGAZINES

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

RECEIVE YOUR DIGITAL EDITION ACCESS CODE INSTANTLY WITH YOUR PAID SUBSCRIPTIONS

3-Pack

Pick any 3 of our magazines and save up to $210.00

Pay only $99 for a 1 year subscription plus a FREE CD

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

6-Pack

Pick any 6 of our magazines and save up to $340.00

Pay only $199 for a 1 year subscription plus 2 FREE CDs

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

9-Pack

Pick 9 of our magazines and save up to $270.00

Pay only $399 for a 1 year subscription plus 3 FREE CDs

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

CALL TODAY! 888-303-5282

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

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

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

☐ **LinuxWorld Magazine**

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

☐ **JDJ**

U.S. - Two Years (24) Cover: $144	You Pay: $99.99 /	Save: $45 + FREE $198 CD
U.S. - One Year (12) Cover: $72	You Pay: $69.99 /	Save: $12
Can/Mex - Two Years (24) $168	You Pay: $119.99 /	Save: $48 + FREE $198 CD
Can/Mex - One Year (12) $120	You Pay: $89.99 /	Save: $40
Int'l - Two Years (24) $216	You Pay: $176 /	Save: $40 + FREE $198 CD
Int'l - One Year (12) $108	You Pay: $99.99 /	Save: $8

☐ **Web Services Journal**

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

☐ **.NET Developer's Journal**

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

☐ **Information Storage + Security Journal**

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

☐ **Wireless Business & Technology**

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

☐ **MX Developer's Journal**

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

☐ **ColdFusion Developer's Journal**

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

☐ **WebSphere Journal**

U.S. - Two Years (24) Cover: $216	You Pay: $129.00 /	Save: $87 + FREE $198 CD
U.S. - One Year (12) Cover: $108	You Pay: $89.99 /	Save: $18
Can/Mex - Two Years (24) $240	You Pay: $159.99 /	Save: $80 + FREE $198 CD
Can/Mex - One Year (12) $120	You Pay: $99.99 /	Save: $20
Int'l - Two Years (24) $264	You Pay: $189.00 /	Save: $75
Int'l - One Year (12) $132	You Pay: $129.99 /	Save: $2

☐ **PowerBuilder Developer's Journal**

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

☐ **WLDJ**

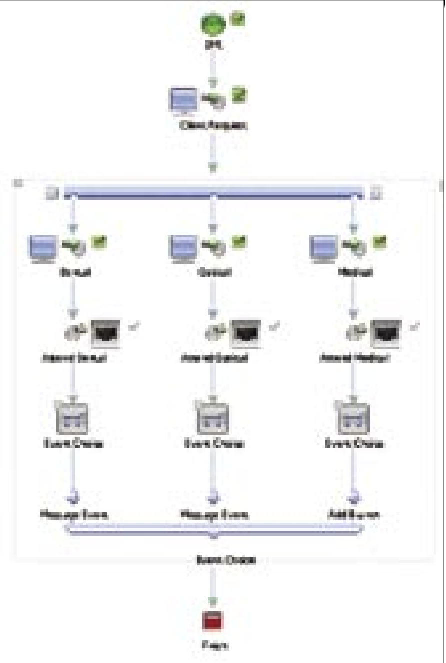
U.S. - Four Years (24) Cover: $240	You Pay: $99.99 /	Save: $140 + FREE $198 CD
U.S. - Two Year (12) Cover: $120	You Pay: $49.99 /	Save: $70
Can/Mex - Four Years (24) $240	You Pay: $99.99 /	Save: $140 + FREE $198 CD
Can/Mex - Two Year (12) $120	You Pay: $69.99 /	Save: $50
Int'l - Four Years (24) $240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

*WHILE SUPPILES LAST. OFFER SUBJECT TO CHANGE WITHOUT NOTICE

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

SYS-CON MEDIA

FIGURE 4



WLI 8.1 Interleaved Parallel Routing implementation

nearly identical, using a Boolean flag, a while loop, and a "pick" (BPEL's equivalent of an Event Choice) to model the milestone logic.

Discriminator: Supported by WLI 8.1

In the Discriminator pattern, when multiple parallel branches converge at a given join point, exactly one of the branches is allowed to continue on in the process, based on a condition evaluated at run time; the remaining branches are blocked. Discriminator is a special case of the N-of-M Join pattern, where M parallel branches meet at a point of convergence and only the first N are let through; in Discriminator, N=1.

The need for this pattern arises when only a subset of assigned work in required. For example:

- In order to obtain a security clearance, an applicant must demonstrate two of the following three criteria: good credit, no criminal record, and natural citizenship (*2-of-3*)
- A complicated Web search is submitted to two search engines; as soon as one engine completes, its results are collected and the other search is ignored (*1-of-2*, or *Discriminator*).

Interestingly, WLI 8.1 supports Discriminator out of the box, but has excessive difficulty with the more general N-of-M. The search engine example of Discriminator is shown in Figure 3. In separate branches in a parallel structure, a search is submitted to two engines (Search Engine 1 and Search Engine 2). The parallel structure is set with an OR join condition (shown as parallel bars in the circle at the bottom of the parallel structure), so that as soon as one engine completes, the parallel structure itself completes, and control is passed to the next activity, Present Search Results.

WLI 8.1 cannot easily model the 2-of-3 security clearance example, or any other N-of-M where N>1. The language is limited: the parallel join condition is not sophisticated enough to select, say, two branches. BPEL is no better off, lacking even basic OR join capability that could satisfy Discriminator. (BPEL has an OR join, but it waits for each inbound path to complete, whereas the WLI 8.1 join accepts the first inbound path and discards the others.) The easiest way to implement the search engine example in BPEL is to run the searches in separate processes (Listing 3) that perform the work (line 3) and publish an event to announce when they are done (line 4). The main process (Listing 2) spawns the subprocesses asynchronously (the "invoke" statements in lines 2 and 3) and waits for the first of the completion events to arrive (the "receive" in line 4).

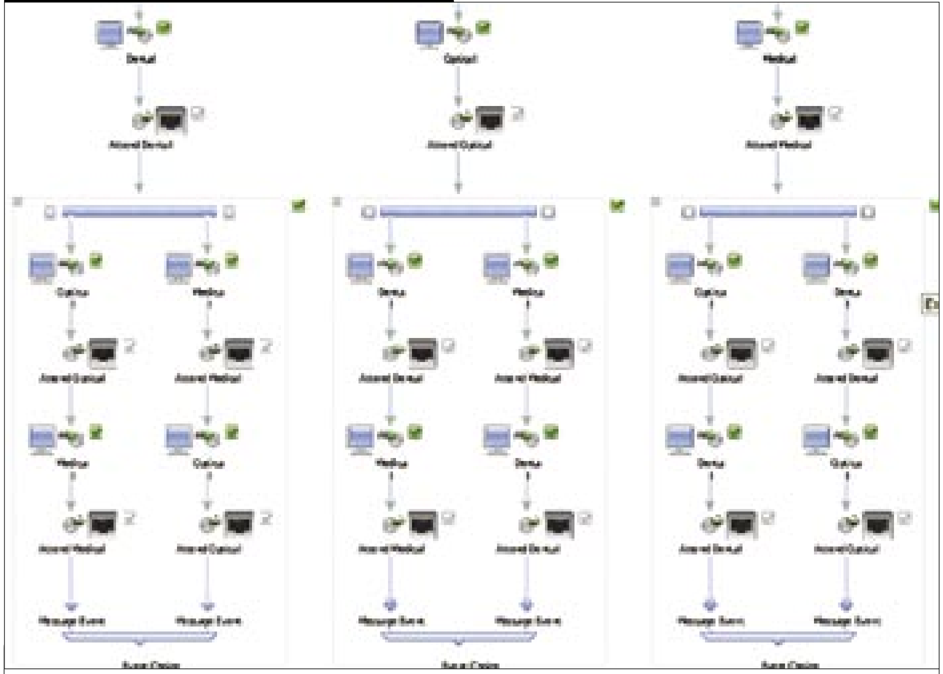
The BPEL implementation of the security clearance example, shown in Listing 4, is similar: spawn (lines 2-5) the three checks as subprocesses, and wait for any two to complete (lines 5-6). WLI 8.1 could follow this approach too, perhaps using Message Broker subscriptions to catch the events.

Interleaved Parallel Routing: Supported by BPEL Only

Interleaved Parallel Routing is an unusual pattern in which several activities are to be performed in an order that is indeterminate at design time. The activities are performed *in sequence* (not in parallel, as the name of the pattern suggests), but the order of execution is arbitrary. Interleaved Parallel Routing is a type of *ad hoc* processing, though *ad hoc* includes cases that are even more monstrous, where there is no clear-cut distinction between parallel and sequential, and execution is determined by the performing actors (e.g., the work of a software developer on a typical project, whose job might be to develop three components, contribute two sections to the architecture document, and set up a stand-alone test environment for the development group).

For a good example of Interleaved Parallel Routing, consider the case in which an applicant to the army must take three tests – dental, medical, and optical – one after the

FIGURE 5



WLI 8.1 Interleaved Parallel Routing expanded view

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

TABLE 1

Name	Description	WLI 8.1 Implementation	WLI 8.1 Score	BPEL Score
Sequence	Run activities sequentially. For example, run activity A, followed by B, followed by C, and so on.	Fundamental	+	+
Parallel Split	From a single activity branch, or fork, to multiple parallel paths.	Parallel control structure	+	+
Synchronization	Several parallel paths converge on a single activity, which waits for the completion of all paths before starting.	Parallel control structure with AND join condition	+	+
Exclusive Choice	From a single activity branch to exactly one of several paths based on the evaluation of a condition.	Conditional control structure	+	+
Simple Merge	Several exclusive conditional paths converge on a single activity, which starts executing when the one chosen path completes.	Conditional control completes when its selected path completes.	+	+
Multi-Choice	Choose one or more parallel branches, where each branch is taken only if it satisfies a particular condition.	Parallel control structure containing one conditional control structure for each branch. This approach is similar to the one used in Business Process Modeling Language (BPML). The BPEL approach is quite different.	+-	+
Synchronizing Merge	Join branches spawned by a multi-choice. In other words, wait for all of the active paths in a parallel structure to complete.	Parallel control structure containing conditional control structures finishes when all conditional paths have completed. Similar to BPML approach.	+-	+
Multi-Merge	Whereas synchronizing merge waits for all of incoming branches to complete before continuing, multi-merge allows each incoming branch to continue independently of the others, enabling multiple threads of execution through the remainder of the process. When concurrent activities A and B are multi-merged into C, the possible combinations of execution are ABCC, ACBC, BACC, and BCAC.	Hard. One approach is to place "C" on the parallel paths for both "A" and "B." If C is complicated, factor it into a separate process and call it on each of the A and B paths.	-	-
Discriminator and N-of-M Join	In the discriminator pattern, when multiple parallel branches converge at a given join point, exactly one of the branches is allowed to continue on in the process, based on a condition evaluated at run time; the remaining branches are blocked. Discriminator is a special case of the N-out-of-M join pattern, where M parallel branches meet at a point of convergence and only the first N are let through.	A parallel control structure with an OR join condition lets through the first of the paths to complete, which satisfies many cases of 1-of-M (Discriminator). The general case of N-of-M where N>1 is more complicated (discussed in detail in this article).	+-	-
Arbitrary Cycles	Repeat an activity or a set of activities by cycling back to it in the process.	Only structured loops, no arbitrary looping.	-	-
Implicit Termination	In most languages, a process, even if it spawns a complex tree of concurrent branches, has exactly one exit point into which all possible paths converge. The implicit termination pattern is a relaxation of the design rules: the process completes when the activities on each of its branches complete.	Similar to the BPML solution, the effect can be achieved by spawning multiple subprocesses asynchronously in a parallel control structure.	+	+
Multiple Instances (MI) Without Synchronization	Perform multiple concurrent instances of an activity, but let each run on its own with no overall synchronization.	Model the instance as a subprocess and call it asynchronously in a while loop from the main	+	+
MI with Design Time Knowledge	Perform N multiple concurrent instances of an activity, where N is a constant at run time. In addition, join these instances before continuing with the remainder of the process.	process. Run each instance in a separate path in a parallel control structure.	+	+
MI With Runtime Knowledge	Perform N multiple concurrent instances of an activity, where the value of N is known at runtime before the multiple instances loop is started. In addition, join these instances before continuing with the remainder of the process.	Model the instance as a subprocess and call it asynchronously in a while loop from the main process. Design the subprocess so that it publishes a message when it has completed. The main process contains a loop that waits for this event from each of its spawned processes.	-	-
MI Without Runtime Knowledge	Perform multiple concurrent instances of an activity, where the number of instances is not known until some point during the actual processing of the instances. In addition, join these instances before continuing with the remainder of the process.	Similar to MI With Runtime Knowledge, except additional logic is required to determine how many subprocesses to spawn.	-	-
Deferred Choice	Deferred choice is similar to exclusive choice, in that a decision is made to follow one of multiple paths, except the decision is not made immediately but is deferred until an event occurs.	Event choice control structure.	+	+
Interleaved Parallel Routing	Several activities are to be performed in sequence (not in parallel, as the name of the pattern suggests), but the order of execution is arbitrary and is not known at design time.	Hard. Design separate paths to account for each combination of activities. For example, if activities A, B, and C are to be interleaved, design paths for ABC, ACB, BAC, BCA, CAB, and CBA (discussed in detail in this article).	-	+-
Milestone	An activity can be performed only when a certain milestone is reached, and cannot be performed after the milestone expires.	Hard. One approach is to place the activity in a while loop, which continues until it encounters a disabling condition. The disabling condition is initially set to false, and becomes true only when a disabling event occurs (discussed in detail in this article).	-	-
Cancel Activity	Stop the execution of a particular process activity on a cancellation trigger.	Message path associated with the activity in question responds to "cancel" event. The message path can be configured on completion to skip to the next activity in the main logic of the process.	+	+
Cancel Case	Stop the execution of an entire process on a cancellation trigger.	Message path scoped at process level responds to "cancel" event. Have the message path conclude with a finish node (discussed in detail in this article).	+	+

WLI 8.1 pattern scorecard

other, but in any order; when the applicant completes one test, the next one to take is determined by the availability of doctors (example suggested in Reference #6, p.10).

The WLI 8.1 implementation is a combinatorial mess! The overall process, shown in Figure 4, starts with an Event Choice with separate paths for Dental, Optical, and Medical. The Dental event means that the applicant has been called in for a dental examination; the applicant responds by attending the examination (Attend Dental), before entering an inner Event Choice (shown collapsed in Figure 4), which models the remaining two appointments. The paths for medical and optical have the same form as dental. Figure 5 shows the expanded Event Choice structures for the three top-level choices. If the first choice had been dental, the next choice is between medical and optical. If called for medical, the applicant attends the medical appointment, and then waits to be called for the optical; if called for optical, the applicant attends the optical and waits for the medical.

In total, six paths are required to model the permutations of dental, medical, and optical. If there had been four activities, 24 paths would have been needed; if five activities, 124 paths. Messy indeed! Alternative implementations should be considered on a case-by-case basis.

BPEL, probably by accident rather than by design, has an elegant, linear solution. The idea is to place the activities in a parallel structure (the "flow" activity in lines 5-27 of Listing 5), but to force each activity to acquire a mutually exclusive, shared variable (defined in line 3) before running. The effect is that the activities run sequentially, but in indeterminate order. The key to this maneuver is the variableAccessSerializable flag in lines 6, 13, and 20, which ensures, in effect, that as soon as one of the scoped activities (optical in lines 6-12, dental in lines 13-19, medical in lines 20-26) accesses the variable, the others are locked out until the activity has completed.

In the BPEL approach the order of activities is effectively random, determined by the proprietary low-level logic of the BPEL engine. This approach is not suitable for cases in which order is to be driven by application logic or external stimuli.

WLI 8.1 Scorecard

Table 1 provides the scorecard for WLI 8.1 on the 20 patterns. Following the style of the workflowpatterns.com authors, the table uses the symbol "+" to indicate that the language fundamentally supports the given pattern, "+-" to indicate that the language supports the pattern with straightforward coding, and "-" to indicate that the pattern is either hard or impossible to implement in the language. Counting both + and +-, both WLI 8.1 and BPEL score 14.

Summary

BPEL and WLI 8.1 score differently on support for the 20 process patterns documented on the renowned site www.workflowpatterns.com. Some patterns are supported by both, some by neither, some by BPEL but not WLI 8.1, and some by WLI 8.1 but not by BPEL.

References

- www.workflowpatterns.com (this site also includes links to several key papers)
- http://e-docs.bea.com/wli/docs81/index.html
- T. Andrews, M. Curbera, et al. "Business Process Execution Language for Web Services," Version 1.1. www.oasis-open.org, May 2004. [Available at the following URLs: http://dev2dev.bea.com/technologies/webservices/BPEL4WS.jsp (BEA), www-106.ibm.com/developerworks/webservices/library/ws-bpel/ (IBM), http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnbiz2k2/html/bpel1-1.asp (Microsoft), http://ifr.sap.com/bpel4ws/ (SAP), www.siebel.com/bpel (Siebel).]
- Michael Havey, *Essential Business Process Modeling*, O'Reilly Media, 2005, Cambridge, MA. www.oreilly.com/catalog/essentialpm. Descriptions of the patterns in this article are based on material from this book.
- P. Wohed, W. van der Aalst, M. Dumas, A.H.M. ter Hofstede. "Pattern Based Analysis of BPEL4WS," FIT Technical Report, FIT-TR-2002-04.
- M. Dumas and A.H.M. ter Hofstede, "UML Activity Diagrams as a Workflow Language," in *Proceedings of the International Conference on the Unified Modeling Language*, Springer-Verlag, Toronto, Canada, October 2001.

Listing 1: BPEL Cancel Case Implementation

```
<process>
   <eventHandlers>
      <eventHandler name="cancelEvent" . . .>
         <terminate/>
      </eventHandler>
   </eventHandlers>
   <sequence>
      . . . <!-- main flow -->
   </sequence>
</process>
```

Listing 2: BPEL Discriminator Implementation

```
<process>
   <invoke name="Submit Search Engine 1" . . . />
   <invoke name="Submit Search Engine 2" . . . />
   <receive name="Search Complete" . . ./>
   <!-- got one, proceed -->
```

Listing 3: BPEL Discriminator Subprocess

```
<process>
   <receive name="Get Search Request" . . . />
   <!-- perform the search -->
   <invoke name="Search Complete" . . ./>
</process>
```

Listing 4: BPEL N-of-M Join Implementation

```
<process>
   <invoke name="Check credit" . . . />
   <invoke name="Check criminal record" . . . />
   <invoke name="Check natural citizen" . . . />
   <receive name="Clearance Event" . . ./>
   <receive name="Clearance Event" . . ./>
   <!-- got two of three, proceed with the clearance -->
```

Listing 5: BPEL Interleaved Parallel Routing Implementation

```
01 <process>
02    <variables>
03       <variable name="C" . . . />
04    </variable>
05    <flow>
06       <scope name="optical" variableAccessSerializable="yes">
07          <sequence>
08             write to variable C
09             run optical activity
10             write to variable C
11          </sequence>
12       </scope>
13       <scope name="dental" variableAccessSerializable="yes">
14          <sequence>
15             write to variable C
16             run dental activity
17             write to variable C
18          </sequence>
19       </scope>
20       <scope name="medical" variableAccessSerializable="yes">
21          <sequence>
22             write to variable C
23             run medical activity
24             write to variable C
25          </sequence>
26       </scope>
27    </flow>
28 </process>
```

Custom Debugging with WebLogic JMX

TRACE YOUR JDBC CALLS WITHOUT MODIFYING YOUR EXISTING CODE

By Salma Saad

Maintaining complicated legacy applications is a challenge, which is often made worse by lack of documentation, nonintuitive design, and coding practices. Unfortunately almost all software developers will find themselves with such an assignment at some point in their careers.

In the case of any application that utilizes a database, it is very useful to trace SQL statements generated by the application. Such a trace would help with profiling performance bottlenecks, debugging errors, and in facilitating the developer's understanding of the business processes associated with the application.

In the case of legacy applications we would want to do such tracing without changing any code or application configuration. I was able to use WebLogic's JMX API to quickly put together a little code to trace the JDBC calls of a very large and complicated legacy application without impacting the code and configuration of the application. In addition, this small project helped facilitate my understanding of JMX and how WebLogic uses JMX behind the scenes. In this article I will go over the details of using WebLogic JMX to trace SQL statements.

What is JMX?

JMX stands for Java Management Extensions. MBeans, or *managed beans*, are resources that can be managed through the JMX API. Most application servers use JMX to provide administration consoles and to manage resources. In addition, application developers can use JMX to provide administration and auditing capabilities in their custom applications.

What Advantages Does WebLogic's JMX Implementation Provide for Developers and Administrators?

WebLogic Servers use JMX MBeans for configuration and management. A WebLogic Server each has a copy of its own MBeans, which is updated by the administration server. The administration server maintains a master copy of the MBeans for all of the servers that it manages. If the administration server fails to come up, the managed servers can function based on their local copy of the MBeans until the administration server can become available to update the server's local MBeans.

Web Logic provides both an administration console, which does its work using JMX MBeans, as well as an API to allow application developers to configure and explore WebLogic resources. The easiest way to take advantage of WebLogic JMX is to use the WebLogic console to change the configuration of WebLogic resources and to view the metrics available in the console. While the monitoring and configuration capabilities of the WebLogic console are very powerful and will meet the needs of most applications running on WebLogic, the WebLogic JMX API provides an even more powerful instrument for managing applications running on the WebLogic platform. Using the WebLogic JMX API it is possible to configure and extend WebLogic resources and to also receive notifications from WebLogic's subsystems. For

Author Bio:
Salma Saad works for International Survey Research and is responsible for ISR's sophisticated, high availability global survey reporting Web site. She has almost a decade of experience in Java and related technologies.

ssaad@ecascabel.com

example, an application that is configured with *n* as the value for minimum and maximum number of JDBC connections might want to have a listener that listens to notifications from the WebLogic JMX MBeans and sends e-mail to an administrator in case the usage of the application crosses *n-x* concurrent JDBC connections, so that the administrator can decide on an increased value for *n* and reconfigure the JDBC connection pool(s) (where *x* is an arbitrary number that depends on the comfort level of the administrator). Examples of advanced uses of JMX by application developers include tracing events in the WebLogic subsystems, including EJB events and server start/stop events.

What Options Are Available for Profiling JDBC Statements in WebLogic Applications?

There are several techniques that can be used to create a dynamic trace of JDBC statements in a WebLogic application. Subclassing the Statement,PreparedStatement and CallableStatement classes from the java.sql package to print a trace using a logging system like Log4J or WebLogic logging and then using the subclasses in the application is a viable option, but it is not practical in the case of legacy code. Such traces may be available from tools such as TOAD, but such tools may not be easily available to application developers and may not provide all of the information that is required. AOP techniques are another valid choice for printing JDBC statements. However, at the time of writing AOP is not officially supported in WebLogic by BEA even though articles about WebLogic AOP have appeared on the dev2dev site. At the time of writing, getting AOP to work in WebLogic is not a trivial project. Using WebLogic JMX with WebLogic 6.1 and 8.1 does not require the use of any additional libraries or configuration, since all the required classes are available in weblogic.jar and the code is very simple to implement. In addition WebLogic JMX is a very mature technology and can be implemented without any changes to the core application code or bytecode.

Using the WebLogic JMX API

The WebLogic javadocs are available online at http://e-docs.bea.com/wls/docs81/javadocs/. The API contains several packages with *management* in their names. These packages are WebLogic's JMX implementation (see Table 1).

TABLE 1

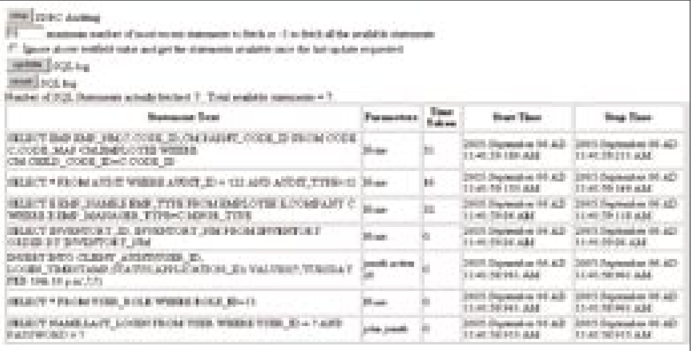
Package Name	Package Functionality
weblogic.management	This package contains classes, which manage MBeans across a domain (a list of domain-wide MBeans can be obtained from here)
weblogic.management.configuration	This package contains classes, which allow configuration of the sub subsystems in a WebLogic domain (for example, turning on SQL statement tracing)
weblogic.management.console.extensibility	Use this package if you would like to extend the WebLogic console
weblogic.management.deploy	Use this package to programmatically deploy server instances and clusters in a domain
weblogic.management.logging	This package is useful if you are interested in getting notifications about logging events
weblogic.management.runtime	This package is where you will be able to get information about run-time events such as JDBC statement execution
weblogic.management.security weblogic.management.security.audit weblogic.management.security.authentication weblogic.management.security.authorization weblogic.management.security.credentials weblogic.management.security.pk	These packages allows for configuration and management of security providers
weblogic.management.timer	This package is used to send timed notifications from the WebLogic subsystems
weblogic.management.utils	This package contains utilities that are useful for writing custom JMX administration code, including code to interface with an external LDAP server

WebLogic's JMX implementation

Using JMX to Trace JDBC Calls

A simple way to organize the tracing code and provide a UI to view the SQL is to write a JSP, a servlet, and a Java Bean or object. We'll go through the details of the bean/POJO here and leave most of the details of the UI/controller aspect out since most WebLogic developers already understand these very well. Note that you need not modify any deployment descriptors, database connection pools, or data sources in order to get the tracing to work. All changes to the needed application will happen at run time.

FIGURE 1



Sample output from a trace using WebLogic JMX

Step 1

First we'll create a class called *MyTracerBean.java* and import the WebLogic JMX packages and classes that we'll need.

```
import javax.naming.Context;
import weblogic.jndi.Environment;
import weblogic.management.MBeanHome;
import weblogic.management.configuration.JDBCConnectionPoolMBean;
import weblogic.management.runtime.JDBCStatementProfile;
import weblogic.management.runtime.JDBCConnectionPoolRuntimeMBean;
import javax.management.InstanceNotFoundException;
import javax.management.InvalidAttributeValueException;
import javax.naming.NamingException;
```

These classes are in the weblogic.jar so you won't need to add any JARs or classes to the WebLogic classpath.

Step 2

Next we'll write a method to get the MBeanHome.

```
private MBeanHome getMBeanHome() {
```

```
//URL to the serve whose JDBC activity we are tracing
String url = "t3://localhost:7001";
          String username = "mywlconsoleuname";
          String password = "mywlconsolepsswd";
//The MBeanHome will allow us to
//retrieve the MBeans related to JDBC statement tracing
MBeanHome home = null;

try { //We'll need the environment so that we can //retrieve the initial
context
          Environment env = new Environment();
          env.setProviderUrl(url);
          env.setSecurityPrincipal(username);
          env.setSecurityCredentials(password
                    Context ctx = env.getInitialContext();
//Retrieving the MBeanHome interface for the server with //the url t3://
localhost:7001
            home =(MBeanHome)ctx.lookup(MBeanHome.LOCAL_JNDI_NAME);
          } catch (NamingException ne) {
                    System.out.println("Error getting MBeanHome " + ne);
          }
          return home;
}
```

The code above will work for the simplest situation where the admin server also hosts the JDBC application we're tracing, but for the situation where the admin server is independent from the managed server(s) and several independent JVMs are involved we'll need to get the admin MBeans home instead of the local MBeans home. The difference between the two is that the local home provides the MBeans for a single server while the admin home provides MBeans for all of the servers, which are administered by an administration server. To get the admin MBeans home instead of the local MBeans home, replace LOCAL_JNDI_NAME with ADMIN_JNDI_NAME in the code above.

Step 3

It's useful if you provide a way to turn the JDBC profiling on and off because profiling is expensive and you should turn it off if you don't need it. The profiling is off by default so you'll have to turn it on before you can trace any JDBC statements. Create a method such as the following:

```
  public void configureJDBCAuditing(boolean isOn) {
          try {
                    MBeanHome home = getMBeanHome();
                    //Retreive the bean to help us configure the Pool
JDBCConnectionPoolMBean mConfigBean = (JDBCConnectionPoolMBean)home.
getConfigurationMBean("MyPool","JDBCConnectionPoolConfig");
                    mConfigBean.setSqlStmtProfilingEnabled(isOn);
                    mConfigBean.setSqlStmtParamLoggingEnabled(isOn);
          } catch (InvalidAttributeValueException iave) {
System.out.println("Invalid attribute while configuring tracing " + iave);
          } catch (InstanceNotFoundException infe) {
System.out.println("Instance not found while configuring tracing " +
infe);
                    }
  }
```

In the code above we are also telling the connection pool that we want to see the parameters going into the SQL statements. This adds to the overhead of the tracing but provides us with some valuable information.

Step 4

After we have configured the JDBC pool to store the profiles we can go ahead and ask for them. Remember that the number of profiles retrieved will be equal to the number of SQL statements executed since you turned the profiling on and not equal to all of the SQL statements that have ever been executed by the MyPool ConnectionPool. The code below retrieves the profiles; the parameter *maxProfiles* indicates how many of the most recent profiles should be fetched. Create a method such as:

```
  /** Pass in -1 to get all profiles */
  public JDBCStatementProfile[] getProfiles(int maxProfiles) {
          JDBCStatementProfile[] profiles = null;
          try {
                    MBeanHome home = getMBeanHome();
JDBCConnectionPoolRuntimeMBean mbean =             (JDBCConnectionPoolRu
ntimeMBean)home.getRuntimeMBean("MyPool ","JDBCConnectionPoolRuntime");
                    int numProfiles = mbean.getStatementProfileCount();
                    int profilesIndex = 0;
                    //figure out index to start at and how many we want
                    if (maxProfiles != -1) {
                              profilesIndex = numProfiles - maxProfiles;
}else {
                              maxProfiles = numProfiles;
}
profiles =mbean.getStatementProfiles(profilesIndex,maxProfiles);
          }  catch (InstanceNotFoundException infe) {
System.out.println("Problem retrieving jdbc profiles " + infe);
          }

          return profiles;
  }
```

The method **getStatementProfiles** of the JDBCConnectionPoolRuntimeMBean is missing from the WebLogic 8.1 API documentation, though it appears in the documentation for WebLogic 6.1. However, this appears to be a mistake since the method is available in 8.1 and a bug was fixed in this method in WebLogic 8.1 (CR094729 at http://e-docs.bea.com/wls/docs81/notes/resolved_sp01.html), which means that the method was meant to be included with WebLogic 8.1

Step 5

You can add a reset function so that you can clear the statement cache when you restart the server:

```
  public void reset() {
          MBeanHome home = getMBeanHome();
          try {
JDBCConnectionPoolRuntimeMBean mbean = (JDBCConnectionPoolRuntimeMBean)ho
me.getRuntimeMBean("MyPool ","JDBCConnectionPoolRuntime");
//Remove everything from the cache
                    mbean.resetStatementProfile();

          } catch (InstanceNotFoundException infe) {
```

```
System.out.println("Problem while resetting JDBC profiles " + infe);
          }
   }
```

Step 6

Iterate through the profiles to get the information you'd like to show (see Listing 1). This code would probably reside in the UI layer, such as in a JSP.

The Future of WebLogic JMX

WebLogic 9.0, which is a very recent release at the time of writing, supports JMX 1.2 instead of JMX 1.0, which was supported until WebLogic 8.1. In response to the change in the JMX specification, the WebLogic JMX API has changed quite a bit in 9.0 and the code in Listing 1 may generate deprecation warnings. When upgrading to 9.0 the JDBCDataSourceRuntimeMBean should replace the JDBCConnectionPoolRuntimeMBean. However, the vast majorities of legacy applications that are running on WebLogic are not using the 9.0 WebLogic Server at the time of writing, and probably will not do so for quite some time.

Listing 1

```
MyTracerBean myTracer = new MyTracerBean();
//In this case we want the 100 most recently executed statements
JDBCStatementProfile[] profiles = myTracer.getProfiles(100);
//Doing the looping so that the most recent statements information
is
//retrieved first
for (int i=profiles.length-1;i>-1;i--) {
//Getting the number of parameters passed into the current //state-
ment
int paramCount = profiles[i].getParameterCount();
   //Format the start and end time for the current statement
   SimpleDateFormat simpleDateFormat =
   new SimpleDateFormat("yyyyy.MMMMM.dd GGG hh:mm:ss:SS aaa");
String startTime=simpleDateFormat.format(new Date(profiles[i].get-
StartTime()));
String endTime=simpleDateFormat.format(new Date(profiles[i].getStop-
Time()));

//Append the parameters together in order to display them
StringBuffer paramsBuffer = new StringBuffer();
   if (paramCount < 1) {
      paramsBuffer.append("None");
   } else {
for (int j=0;j<paramCount;j++) {
         paramsBuffer.append(profiles[i].getParameter(j));
         paramsBuffer.append(" ");
         }
   }

   String statementTxt = profiles[i].getStatementText();
String paramsTxt = paramsBuffer.toString();
String timeTaken = profiles[i].getTimeTaken()

//Then use statementTxt, paramsTxt, timeTaken, startTime, endTime //
etc to show the statement details in a UI
}
```

Performance

KEEPING YOUR BOTTLENECKS FOR THE CANDLES

By Peter Holditch

"High performance" is what everybody strives for when putting together a new system. Technical folk often spend hours hung up on the raw speed of their code, and a certain machismo can be derived from shaving milliseconds off that pesky transaction that is the latest pride and joy. Often, this time is not very well spent.

In the pursuit of raw speed, not only does readable code often get substituted for the obscure (usually to no avail – optimizing compilers are pretty good these days), thus causing a maintenance headache in the future, but in many cases the performance metric being optimized is a pretty uninteresting one in the overall scheme of things. Many people develop a blinkered focus on the amount of time it takes for one request to do a round-trip – "That debit transaction just took 20ms to debit £500 from a savings account, whoopee!!"

For a transactional system, the throughput of the system is generally far more interesting than the absolute speed of a request (although, that said, there is mostly some constraint that must be met around the time it takes to execute transactions). Throughput is a measure of the amount of work that can be driven through the system while achieving the targets for response time. More work, of course, is partly driven by more clients requesting transactions in the system. The bonus amusement factor here is that the response time and the number of clients are not independent variables – the more clients throwing work at the system, the more likely it becomes that the individual transactions will take longer. Thus, the maximum throughput of a system governs (for a given deployment environment) how many clients can throw transactions into the system at some desired rate, keeping the response time of some fraction of the transactions (say, 90 percent) within the required limit. Having arrived at this conclusion, hours of fun ensue changing various system parameters (number of execute threads, number of database connections, number of machines, etc.) in attempting to predict what the production setup needs to look like to meet the required service levels and to optimize use of server resources.

In fact, it should not be at all surprising to anyone who is using an application server that the individual round-trip time of a single transaction is of little interest. Clearly, the way to optimize that is to have as short a code path as possible between the client and the back end it affects – sticking layers of application server infrastructure between the client and the back end is clearly not going to shorten the codepath. It does, however, improve the throughput. It does this mainly by sharing scarce server-side resources (database connections, threads, etc.) among the population of clients. With the simple-minded short codepath route to performance, you will end up with a one-to-one relationship between the number of clients and the resources consumed on the server, and once all the server's resources are consumed, you have reached a performance bottleneck and the throughput can be increased no further until you can scrape together some more resources to share out. What the application server is doing in effect is economizing the number of resources consumed on the server side by sharing them out among a population of clients (at the expense of a longer codepath, implying marginally increased transaction round-trip time), thereby increasing the maximum possible throughput of the system. At this point, everyone's favorite knee-jerk reaction: "I don't use transactions, they slow things down" can be usefully reviewed.

By now we have seen a system rushing as many requests as possible through a limited set of server resources. It goes without saying that all of these requests must achieve the correct results – that is, the result of any one of the requests executing on its own should be the same if the request is executed in parallel with lots of other requests (or the requests should be isolated, to use the parlance and yes, isolation is the "I" in ACID, and transactions are what give you ACID properties). So the cost of using XA transactions is a little absolute performance, but the benefit is that the results of your transactions are correct. You can rest easy at night knowing that

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

withdrawals from your bank account aren't going to go screwy under some obscure loading conditions IF your bank is using XA!

This discussion takes place completely at the data level – XA is all about database transactions. Each transaction locks the data in the database until the transaction is complete, whereupon the results become visible to the waiting world. Transactions cause bottlenecks when multiple requests are trying to access the same data – all but the first locker of the contended data get blocked or thrown out, which clearly hurts throughput since work is being done that will not result in a successful transaction flowing through the system.

However data in the database is not the only shared resource in a transaction system. I already talked about application servers being resource sharing mechanisms – any of these resources could be contended for, so the application server itself needs to lock in-memory data structures to avoid problems if they are accessed on behalf of multiple requests in parallel, and contention can result here too. Of course, layers other than the app. Server are resource sharing too – on a typical server, those 60 execute threads you configured for WebLogic are probably being executed across a handful of CPUs at the most – when contention for CPUs occurs, the unlucky threads must wait
in line until a CPU is free and so on with memory, disk, etc.

Overall, everything I have thus far mentioned represents a massive headache – the development of the performance tests, the running of the tests while all the tuning is done, the allocation of server resources to meet the predicted demands, and all of this is repeated with changing application releases and server environment releases and the resulting sized systems then have to stand up to unpredictable real-life loading. Not only is this a headache, it is a major lifetime cost of every application.

The astute among you will have noticed that my e-mail address has changed; that's because I have joined Azul Systems, who have a unique solution to these issues. Java is highly multithreaded, so it provides systems that contain many CPUs, each with many cores so those Java threads really result in parallelization. Then it provides support for optimistic locking on synchronized Java objects, to reduce contention within the Java software layer, then it provides pools of memory that can be shared between multiple applications. In short, the Azul Appliance provides a Java execution engine that dwarfs your systems' capacity to consume resources by overwhelming your individual applications with CPU and memory resources in order to try and alleviate any bottlenecks caused by the shortage of either in the traditional VM environment.

This tactic has the additional benefit that it more or less removes the need to do tuning on a per-application basis, since the set of resources provided by the appliance is so big that it can be readily shared among multiple applications that previously each required headroom on their own servers for peaks in demand. This sharing is possible because on aggregate the demand peaks will be smoothed (since it is improbable that many unrelated applications will all hit a demand peak at once), so demand can be met from the pooled resources, which leaves you free not only from application-level sizing (and the associated costs), but also from large-scale overprovisioning of hardware on a per-application basis too. All this heralds a new era of cheaply procured and managed appliances to run highly performant Java applications. For more details, visit www.azulsystems.com. That's all for this month; watch out for a repeat performance next month.

SSA

HP'S FRAMEWORK FOR CREATING ENTERPRISE WEB SERVICES

By Anjali Anagol-Subbarao and Sankar Ram Sundaresan

Author Bio:
Anjali Anagol-Subbarao works in HP's IT organization as an IT architect. She has 12 years of IT experience, the last five in Web services. Her book on J2EE Web services on BEA WebLogic was published in October 2004.

Sankar Ram Sundaresan is the chief architect of HP's E-Business IT organization. Based in Palo Alto, California, Sankar is responsible for formulating and implementing the E-Business architecture and technology strategy in support of HP's business objectives. Over the past three years, Sankar has been instrumental in transforming HP's E-Business IT organization to a services-oriented architecture.

Contact:
anjali.anagol-subbarao@hp.com
sankar-ram.sundaresan@hp.com.

The potential and promise of service-oriented architecture (SOA) can be realized by designing, developing, and deploying well-designed, flexible, extensible, and scalable Web services, Web applications, and portlets.

Reusable enterprise class Web services encapsulate business logic and drive business processes. A variety of clients consume these Web services without worrying about specific implementation technologies. The user interface, which is responsible for driving dynamic user experience, is most often Web based. These user interfaces serve the needs of several audiences and invoke the enterprise Web services on an as-needed basis. With Web services, the usage patterns tend to ramp up much more quickly, and the concurrent loads tend to be much higher than traditional applications or services. To successfully make the transition to an SOA, the Web services need to be designed and developed using proven design patterns and software engineering best practices. SOAP toolkits alone are not sufficient.

Web services are a standards-based way of describing, discovering, and invoking services. SOAP toolkits and Web services standards offer very little to tackle the gory details of *actually implementing* a useful piece of business functionality as a reliable, scalable, and high-performance service. This is where frameworks step in.

Frameworks and Web Services

A framework is a reusable, "semi-complete" (skeleton) application that is designed to be extended to build specific applications or services. A framework typically defines an overall flow, and invokes developer-provided components at predefined points in the flow using predefined interfaces. Developers develop business components and then "plug" them into the framework to create a functional application or service. Frameworks are different from utility classes or libraries, although frameworks often include several utility classes or libraries. The key differentiating factor is that frameworks invoke developer-written code, rather than the other way around.

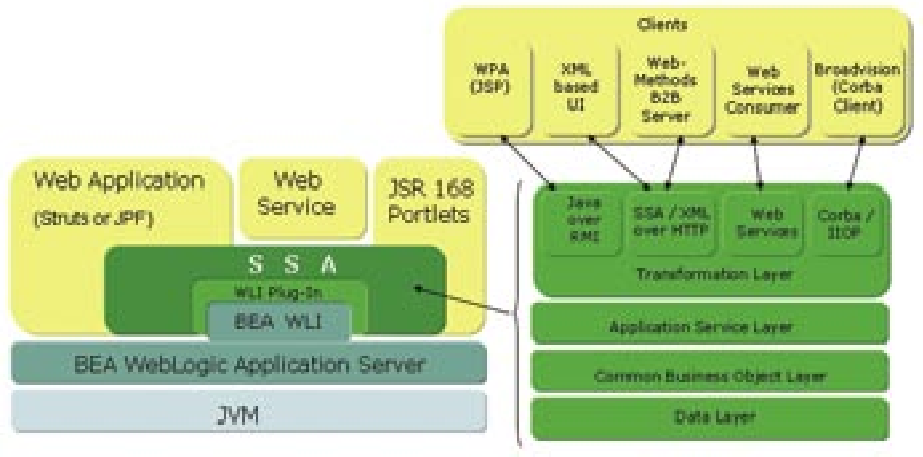
While computing power and network bandwidth has increased dramatically over the past few years, so has the complexity of developing applications. A lot of the cost and complexity of software design and development stems from the continuous rediscovery and reinvention of core concepts. Frameworks minimize this complexity by providing development teams with a much better starting point, rather than having to start from scratch. Frameworks catalyze and support the move to an SOA by helping to accelerate the creation of well-designed, reusable services in a consistent and repeatable manner.

Developers benefit from improved productivity as a result of using a framework that already incorporates design patterns and best practices. Developers utilize off-the-shelf features of the frameworks and write less code. They don't need to understand the nuts and bolts of J2EE standards and specifications. They don't need to be experts at object-oriented design and design patterns to benefit from using them. Frameworks help developers deal with the complexities of the actual execution of a Web service that would later be invoked by a SOAP toolkit.

At a corporate level, frameworks help in getting to an SOA quickly and at a lower cost. Corporations benefit from consistency of design and development across projects. The combination of repeatability and the ability to guarantee a minimal level of architecture and design rigor minimizes the development cost for future projects. Corporations also benefit from improved business agility as a result of having modular solutions that can be changed easily (often via configuration changes). In essence, frameworks facilitate the use of software engineering best practices among developers with varying skill levels, and promote the creation of consistent, predictable, and better-tested solutions.

SSA complements Web service toolkits by offering

FIGURE 1



SSA's layered approach

Web service developers a solid, extensible foundation for creating J2EE-based Web services. While SOAP toolkits focus on turning SOAP invocations into corresponding Java invocations and Java responses into SOAP responses, SSA focuses on offering developers a robust foundation for dealing with the complexities of developing enterprise-class, mission-critical Web services. SSA supports the WebLogic Workshop SOAP toolkit and Apache Axis.

Shared Services Architecture (SSA)

The shared services architecture (SSA) is a framework developed in HP for designing and developing well-designed, modularized, reusable, scalable, and easily maintainable Web services and other enterprise services using Java and J2EE. The first version of the SSA was released in the year 2000, and support for creating Web services was added in 2002. SSA is developed and maintained by a small team within HP and is the corporate standard framework for all internal Java-based Web services development. SSA is currently being used by several large projects for mission-critical services and applications that handle millions of transactions per day.

IT organizations are aggressively moving away from monolithic solutions and toward creating Web services that encapsulate and expose key business processes. SSA provides a set of higher-level programming abstractions (Request, Result, Feature, Business Policy) and a superior starting point for creating enterprise-class services by incorporating several design patterns and time-honored software engineering best practices. By combining a layered architecture (as shown in Figure 1), ease of use, and a deep emphasis on good architecture and reuse, SSA offers an ideal foundation to create enterprise-class, mission-critical Web services in a vendor-neutral and portable manner. SSA is supported on multiple versions of BEA WebLogic and JBoss application servers.

SSA also provides automated metrics gathering and APIs for logging and environment and life-cycle management. The metrics automatically collected by SSA include business metrics, performance metrics, and error information. These metrics can be persisted to any destination of choice (file system, database etc.) in any format. SSA also provides tools, including development tools such as code generation wizards and deployment tools such as scripts, Ant build files, etc. A wealth of enterprise data still resides in back-end legacy systems. SSA recognizes this fact and provides built-in support for efficient and reliable access to legacy systems.

SSA Epitomizes Software Engineering Best Practices

SSA has made extensive use of the Gang of Four (GOF) and J2EE design patterns and other best practices to specify an architectural blueprint for server-side logic components. SSA separates business logic from protocol and presentation logic and is designed in a simple, flexible, and extensible manner.

SSA offers a rich set of features for implementing Web services based on SOA principles, design patterns, and best practices, including:

- A robust framework that incorporates inversion of control and a well-defined flow
- A layered architecture that offers a high degree of modularity
- Ability to implement several business policies at multiple levels
- Ability to change service behavior through configuration changes
- Automated metrics gathering and reporting
- Extensive support for exception and error handling
- Ability to service new protocols with no change to core service code
- Server-side session state that survives app server crashes and restarts
- Support for accessing legacy systems

The SSA framework provides a skeleton for building shared services using J2EE. Developers install, develop, and run their services on the SSA framework. The framework specifies and controls the overall flow and calls service components at predefined points in time using predefined interfaces.

The SSA architecture complements the framework and describes the overall structure of a shared service, and provides a design blueprint for organizing a service, including the logical layers that a service is recommended to have as well as the responsibilities of those layers.

BEA WebLogic and SSA

The BEA WebLogic application server is the preferred environment for deploying the SSA framework and the services developed using SSA. Several mission-critical SSA-based services run on the BEA WebLogic server and handle hundreds of concurrent transactions. Web services developed on SSA can be exposed using the WebLogic SOAP toolkit.

SSA's WLI 8.1 plug-in offers two-way connectivity from an SSA-based service to WebLogic Integration workflows, and vice versa. An SSA-based service can invoke a WLI workflow or orchestration as part of its business logic. In addition, SSA-based services can be participants in a workflow or process that is orchestrated by WLI. The WLI plug-in offers a custom SSA control that can be inserted into a WLI workflow, and this control will route requests to an instance of the SSA framework.

The next version of SSA will leverage the Apache Beehive framework and will support Beehive controls and declarative programming using JSR-175 annotations. This will improve developer productivity by allowing for more visually driven programming and asynchronous and re-entrant pipelines.

Business Benefits of Using the SSA Framework

The use of frameworks is fast becoming an industry-wide best practice because of the reuse and leverage they make possible. From an application-development perspective, SSA focuses on tasks that span the entire development life cycle, from design to development to ongoing management, and provides a complete baseline framework that includes facilities to handle commonly sought-after functionality in a vendor-neutral fashion. Application development teams and IT organizations benefit on several fronts by using SSA to create enterprise class services, including faster time to market, reduced total cost of ownership (TCO), reduced support costs, improved developer productivity, and improved business agility.

Summary

Service orientation has become the key architectural driver that is guiding a lot of business and architectural decisions. Frameworks provide a great, cost-effective foundation for building services, and play a pivotal role in helping to realize the benefits of an SOA. Frameworks and Web services toolkits complement each other.

The SSA framework codifies the use of best practices, including design patterns, into a ready-to-use architecture and offers a robust, flexible framework for implementing modular, reusable, high performance Web services. SSA complements standard J2EE Web service toolkits, and helps HP IT in creating a consistent, repeatable approach to developing Web services in J2EE.

SSA, combined with BEA WebLogic Application Server and BEA WebLogic Integration, provides a platform for developing reliable and scalable Web services based upon software engineering best practices. SSA has allowed several IT organizations in HP to jump-start their SOA transformation by allowing developers to easily and rapidly create mission-critical Web services.

Run-Time Management of WebLogic Messaging Services

– continued from page 8

Whatever your usage of enterprise messaging is, you need to prepare for complications. If you have strict requirements for data integrity, make sure you have proper monitoring in place and that you have a carefully thought out strategy for handling messaging problems, such as message processing failures. Appropriate tools are readily available and they are easy to use, so there is no excuse for being unprepared when you find yourself in a messaging mess.

References

- Yochem, A., Carlson, D., and Stephens, T. (2004). J2EE Applications and BEA WebLogic Server. Prentice-Hall.
- Programming WebLogic JMS: http://e-docs.bea.com/wls/docs90/jms/index.html
- Developing Custom Management Utilities with JMX: http://e-docs.bea.com/wls/docs90/jmx/index.html

Listing 1: Browsing Through the Contents of a Queue

```
public Enumeration getQueuedMessages(
  javax.jms.QueueSession qSession,
  javax.jms.Queue queue)
  throws javax.jms.JMSException
{
  javax.jms.QueueBrowser browser =
     qSession.createBrowser(queue);
  return browser.getEnumeration();
}
```

Listing 2: Getting a Queued Message Without Consuming It

```
public javax.jms.Message getMessageFromQueue(
  javax.jms.QueueSession qSession,
  javax.jms.Queue queue ,String messageId)
  throws javax.jms.JMSException
{
  javax.jms.QueueBrowser browser =
     qSession.createBrowser(
        queue, "JMSMessageID = '" + messageId + "'");

  return
      (javax.jms.Message)
      browser.getEnumeration().nextElement();
}
```

Listing 3: Deleting a Message from a Queue by Consuming It

```
public void deleteMessageFromQueue(
  javax.jms.QueueConnection qConnection,
  javax.jms.QueueSession qSession,
  javax.jms.Queue queue, String messageId)
  throws javax.jms.JMSException
{
  javax.jms.QueueReceiver qReceiver =
      qSession.createReceiver(
          queue, "JMSMessageID = '" + messageId + "'");

  qConnection.start();
  qReceiver.receive(1000);
}
```

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

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

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

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

netIQ
Work Smarter.

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

No more endless hours of sorting out server problems. AutoPilot ™ from Arcturus Technologies, automatically does that for you.

It's an affordable new concept that eliminates your 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.
- Cost Effective at only $999 per IP address, regardless the number of CPUs

Want to know more & place an order ?
Log on to **www.arcturustech.com**

AUTOPILOT™
AUTO OPTIMIZER