PREMIER ISSUE

FOR DEVELOPERS • BY DEVELOPERS

BEA WebLogic DEVELOPER'S JOURNAL

Volume:1 Issue:1

weblogicdevelopersjournal.com

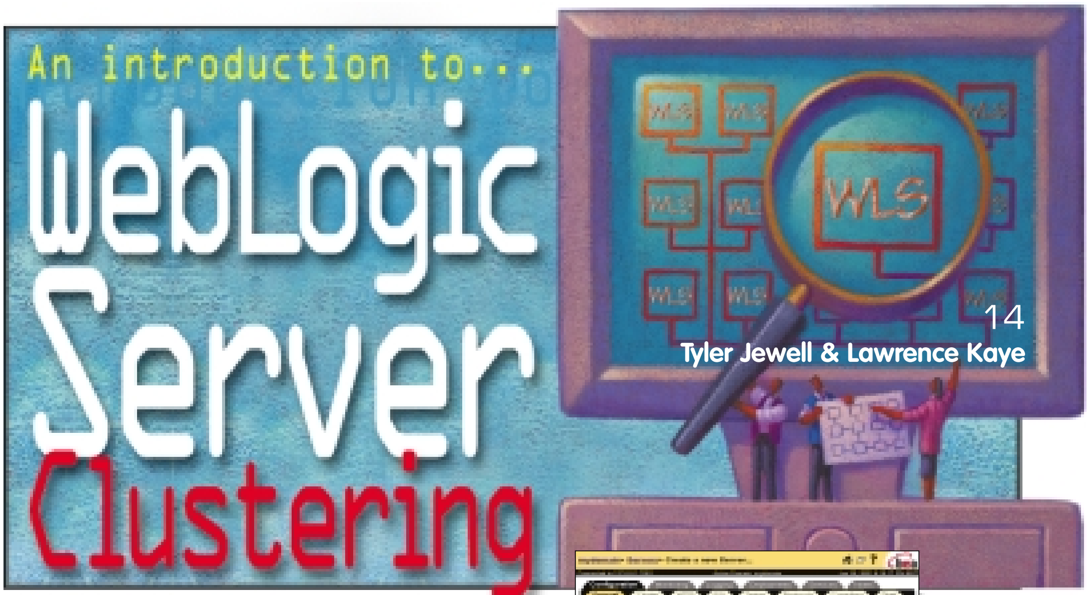
web services EDGE conference&expo 2002 WORLD TOUR

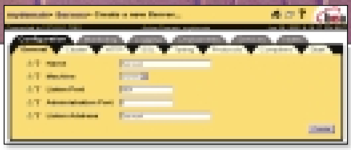
BOSTON, MA	JANUARY 29
WASHINGTON, DC	FEBRUARY 26
NEW YORK, NY	MARCH 19
SAN FRANCISCO, CA	APRIL 23

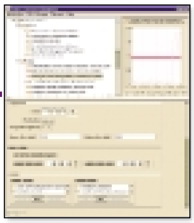
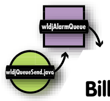
Page 56

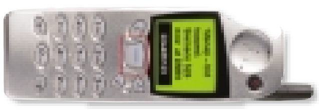
RETAILERS PLEASE DISPLAY UNTIL FEBRUARY 28, 2002
$15.00US $16.00CAN

SYS-CON MEDIA







bea

http://developer.bea.com

bea

http://developer.bea.com

wily technologies

www.wilytech.com

BEA WebLogic DEVELOPER'S JOURNAL

EDITORIAL ADVISORY BOARD
TYLER JEWELL, FLOYD MARINESCU, SEAN RHODY

FOUNDING EDITOR
PETER ZADROZNY

EDITOR-IN-CHIEF
JASON WESTRA

EDITORIAL DIRECTOR
JEREMY GEELAN

EXECUTIVE EDITOR
GAIL SCHULTZ

MANAGING EDITOR
CHERYL VAN SISE

SENIOR EDITOR
M'LOU PINKHAM

EDITOR
NANCY VALENTINE

ASSOCIATE EDITOR
JAMIE MATUSOW

ASSISTANT EDITOR
LIN GOETZ

WRITERS IN THIS ISSUE
PHILIP ASTON, JOHN GREEN, PETER HOLDITCH, TYLER JEWELL, LAWRENCE KAYE, WILLIAM KEMP, SAM PULLARA, CARL SEGLEM, JASON WESTRA, ROB WOOLLEN

SUBSCRIPTIONS
For subscriptions and requests for Bulk Orders, please send your letters to Subscription Department.
SUBSCRIPTION HOTLINE:
SUBSCRIBE@SYS-CON.COM
Cover Price: $15/Issue
Domestic: $149/YR (12 Issues)
Canada/Mexico: $169/YR
Overseas: $179/YR
(U.S. Banks or Money Orders)

PUBLISHER, PRESIDENT AND CEO
FUAT A. KIRCAALI

VP, PRODUCTION
JIM MORGAN

SENIOR VP, SALES & MARKETING
CARMEN GONZALEZ

VP, SALES & MARKETING
MILES SILVERMAN

VP, EVENTS
CATHY WALTERS

VP, BUSINESS DEVELOPMENT
GRISHA DAVIDA

CHIEF FINANCIAL OFFICER
BRUCE KANNER

ASSISTANT CONTROLLER
JUDITH CALNAN

ACCOUNTS PAYABLE
JOAN LAROSE

ACCOUNTS RECEIVABLE
JAN BRAIDECH

ACCOUNTING CLERK
PATTI DEL VECCHIO

ADVERTISING ACCOUNT MANAGERS
MEGAN RING • ROBYN FORMA

ASSOCIATE SALES MANAGERS
CARRIE GEBERT • ALISA CATALANO
KRISTIN KUHNLE

CONFERENCE MANAGER
MICHAEL LYNCH

EXECUTIVES, EXHIBITS
MICHAEL PESICK • RICHARD ANDERSON

SHOW ASSISTANTS
NIKI PANAGOPOULOS • JACLYN REDMOND

ART DIRECTOR
ALEX BOTERO

ASSOCIATE ART DIRECTORS
AARATHI VENKATARAMAN • LOUIS CUFFARI
CATHRYN BURAK • RICHARD SILVERBERG

WEBMASTER
ROBERT DIAMOND

WEB DESIGNERS
STEPHEN KILMURRAY • CHRISTOPHER CROCE

JDJSTORE.COM
ANTHONY D. SPITZER

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

SYS-CON MEDIA

Setting the **Standards**

BY JASON WESTRA
EDITOR-IN-CHIEF

Welcome to the inaugural issue of ***BEA WebLogic Developer's Journal*** (***WLDJ***)***!*** Anyone who has not been living under a rock for the past two years has seen J2EE (Java 2 Enterprise Edition) become the de facto standard for developing component-based, server-side applications. As a leading follower of that standard, BEA's WebLogic Server has become a standard in its own right. It is the most widely licensed application server in the world according to a recent independent study by Gartner Dataquest, which showed that BEA holds 41% of the new direct license revenue in the application server software market. The next closest competitor was IBM's WebSphere with approximately 31%. WebLogic Server has become the standard by which all others in the e-business infrastructure market compare themselves. Its suite of accompanying products, released as BEA WebLogic Portal 4.0 and BEA WebLogic Integration 2.1, also set the bar high for competitors.

This premier issue focuses on the core server that provides the foundation of BEA's new products. We feel it is important to understand the underlying platform upon which BEA's products are built before delving into the products themselves. In the future, this magazine will focus on the BEA product suite. Articles will be organized into categories to help you understand where and how the products interact with the core server –WebLogic Portal, WebLogic Integration, and WebLogic Server.

WebLogic Portal articles will show you how to customize access to information, applications, and business processes through its foundation services, administration services, and personalization and campaign management capabilities. WebLogic Integration articles will focus on how to deploy new Web and wireless applications so you can rapidly integrate these technologies with existing systems, as well as connect your customers and business partners from anywhere. Finally, WebLogic Server articles will help you with the platform services provided by the underlying J2EE server.

We have experts in BEA products from around the world contributing content to ***WLDJ***. For instance, this month we have a case study on a wireless application developed on WebLogic Server from Peter Zadrozny, the chief technologist for BEA Europe, the Middle East, and Africa. We will be providing case studies regularly, to show you innovation at work with WebLogic products.

Have you ever wanted to hear about the WebLogic EJB container? Each month, you can look forward to hearing from the WebLogic Server engineering team, who will be providing articles on a range of container-related topics like EJB development, deployment, and performance in WebLogic Server. Rob Woollen, the lead engineer of the WebLogic EJB container, heads up this month's issue with a discussion of performance-enhancing techniques for your entity beans. Also in this issue, we have Sam Pullara speaking from his soapbox about J2EE and WebLogic's continued industry-leading support for this standard. Sam was one of the first engineers to develop WebLogic (you know, before BEA even), and we are honored to have him writing on our team.

Finally, I personally invite you to send me your thoughts about what is working or not working for you with the magazine. ***WLDJ*** is here to provide you with the technical information you need to be successful with BEA WebLogic products. Just as WebLogic has set the standard to which other servers are compared, we at ***WLDJ*** will set the standard in providing the highest degree of technical, innovative writing about WebLogic! Enjoy!

AUTHOR BIO...

Jason Westra, CTO of Verge Technologies Group. is the editor-in-chief of **WebLogic Developer's Journal**. Jason has written for SYS-CON's publications for several years and has substantial knowledge of WebLogic Server.

CONTACT: jason@sys-con.com

JNDI

The Java Naming and Directory Interface (JNDI) has a central role in the Java 2 Platform, Enterprise Edition (J2EE). J2EE applications use JNDI to look up components and services, including JMS queues and topics, transaction services, JDBC connections, and EJBs.

To begin at the beginning... JNDI Initial Contexts

A PRIMARY STEP IN J2EE PROGRAMS

BY **PHILIP ASTON**

AUTHOR BIO...

Philip Aston is a senior consultant for BEA Professional Services, specializing in WebLogic Server. He also maintains "The Grinder," an open source load testing tool at http://grinder.sourceforge.net.

CONTACT...

paston@bea.com

The first programming step required to use JNDI is to create an initial context object that acts as the entry point into the JNDI namespace. In this article, I discuss how to create JNDI initial context objects for use with BEA WebLogic Server. The information provided is based on JNDI 1.2/Java 2 and is equally applicable to versions 5.1, 6.0, and 6.1 of WebLogic Server.

Client JNDI Access

If you've done any J2EE programming, the following code should be familiar to you.

```
import java.util.Hashtable;
import javax.naming.Context;
import javax.naming.InitialContext;

// ...

final Hashtable properties = new Hashtable();
properties.put(Context.INITIAL_CONTEXT_FACTORY,
"weblogic.jndi.WLInitialContextFactory");
final Context context = new InitialContext(properties);
```

This is the minimal J2EE code required for a Java client to establish a connection to the WebLogic Server JNDI tree. You must provide the fully qualified class name of an initial context factory that will be used to make the connection. For the WebLogic Server JNDI implementation, use the class weblogic.jndi.WLInitialContextFactory.

If you're not interested in J2EE compatibility, you can use the weblogic.jndi.Environment class, which provides type-safe methods for building the properties used to create initial context objects. Everything you can do with the Environment class you can do by setting properties directly, so I won't discuss it further.

The initial context factory class is loaded dynamically, so it must be in the run-time classpath but needn't be in the compilation classpath. If you consider compile-time type checking to be a good idea, you might prefer this modification.

```
// ...
final Hashtable properties = new Hashtable();

properties.put(Context.INITIAL_CONTEXT_FACTORY,
weblogic.jndi.WLInitialContextFactory.class.getName());

final Context context = new InitialContext(properties);
```

By default, WLInitialContextFactory attempts to connect to a WebLogic Server instance at the URL t3://localhost:7001. If you run this code snippet without an instance of WebLogic Server running on the local machine, a javax.naming.CommunicationException will be thrown. To connect to a server running on a different machine or port you need to specify a provider URL.

```
// ...
properties.put(Context.INITIAL_CONTEXT_FACTORY,
weblogic.jndi.WLInitialContextFactory.class.getName());

properties.put(Context.PROVIDER_URL,
"t3://server:9999");

final Context context = new InitialContext(properties);
```

The provider URL specifies the server host name and port and the protocol to use. If you're bridging a firewall, you might wish the connection to be made using an HTTP tunnelled connection. (You should ensure that the server has HTTP tunnelling enabled.)

```
// ...
properties.put(Context.PROVIDER_URL,
"http://server:9999");
// ...
```

If the connection to the remote server takes an inordinately long time, say several seconds, there is usually a problem with the client's DNS configuration. Try replacing the host name with the server's IP address. If this cures the problem, investigate further: using host names rather than IP addresses is preferable from a network administrator's viewpoint. Context.INITIAL_CONTEXT_FACTORY and Context.PROVIDER_URL are examples of JNDI environment properties.

Server JNDI Access

Creating an initial context from server-side code is trivial.

```
final Context context = new InitialContext();
```

This provides a connection to the local WebLogic Server JNDI tree. Other than Context.SECURITY_PRINCIPAL and Context.SECURITY_CREDENTIALS (discussed below) you rarely need to pass additional JNDI environment properties. If you want to, you can safely specify Context.PROVIDER_URL. WebLogic Server detects attempts to connect to the local JNDI tree and doesn't create a new socket connection to the local listen port.

Alternative Ways to Set JNDI Environment Properties

What if you don't want to hard-code JNDI environment properties? You might not know the location of the servers in your deployment environment, you might want the option of switching to use another JNDI provider without recompiling or you might want to write library code that will be deployed on both client and server machines. There are a couple of standard JNDI features that allow you to do this without writing code.

First, you can set default values using Java system properties. Standard Java Virtual Machines have a *-D* option that allows system properties to be set. To make WLInitialContextFactory the default initial context factory, you could start your client JVM as follows:

```
$ java -Djava.naming.factory.initial=
weblogic.jndi.WLInitialContextFactory
com.bea.paston.jndi.client.MyJNDIClient
```

The string "java.naming.factory.initial" is the value of Context.INITIAL_CONTEXT_FACTORY. Other common values are given in Table 1.

Second, you can set default values by creating a Java properties file (see java.util.Properties) called jndi.properties and placing it in the classpath. The file should contain lines of the form property name=value, where property name is taken from the table. The file can also be placed in the current working directory or in the lib directory of the Java Virtual Machine ($JAVA_HOME/lib).

You can mix and match these methods and use multiple jndi.properties files. If a property is specified using more than one method, the value to use is determined according to the following list (highest precedence first):

- Explicitly coded in creation of an InitialContext
- System property
- jndi.properties file in classpath
- jndi.properties file later in classpath
- jndi.properties file in current working directory
- jndi.properties file in JVM lib directory

Applets and JNDI Environment Properties

JNDI provides a convenient way to set properties for applets. If you code your initial context look-up within your applet as follows, you can use applet parameters to specify JNDI environment properties.

```
// ...
final Hashtable properties = new Hashtable();
properties.put(Context.APPLET, this);
final Context context = new InitialContext(properties);
```

A typical applet tag is shown below.

```
<applet codebase="/classes"
code="com.bea.paston.jndi.client.TestApplet">
  <param name="java.naming.factory.initial"
value="weblogic.jndi.WLInitialContextFactory">
  <param name="java.naming.provider.url"
     value="http://server:7001">
</applet>
```

Applet parameters are not read unless you set the Context.APPLET property.

TABLE 1

NAME	JAVA STRING	DESCRIPTION
java.naming.factory.initial	Context.INITIAL_CONTEXT_FACTORY	JNDI implementation class
java.naming.provider.url	Context.PROVIDER_URL	Network location of JNDI service provider
java.naming.security.principal	Context.SECURITY_PRINCIPAL	Security identity to use for connection authentication
java.naming.security.credentials	Context.SECURITY_CREDENTIALS	Security token to use for connection authentication

Common JNDI properties

Applets don't normally have the necessary security rights to read the system properties and don't have the concept of a current working directory, so JNDI environment properties can't be set using these methods. The order of resolution for applets is as follows:

- Explicitly specified in creation of an InitialContext
- Applet parameter
- jndi.properties file in classpath
- jndi.properties file later in classpath
- jndi.properties file in JVM lib directory

JNDI and WebLogic Server Security

From version 6.0, WebLogic Server has an implementation of the Java Authentication and Authorization Service (JAAS). JAAS is the J2EE standard mechanism for establishing security contexts and is the BEA recommended approach for new developments.

Each thread within a WebLogic Server instance or WebLogic RMI client has an associated stack of WebLogic Server security contexts. When you create a new initial context object using the WLInitialContextFactory with the JNDI environment properties Context.SECURITY_PRINCIPAL and Context.SECURITY_CREDENTIALS set to valid authentication details, a new security context is pushed onto the stack. This security context is used for every access control check against actions performed by that thread, whether the action involves local or remote services. When you invoke close on the initial context object, the security context is popped off the thread stack and the previous security context takes effect.

“Client-side code can save unnecessary network calls by reusing the references returned from JNDI look-ups where possible”

Within WebLogic Server, each request, whether it is a servlet call, JMS message, or RMI invocation, is processed by a single execute thread. The initial security context is determined by the authentication associated with the request. The creation of an initial context can be used to associate a new security context with the request. When the execute thread has finished processing the request, the security context stack is discarded.

If you have some code that needs to access a component, say an EJB, under a specific identity you can simply set up an initial context appropriately before looking up the EJB's home interface, use the EJB, and then close the initial context. However, while the association of security context with JNDI is convenient in some circumstances, it is very clumsy in others. The security context is associated with the current request, not the object obtained from JNDI, so you have to create a new initial context every time you need to access the EJB. This is where JAAS can help; it provides an independent way to alter the security context.

You should not attempt to share initial context objects created using Context.SECURITY_PRINCIPAL and Context.SECURITY_CREDENTIALS properties between threads.

Closing Contexts

To be a good J2EE citizen, you should ideally ensure that close is invoked on an initial context object after use. This lets the JNDI implementation free up any associated resources. Don't worry too much about this with WebLogic Server, there are no costly associated resources and the current implementation of close just affects the security context.

Should I Cache JNDI Look-Ups?

Client-side code can save unnecessary network calls by reusing the references returned from JNDI look-ups where possible. Where the client is retrying a failed call to a remote component it can make sense to perform the JNDI look-up again; this allows the client to be more robust when components are redeployed, the server is rebooted and so on.

Creating an initial context object and performing a local JNDI look-up from server-side code both take about a hundred nanoseconds on typical server hardware. It is usually convenient and frugal to cache the results of the look-up. This is commonly done in servlet init and session EJB setSessionContext methods.

There are two reasons you might not want to cache the reference resulting from server-side JNDI look-ups. First, you may wish to detect dynamic changes to the JNDI tree. Second, you may wish to use the InitialContext creation to set the appropriate security context. (You can cache references and create independent initial context objects to alter the security context but I hope, after reading the previous section, you are searching for the JAAS documentation.)

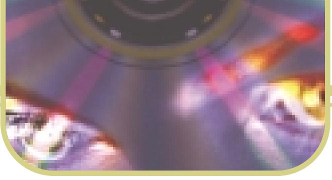
Next Steps

I've covered the details of establishing an initial context to the WebLogic Server JNDI tree. This only scratches the surface of the JNDI API but is a key first step for most J2EE programs.

In future articles I will delve into more involved use of the JNDI, examining topics such as JNDI federation and the role of JNDI in WebLogic clusters.

BEA e-world

www.bea.com/events/eworld/2002



An oxymoron? I don't think so...

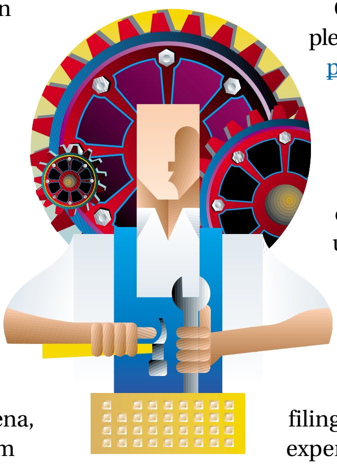
BUILDING A KNOWLEDGEABLE AND EMPOWERED FORCE

BY **JOHN GREENE**

WE ALL KNOW THE STORY, RIGHT? In today's high-tech world, technical support has suffered a demise worse than Darryl Strawberry's life after baseball. It's virtually impossible to get anyone knowledgeable on the phone. Support is the lowest item on the totem pole. Qualified folks move on while the no-ops remain. Support engineers are bitter, angry, disgruntled individuals, bordering on inhuman. Tech support is such a rip-off. The executives allowing this blatant disrespect toward their customers should lose all of their stock options.

Got that out of your system? Good. Because what I'm going to share with you in this column is not only going to bring back some of the warm fuzzies you used to get from technical support, it's going to show you what BEA is doing to make WebLogic support vehemently defy the pathetic support standard set (and accepted) by the dot-bomb generation.

In this first column, I'd like to introduce myself, describe the column's purpose, and tell you a little bit about the WebLogic support organization at BEA. I'll also share with you why we truly believe that we transcend articles like Jon Katz's acrid article on www.slashdot.org/features/01/04/30/1627201.shtml. I read this and half the responses and just couldn't take anymore – though it did give me some ideas for content. Finally, I'll give you a preview of some future topics and show you how you can improve your support experience with BEA and contribute to this column's success at the same time.



So, let's get started. First of all, I'm John Greene. I'm the Backline Engineering Interface manager for WebLogic Server in the San Francisco office at BEA Systems, Inc. I've been with BEA for more than two years – the first as a DRE and the next as a manager (managing a group of 10 DREs). I know, I know, what is a DRE? We'll get to that. Before coming here, I worked in the computer science department at the University of Massachusetts Amherst as a SysAdmin, Web/database programmer and Mac/UNIX support specialist. Prior to that, I did various consulting projects, mostly in the database arena, while getting my degree in computer science (also from UMass).

BEA's Support: "Developer Relations Engineers"

BEA's principal core value is: "Customer issues transcend all others." Delivering on this mission requires a knowledgeable and empowered support force with good communication skills as well as a solid understanding of object-oriented programming, J2EE, and the internals of WebLogic Server. These valuable engineers build and maintain relationships with customers through support cases filed with BEA. Since our customers are developers, BEA adopted the term Developer Relations Engineer as a more appropriate title for our support force.

But DRE isn't just a title, it's a mindset. The DRE is one of the highest-profile and prestigious positions at BEA (and is compensated accordingly). While troubleshooting support cases, DREs often work directly with CCEs (Customer Centric Engineers), development engineers, high-visibility customers, and their managers. Because of their multifaceted view of the product and their unique understanding of how customers use WebLogic in the real world, senior DREs are often coveted by other groups in the organization. While some DREs do move to other areas after some time, many choose to stay in support. Surprising? Not to me... maybe to Katz. You see, there is a pride and dedication that comes with supporting a product like WebLogic, and DREs have accepted the challenge with dignity, courage, and a desire to learn and excel. Just two years ago there were only about a dozen people supporting WebLogic in the Americas; now there are more than ten times that number. What a roller-coaster ride!

AskBEA

Of course, world-class support isn't just about the people. Last year, we unveiled AskBEA, www.bea.com/support, our answer to self-help on the Web (see the related article in this issue). Within one year, AskBEA's usage has increased over 500%, with daily queries averaging over 3,000 since April 2001. As some customers are able to help themselves through the Natural Language querying capability of AskBEA, DRE resources are freed up to spend more time with the customers whose problems aren't solvable by AskBEA.

Now What?

In my next columns, I'm going to cover topics such as: "WebLogic Self-Help: What to do before filing a support case," "Streamlining the customer/DRE experience," and "Comm101: Effective customer/BEA communication." I'm also going to share current support experiences, tidbits, lessons we (and customers) have learned, etc. Beyond that, well, that's partly up to you – the developer. What would you like to see in a column devoted to WebLogic support? This column is for you! So let me hear from you: jg@bea.com (no Lakers fans, please).

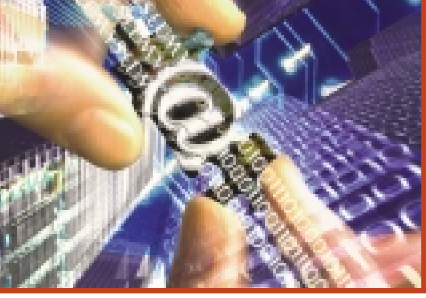
AUTHOR BIO...

John Greene joined BEA as a WebLogic developer relations engineer in June 1999; he is now the Backline Engineering Interface manager. John has a degree in computer science from the University of Massachusetts and is a rabid fan of Philadelphia sports teams.

CONTACT: jg@bea.com

sonic sofrware

www.sonicsoftware.com



What Is a Transaction?

Since this will be a monthly column on the subject of transactions, which from my experience seems to be a subject that everybody has heard of, but nobody is familiar with, I thought I would build up speed with a back-to-basics look at transactions, what they are, and what they're for.

Acid Reign

"THE TRANSACTION PROCESSING MONITOR IS DEAD. LONG LIVE THE TRANSACTION PROCESSING MONITOR"

BY **PETER HOLDITCH**

AUTHOR BIO...

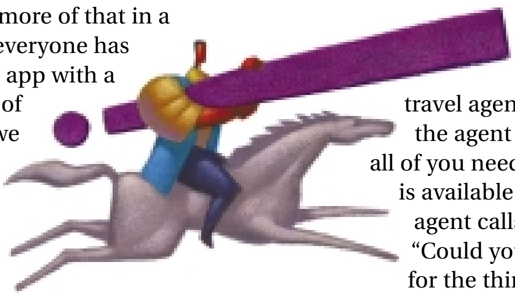
Peter Holditch joined BEA as a consulant in the Northern European Professional Services organization in September 1996. He now works as a pre-sales architect in the UK. Peter has a degree in electronic and computer engineering from the University of Birmingham. When he's not pre-selling architecture, he likes to build furniture, brew beer, and enjoy the long hot British summers.

CONTACT...

peter.holditch@bea.com

Having worked in software infrastructure for some years, I've found that a recurring objection to programming to any kind of framework – J2EE being just the latest one – is that it "adds complexity to development." While I think that this is demonstrably not true, one of the facets of this style of programming that I have heard cited as "making things more complex" is transactions, and the whole business of declaring or coding them (declaring, hopefully, but more of that in a future column). After all, everyone has written that killer two-tier app with a pretty front-end and a bit of drag-and-drop SQL, and we didn't need to worry about transactions then, did we?

Actually, of course, transactions still existed, but they were managed transparently by the database. Each SQL statement was executed bracketed in its own implicit transaction, started, and completed on your behalf by the database engine. For more complex applications, guess what? You coded SQL BEGIN and SQL COMMIT and had to worry about transactions. If your situation was more complex, and you needed a transaction that spanned multiple disparate systems, all bets were off.



So what is a transaction monitor actually doing for us? I'll devote the rest of the column this month to an allegorical walkthrough of a transaction, just so it's clear what we are getting for our time and effort by grappling with transactions in an application system. Next month, I'll cover some "closer to the metal" issues.

Let's assume you want to book a holiday, and also that holidays are always available on demand. You call your travel agent and (checking in your diary for a free week) tell the travel agent to book a holiday for you. The travel agent tells you it's all done, so you write the holiday into your diary and go off in search of your bucket and spade. In this scenario, the *resource* is your time (as allocated by your diary), the *resource manager* is you (you check and write it into the diary), and the *transaction* is the holiday reservation. That all sounded very easy, so what's the fuss about transactions? Well, the astute among you will have noticed that there was only one resource manager, since you have no friends and are going on holiday alone. This allowed for a *one phase commit* – that is, you simply recorded the holiday in your diary and the transaction was complete. Back in the wonderful world of technology, this is equivalent to an update made to a single database instance.

Now, let's book a group holiday. You and four friends want to go away together. That's pretty easy to organize if you're all in the same room, but let's assume that you all live in different time zones. You agree among yourselves where you want to go, and in what month, and you ask the travel agent to book for you. Before the agent can confirm the booking, all of you need to make certain the time is available in your diaries. The travel agent calls all five of you and says, "Could you pencil this holiday in for the third week?" and in response, each of you pencils the holiday into your diary. Next, if everybody can make it, the travel agent makes the reservations and calls all of you again to confirm the holiday.

In this scenario, you and your four friends were the *resource managers*, the diary entries were the *resources*, and the travel agent played the part of the *transaction manager*. The *two-phase commit* was the two telephone calls that the travel agent made to each of you. The first phase (or the *prepare* phase) was when you pencilled the holiday into the diary. If someone had been unable to make the date in question, the holiday would have been aborted. The second phase was when you confirmed the holiday, and committed it in ink into your diary (you guessed it, the *commit* phase). Of course, between the first and second phases if you had a heart attack you could refuse to commit the holiday on the second phase. That would have caused a *heuristic* outcome – some of your friends, but not all, would be going on holiday. In the real world of IT systems, this doesn't mean that you get a get well soon postcard when heuristic completions happen. Instead, your application administrators are paged in the middle of the night and have to sort out the inconsistent business data, possibly giving them a heart attack!

Another difference between our travel booking scenario and the real world is that in the real world, it is generally considered extremely unlikely that a travel agent will suffer a seizure while making a set of phone calls. In the world of technology, computer crashes are more common. For this reason, a transaction manager will write a *commit record* to the *transaction log* when the decision to go ahead with the second phase of the commit is taken (to revert to the terminology, the transaction at this point is said to be *decided*). In fact, the second phase will not commence until this log record is physically on disk. In this way, should the computer running the transaction manager crash during the second phase, it can recover all decided transactions from the transaction log and drive them to completion when the needed systems are brought back online. Once all the resource managers have replied OK to the commit phase, the record for the transaction is no longer needed and may be deleted.

The only place that this analogy breaks down slightly is that our travel agent transaction manager had some knowledge that it was a holiday it was coordinating. In a real transaction system, the transaction manager merely knows that it's coordinating something. Therefore, instead of asking you and your friends, "Can you make this holiday for the third week?" it would just say "Code XYZ123, is that OK?" and you, as the resource manager, would know that XYZ123 mapped to the holiday at the time in question. XYZ123 is a *transaction identifier*, generated by the transaction manager when the transaction was started, and your responsibility as a resource manager is to map it onto some set of changes to your resources.

It is in this way that the transaction manager and resource manager's responsibilities are kept separate. The application code talks to the transaction manager to begin, commit, or abort the transaction, and talks to the resource manager in whatever way seems appropriate, via SQL, isam, a remote procedure call, etc. So long as the resource manager knows that the work is being done on behalf of an external transaction, it knows what to commit or rollback when the time comes.

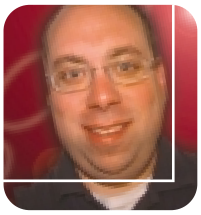
Conclusion

I hope that this light walkthrough has been useful. Next month, I promise more hot technology action as I delve into when to use transactions in an application, and (perhaps more importantly) when not to.

corda technologies

www.corda.com

An introduction to... WebLogic Server Clustering



BY
TYLER JEWELL
AND
LAWRENCE KAYE

AUTHOR BIOS...

Tyler Jewell is BEA's principal technology evangelist. Tyler is the coauthor of Mastering Enterprise JavaBeans 2.0, coauthor of Professional Java Server Programming (J2EE 1.3), a regular J2EE columnist at www.onjava.com>-www.onjava.com, and the technology adviser to theserverside.com where he writes about Web services.

Lawrence Kaye is the manager of delivery strategy in BEA Educational Services. He has five years of Java development experience and has taught WebLogic Server development and administration courses for the past two years.

CONTACT...

tyler@bea.com
lawrence.kaye@bea.com

Welcome to the first issue of BEA *WebLogic Developer's Journal*! This article is the first of a three-part series geared toward the clustering capabilities of BEA WebLogic Server (WLS) 6.1 and aimed at introductory and advanced audiences. This article will talk about the importance of clustering and the high-level clustering capabilities of WLS, provide an in-depth analysis of HttpSession clustering and persistence, discuss basic configuration and troubleshooting, and provide an example that ties together everything discussed in this article. The second article will provide an in-depth analysis of replica-aware stubs, their impact on a clustered system, and how they are used with EJBs, JMS objects, and DataSource objects. The last article will discuss clustering best practices, including the single-tier, coupled model; multi-tier, coupled model; and the multitier, decoupled model.

A STRAIGHTFORWARD YET FLEXIBLE

The Motivation Behind Clustering

A single server, despite the best engineering in the world, can only do so much. There are many things that a single server cannot reliably survive: hardware failure, network failure, or too many requests. This aspect of enterprise development is just a fact of life. How do developers deal with these issues? Clustering. Clustering is a broad term applied to any system that gets multiple servers, components, or resources to act in a coordinated way and to provide a single, simple view of the services they offer.

An e-business system probably makes use of a cluster to maintain levels of throughput for requests, provide back-up systems in case of unforeseen failures in nodes, or to provide additional processing power to the system. Any system that supports clustering can be characterized by the Reliability, Availability, and Serviceability (RAS) properties:

- ***Reliability***: An indication of whether or not the system can predictably handle requests in a consistent amount of time. If the number of requests, transactions, or concurrent users increases, the reliability of the system is high if the system can maintain the same quality of service for requests that was available when there were few requests, transactions, or concurrent users. The reliability of a system is low if response time or quality of service decreases as the load on the system increases. A two-node system has *higher* reliability than a one-node system.
- ***Availability***: An indication of the amount of time that the system makes its services available to clients. Highly available systems have very few periods (if any) that the services cannot be accessed by a client. Systems with many periods where the services are not available have low availability. A two-node system has *higher* availability than a one-node system.
- ***Serviceability***: An indication of the effort involved with maintaining, monitoring, configuring, and operating a system. The serviceability of a system is high if very little effort is involved with monitoring the system and keeping it functioning. The serviceability of a system is low if the effort involved with operating the system is high, cost prohibitive, or tedious. A two-node system has *lower* serviceability than a one-node system.

There is no such thing as a perfect design for a clustered system. *Anything that is done to a system to increase reliability and availability decreases the serviceability of that system.* The antithesis holds true as well: anything done to increase the serviceability of a system decreases the availability and reliability of that system. The guiding light for clustering architectures is a trade-off analysis: the right design is the one that meets your requirements for each of the RAS properties.

WLS Clustering Capability Overview

There are a variety of things that WLS can do for a system to increase reliability or availability. Generally, many of the options that are available to select from increase either the reliability or the availability of the system, but not always both. Generally, WLS supports technologies that make load balancing of requests feasible or facilitate a high availability switchover of a current request to another server that can successfully process the request. This section discusses the major requirements and capabilities of WLS clusters.

SYSTEM REQUIREMENTS FOR CLUSTERING

There are certain physical requirements and guidelines that must be followed when planning a WLS cluster. These include:

- A cluster can include one or more servers. Both administration servers and managed servers can participate in a cluster, but it would not be wise to have an administration server participate in a cluster that is performing the bulk of the processing for your system. Currently, the administration capabilities are not clusterable, but will be in the future. Many people believe that this is a single point of failure of the system, but it is not. If the administration server crashes, any managed servers that have been started will continue to execute. Allow administration servers to administer and allow managed servers to perform your business logic.
- A cluster can be run on one or more computers. Since a single computer can execute multiple WebLogic Server instances, an entire cluster of multiple servers can be run on a single machine. A computer with many processors and lots of RAM could potentially support a cluster with many nodes. Your physical node design should distribute servers that will participate in a cluster appropriately across one or more machines that you may have available. Placing different nodes on different machines allows your system to resist hardware failures. Additionally, when you have a scenario where you are replicating your HttpSession or stateful session EJBs (discussed below), a WLS cluster will attempt to replicate the backup object onto a server located on a different machine.
- WebLogic Server clusters can only operate on a LAN. WLS clusters use IP multicast in certain situations to communicate with one

IDEA FOR USE IN E-BUSINESS SYSTEMS

another. IP multicast traffic is very unreliable over a WAN and would disrupt cluster communication. IP multicast traffic isn't 100% reliable in a LAN environment, but it is much more so than a WAN. This requirement also implies that all servers in the cluster must be reachable by IP multicast.
- WebLogic servers that participate in a cluster must have static IP addresses allocated to them. You cannot have a machine use dynamic IP addresses with a WLS cluster.
- All servers in a cluster must be running the same version of WLS. Since different versions of WLS will have different serialVersionUID numbers associated with each class (due to a recompilation), interserver communication in a cluster would be fraught with ClassCastExceptions when objects are transferred between servers running different versions.

WLS CLUSTERING CAPABILITIES

There are many places, locations, and activities that a WLS cluster performs to provide developers with a reliable and available system. The most common misconception about WLS clusters is that many developers believe that the cluster itself (the servers in the cluster) performs load balancing or request redirection. This *never* occurs in a WLS cluster. Once a request has made it to a server that server (and only that server) will handle the request or propagate an error condition. This means that all load balancing occurs by business logic that is hosted in front of the cluster (or something that communicates to the cluster). Additionally, this means that failover logic exists outside of the cluster, but the cluster replicates any data that needs to be replicated to make sure that a failover request is successful.

The high level clustering capabilities of WebLogic Server include:
- Unified Naming Servers
- Cluster Heartbeats
- Cluster-Aware Stubs
- Replicated Data Objects (HttpSession and stateful session EJBs)

UNIFIED NAMING SERVERS

WLS implements a decentralized, unified naming server approach. Each server in the cluster has its own naming server and all of the naming servers advertise the same services to all clients even if some of the services advertised are located on other servers in the cluster. Each naming server is aware of the services that are hosted on it and the services that are hosted on another server. IP multicast messages are used for interserver communication to announce new, updated, or removed services from one server to the others in the cluster. For example, when you create a new TxDataSource to a connection pool and deploy the TxDataSource to a single server in the cluster, the server that hosts the TxDataSource will send a copy of the TxDataSource object representing the connection pool to all of the other servers in the cluster using IP multicast. The TxDataSource is hard-wired (pinned) to the server that has its connection pool so any client that requests the TxDataSource will be able to download it from any server. Once downloaded to the remote client, the TxDataSource will communicate with the server that contains the connection pool *even if the client downloaded the TxDataSource from a different server*.

A WLS naming server will track all of the services located on all servers. In the situations where the same service (such as an EJB) is available on more than one server, the naming server will track the objects that represent the same service for all of the servers. Following the connection pool example described above, if the TxDataSource is deployed to the whole cluster and the same connection pool that the TxDataSource manages is available on each server in the cluster, each naming server will have a TxDataSource object that was sent to it from each of the other servers in the cluster. If you have a ten-server cluster, each naming server will have ten TxDataSource objects, each one representing a connection pool on a server.

When a client requests a service from a naming server, the server will always attempt to send the object representing the service on the current server to the client. Following our example, the naming server will send the TxDataSource that maps to the connection pool on the current server before it sends a TxDataSource representing the same connection pool hosted on another server. If the service requested is not currently available on the current server, then the naming server will traverse its linked list of objects representing resources on other servers and send one of those instead.

As services are deployed and undeployed, a WLS server will announce the changes over IP multicast. When a naming server receives an undeploy message for a service, it merely has to remove the object representing that service for that particular server from its naming tree. When a service is deployed or redeployed, a naming server merely has to add the object representing the service for that server to its naming tree.

CLUSTER HEARTBEATS

Each server in the cluster sends out a regular heartbeat every ten seconds over IP multicast. This "I'm alive" heartbeat is to allow the other servers in the cluster to be aware that a particular server is still present. All servers in a cluster monitor and watch for heartbeats from all other servers in the cluster. If a heartbeat from another server in the cluster is missed three times (30 seconds), then the server tracking the heartbeats assumes that the other server is dead and removes all of the objects representing services located on that server. In a situation where a network failure has occurred and none of the nodes can communicate over IP multicast, all of the servers will miss the heartbeats from all of the other servers. After 30 seconds, each server will assume all of the other servers are dead and remove any objects from those servers that represent services hosted elsewhere. Essentially, each server would be operating in a standalone fashion, but still waiting for heartbeats to reappear. In the situation where two servers are able to reconnect, they will resynchronize their objects for their naming trees.

CLUSTER-AWARE STUBS

JMS ConnectionFactory objects, EJBs, home stubs, and JDBC TxDataSource objects are all factory objects that are hosted in a naming tree. Clients download these objects and then invoke a method to obtain a remote reference to the resource they intend to use. This is a Connection object for JMS, a remote stub object for an EJB, and a Connection object for JDBC.

In WLS, the factory objects that are downloaded are called cluster-aware stubs. Since the same service may be available on multiple servers in the cluster, the factory object downloaded from JNDI includes load balancing and failover logic for locating an appropriate reference to the resource. JMS ConnectionFactory objects, EJBs, home stubs, and JDBC TxDataSource objects are all cluster-aware stubs that can return a reference to a resource that is located on any server in the cluster.

Cluster-aware stubs communicate with the cluster to maintain an active list of servers that host the service they are a factory for. When a client uses a cluster-aware stub to obtain a reference to a resource, the cluster-aware stub can load balance that request to different servers in the cluster. If you have a connection pool deployed to all of the servers in the cluster, a client that calls getConnection() on a TxDataSource multiple times will receive a Connection object from a different server upon each invocation. *This behavior only applies to Java clients that are not colocated within the same VM as the server itself. In a situation where the requesting client IS colocated in the same VM as the server, it always makes sense to return the reference to the resource hosted locally. Any attempt to load balance a request that can be handled locally is an unnecessary burden to the system.*

Additionally, since a cluster-aware stub is constantly updated with the correct list of servers that are hosting the service, the cluster-aware stub will never make a request to a server that isn't hosting the service.

Cluster-aware stubs and EJBs are an interesting topic because:

1. EJBs have two stubs (home and remote stubs).
2. EJBs have three behavior types (stateless, stateful, and persistent) that can facilitate or limit the load balancing or failover capability of a cluster-aware stub.

Cluster-aware stubs and EJBs will be discussed in more detail in the next article in this series.

REPLICATED DATA OBJECTS (HTTPSESSION AND STATEFUL SESSION EJBs)

The J2EE specification defines two types of server-side data objects for short-term storage of an application's state: HttpSession objects and stateful session EJBs. These objects provide a means of sharing data between requests without explicitly requiring the passing of data as method parameters and return values. These objects serve to simplify intra-application communication, not to provide robust storage of transactional data – that is the role played by entity EJBs or direct database interactions via JDBC.

Since HttpSession objects and stateful session EJBs are not designed to survive server crashes, WebLogic Server provides data object replication to help resist failures in enterprise systems. For HttpSession objects, there are several options: JDBC replication, cookie persistence, and in-memory replication.

JDBC replication stores session content in a relational database table. This process offers two advantages. First, persistent storage ensures survival of the session data even if the entire cluster were to become inoperable. Second, storage in a central repository allows any server in a cluster to be able to handle any session. Despite this, JDBC replication has poor performance since every update to a session instance requires synchronization with the database.

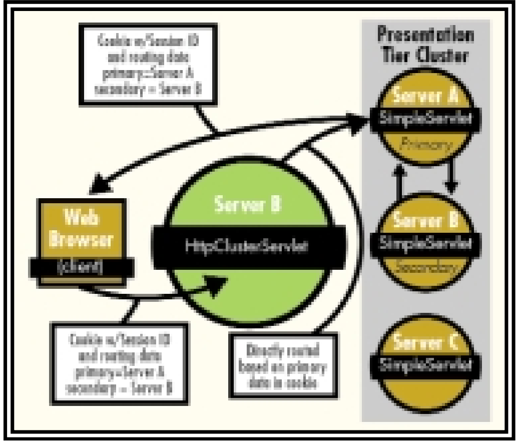
Cookie persistence involves serialization of the session object into a cookie that is passed between the Web browser and server with each HTTP request. Cookie persistence allows any server access to any session while also providing fast in-memory access to the session's contents. However, there are some drawbacks:

- Only String attributes can be stored; all others will be ignored when the cookie is generated.
- The cookie must be written to the header data before the response is committed. The HTTP response cannot be flushed programmatically or based on buffer overflow until the cookie has been written.
- There is limited security since the session data is sent in clear text.
- The browser may impose a maximum size of the cookie and, therefore, the amount of stored data.

The most popular method for HttpSession failover is in-memory replication, which can also be applied to stateful session EJBs. With this technique, there are at most two servers in the cluster that have a copy of a data object in memory. This method offers high efficiency with quick access to the data object's contents while conserving runtime memory across the servers in a cluster.

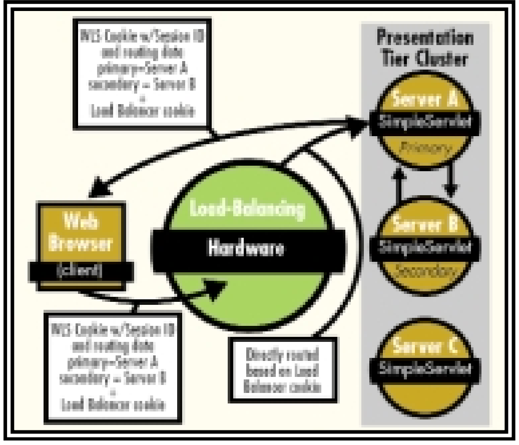
In-memory replication requires routing all requests for a given HttpSession to the server that has that session. This can be done with cookies or URL rewriting. When a clustered server receives an HTTP request unaffiliated with an HttpSession, the server creates a new HttpSession object with a unique ID, registers itself as the primary server, and

FIGURE 1



Request Routing with HttpClusterServlet

FIGURE 2



Request Routing with Load–Balancing Hardware

selects a backup server from those remaining in the cluster. Point-to-point communication between the primary and backup servers is established at this time to allow for the synchronization of the HttpSession object. The IP addresses and ports of the primary and backup servers are encoded as part of the session ID for the HttpSession.

LOAD BALANCING AND HTTPSESSION FAILOVER

The specifics of how subsequent HTTP requests are handled are based on the load-balancing mechanism selected for initial access to the cluster. WLS offers several options: the HttpClusterServlet, Web server plug-ins, and support for load-balancing hardware.

The HttpClusterServlet, available with the standard WebLogic Server installation, runs in a non-clustered instance of WLS (i.e., a proxy server) and contains the logic necessary to parse the routing information in the WLS-generated session ID. HTTP requests not containing a session ID are load balanced by a round-robin algorithm. HTTP requests that contain a session ID are routed to the server containing the primary HttpSession instance (see Figure 1). When the server hosting the primary instance becomes unavailable, the backup server senses it through the loss of point-to-point communication, promotes itself to primary, and selects a new backup from the remaining clustered servers. Upon the next request, the HttpClusterServlet detects that the original primary server is unavailable and reroutes the request to the specified backup server. The response contains the session ID with revised routing data. The ease of configuring the HttpClusterServlet makes it a valuable tool during the development process.

With Web server plug-ins, developers can capitalize on the investment already made on Netscape, Microsoft, and Apache Web server licenses to serve static Web content while delegating requests for dynamic page generation, via servlets and JSPs, to the WebLogic Server. The plug-ins, available as shared or dynamic libraries, provide the same round-robin load balancing and HttpSession routing mechanisms as the HttpClusterServlet. Additionally, the Netscape and Apache plug-ins provide routing based on URL pattern matching.

Load-balancing hardware provides built-in firewall capabilities and sophisticated load-balancing algorithms. To consistently route traffic to a server hosting a primary instance, most vendors will either insert their own cookie in the client session (active cookie persistence) or interpret part of an existing cookie as a single numerical differentiator (passive cookie persistence). WebLogic Server in-cluster routing mitigates load-balancing hardware's disregard for the backup server (see Figure 2). When a request is received on a clustered server that has no knowledge of a given session, the routing data stored in the session ID is queried, the server hosting the backup instance is contacted, and the receiving server becomes the host to the new primary instance. In other words, the server hosting the backup instance becomes a factory for creating new primary instances, rather than being promoted itself.

TABLE 1

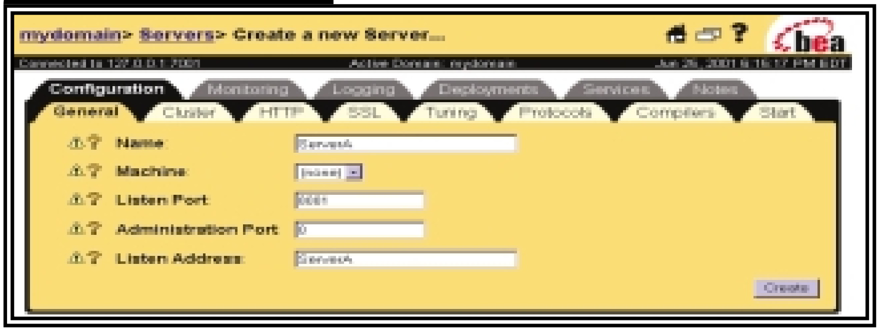
CONFIGURATION PARAMETER	VALUE
For Clustered Servers:	
Name	Server A
Listen Port	8001
Listen Address	Server A
Name	Server B
Listen Port	8001
Listen Address	Server B
Name	Server C
Listen Port	8001
Listen Address	Server C
For Proxy Server:	
Name	ProxyServer
Listen Port	8001
Listen Address	192.168.58.103
For Cluster:	
Name	PresentationTierCluster
Cluster Address	192.168.58.100,192.168.58.101,192.168.58.102:8001
Multicast Address	224.0.0.1
Chosen Servers	Server A, Server B, Server C

Server and cluster configuration data

altoweb

www.altoweb.com

FIGURE 3


mydomain> Servers> Create a new Server...

Connected to 127.0.0.1:7001 Active Domain: mydomain Jun 26, 2001 6:16:17 PM EDT

Name: ServerA

Machine: (none)

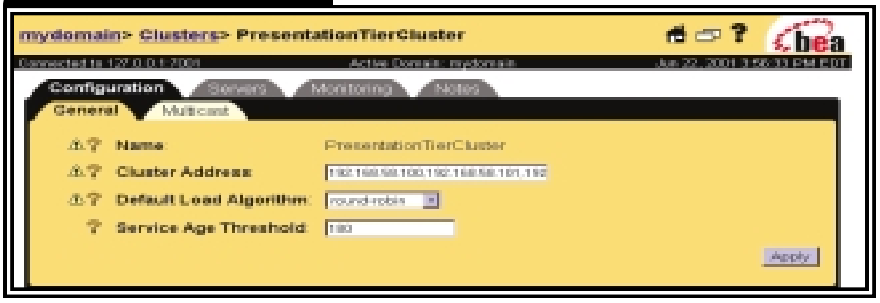
Listen Port: 8001

Administration Port: 0

Listen Address: ServerA

Server configuration

FIGURE 4


mydomain> Clusters> PresentationTierCluster

Connected to 127.0.0.1:7001 Active Domain: mydomain Jun 22, 2001 3:56:33 PM EDT

Name: PresentationTierCluster

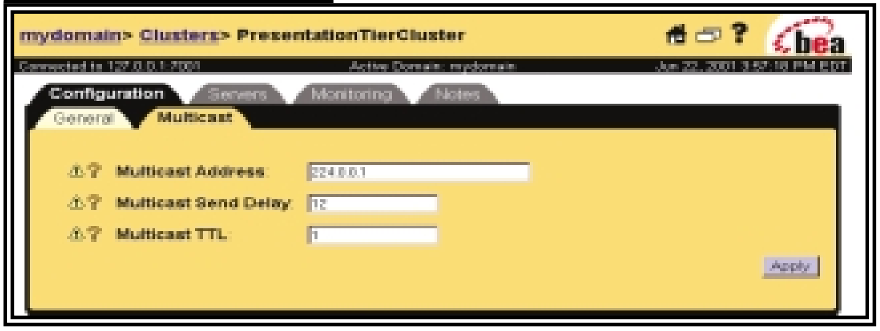
Cluster Address: 192.168.58.100,192.168.58.101,192

Default Load Algorithm: round-robin

Service Age Threshold: 180

Adding servers to a cluster

FIGURE 5


mydomain> Clusters> PresentationTierCluster

Connected to 127.0.0.1:7001 Active Domain: mydomain Jun 22, 2001 3:52:18 PM EDT

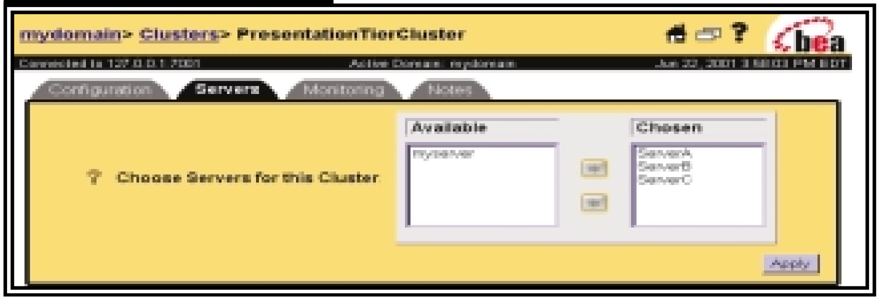
Multicast Address: 224.0.0.1

Multicast Send Delay: 12

Multicast TTL: 1

Adding servers to a cluster

FIGURE 6


mydomain> Clusters> PresentationTierCluster

Connected to 127.0.0.1:7001 Active Domain: mydomain Jun 22, 2001 3:58:03 PM EDT

Choose Servers for this Cluster

Available	Chosen
myserver	ServerA
	ServerB
	ServerC

Adding servers to a cluster

Setting Up a Simple Cluster

It's time to see how easy it is to set up a cluster. This last section will set up a cluster of three servers for handling business components whose external access is controlled by a fourth server acting as a proxy with the HttpClusterServlet. The business content will consist of a servlet, which will display the session ID associated with each request that illustrates the in-memory replication mechanism described earlier.

The entire domain for this exercise is available for download at the Web site (www.sys-con.com/weblogic). The downloaded domain files will have the cluster, business components, and proxy configured for some default IP addresses and ports. You can set up your cluster by following the steps listed here or by installing the domain download and remapping IP addresses to match those on your machine.

BASIC CLUSTER CONFIGURATION WITH THE ADMINISTRATION CONSOLE

First, using the Administration Console, we must create our server and cluster profiles and tie the former to the latter.

To create the server profiles, right click on the Servers folder and select "Configure a New Server." In the Configuration->General pane, fill in and submit the Name, Listen Port, and Listen Address for each server. Remember that the clustered servers must share the same Listen Port and have unique Listen Address settings. Figure 3 has a screen shot of what our screen looks like when filled out. Please note that while the Listen Address could be either an IP address or host name the use of the latter is necessary to run this example. To map IP addresses to host names on your individual computer, add entries to C:\WINNT\system32\drivers\etc\hosts.

To create the cluster profile, right click on the Clusters folder and select "Configure a New Cluster." In the Configuration->General pane, fill in and submit the Name and Cluster Address for the cluster. Set and apply the Multicast Address in the Configuration->Multicast pane. Finally, add the clustered servers to the Chosen column in the Configuration->Servers pane. Figures 4, 5, and 6 are screen shots of what our cluster configuration looks like when filled out.

All of the configuration data used for the business servers, proxy server, and cluster is summarized in Table 1.

SETTING UP THE HTTPCLUSTERSERVLET

It's time to configure the front-end load-balancing mechanism. This example will use the HttpClusterServlet running in its own instance of WLS. In the web.xml file of the proxy server's default Web application, XML elements are added to:

- register the HttpClusterServlet with its fully qualified class name;
- initialize the parameter defaultServers with the list of servers participating in the cluster;
- determine which URLs the servlet will intercept.

Listing 1 shows the web.xml configuration we used. The listing may be found at www.weblogicdevelopersjournal.com.

DEFINING BUSINESS CONTENT AND A FAILOVER MECHANISM

It's now time to build and deploy the

SimpleServlet, which displays the current contents of a session ID in two parts: the unique identifier (by default, 52 bytes in length) and the routing data (see Listing 2).

In the web.xml file of SimpleWebApp, the clustered Web application, XML elements are added to register SimpleServlet with its fully qualified class name and provide a single URL mapping element to allow invocation of the servlet through the alias simple. Listing 2 contains the source code for the SimpleServlet servlet. In the weblogic.xml file of the application, XML elements are added to set the session-related property PersistentStoreType to replicated. This is the only parameter required to enable in-memory replication. Listing 3 lists the configuration files for the SimpleWebApp.

The last step for configuring the test application is to deploy it to the cluster. Deployment to a cluster can be accomplished in the Administration Server:

1. Select the SimpleWebApp node from the navigation pane.
2. Select the Targets->Clusters pane.
3. Move the appropriate cluster name (in our case, PresentationTierCluster) from the Available list to the Chosen list and click Apply.

STARTING THE CLUSTER AND INTERPRETING RESULTS

To start the cluster:

1. Start the administration server with the startWebLogic.cmd script.

Start each of the clustered servers with the startManagedWebLogic.cmd script, with two command-line parameters. The first parameter is the name of the managed server (i.e., Server A, Server B, or Server C). The second parameter is the URL of the administration server (in our case, http://127.0.0.1:7001).

In order to determine if each server successfully joined the cluster, you can first check the standard output or server log for the text "Starting Cluster Service." For more detailed monitoring information, click on the PresentationTierCluster node in the navigation pane, tab over to Monitoring, and click on the "Monitor server participation in this cluster" hyperlink. The displayed table provides valuable information on the number of packets sent/received as well the role of each server with respect to in-memory replication.

If the table does not show three servers receiving updates, try the following troubleshooting measures:

- Check your start script command line parameters for typos.
- Verify that there are no physical network problems with the PING option of the Java application weblogic.Admin.
- Verify that multicast communications are working by using the Java application utils.MulticastTest.
- If the multicast utility fails, confirm that the selected multicast address falls within the valid range (224.0.0.1 – 239.255.255.255) and that no other application on the network is using that address.

If none of these measures shed any light on the encountered problems, consult Appendix A, "Troubleshooting Common Problems" of the document "Using WebLogic Server Clusters."

2. Start the proxy server with the startManagedWebLogic.cmd script, with the following parameters: ProxyServer and http://127.0.0.1:7001).

To test the sample application:

1. Start a Web browser and disable cookies.
2. Go to http://192.168.1.103:9001/SimpleWebApp/simple. If everything is working properly, the session ID and routing data will be displayed in your browser. Embedded within the routing data are aliases for the servers hosting the primary and back HttpSession instances.
3. Reload the servlet multiple times. With each reload, both the session ID and routing data change. Since the browser is not sending the cookie along with each HTTP request, the WLS servlet container always treats the user as a first-time visitor to the site. Note that the use of the round-robin load-balancing algorithm is evident in the server aliases displayed in the routing data.
4. Now, enable cookies in your browser and reload the servlet several times. With each reload, the session ID and routing data stay the same. Since cookies are being sent with each HTTP request, the proxy server is able to use the routing data to direct the traffic to the server hosting the primary HttpSession instance.
5. To demonstrate failover, bring down the clustered server whose alias appears first in the displayed routing data and reload the servlet. Observe that the session ID, which represents that application state, did not change, but the routing data has been modified. The server alias has moved forward, representing a promotion of the backup server to primary, and a new second alias is specified, representing the selection of a new backup server. Bringing down a second server will result in the last remaining clustered server being promoted to primary with the text "None" being inserted into the routing data since there are no candidates for backup.
6. As a final test, restart the dead servers and reload the servlet. The routing data will change to reflect the selection of a new backup server demonstrating the dynamic nature of cluster membership.

Conclusion

Whew! Granted, that's a lot to swallow, but WLS clustering is straightforward. Its flexibility comes from the number of integration touch points that it offers to architects of e-business systems. Future articles will discuss the implications of EJBs in a cluster and clustering best practices, so keep practicing and reading!

EJB TIPS

—continued from page 51

the finder, this example does N+1 database hits. The finder hits the database and each of the subsequent getSalary callbacks hits the database.

By now, you've probably guessed that we'll try to execute the finder and the getSalary methods within a single transaction. Now, how many databases accesses will occur? The answers may surprise you.

If Employee is a CMP entity bean and the finder and subsequent getSalary methods occur in the same transaction, there can be one database access. The finder executes a select query which loads the associated primary keys and the other Employee fields. These prefetched beans are entered into the EJB cache and the getSalary method calls read directly from the memory cache. This is the finders-load-bean option in the weblogic-ejb-jar.xml. It is enabled by default, and it allows finders to prefetch additional data into the EJB cache. In this common case, it reduces the database access from N+1 to just 1.

What if the Employee is a BMP entity bean? It may surprise you, but there will still be N+1 database hits. The BMP entity bean will implement an ejbFindEmployeesWithSalaries GreaterThan method in the bean class to return a Collection of primary keys to the container, but it cannot prefetch into the cache. Each getSalary call then produces an ejbLoad call and another database hit.

Finders prefetching beans is a big performance advantage of CMP, and in general CMP (especially EJB 2.0's CMP) entity beans outperform BMP entity beans.

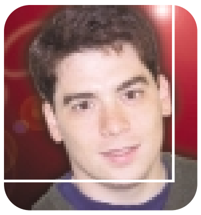
Conclusion

I hope these tips help you get the best performance out of your entity beans. We'll be looking at other performance tips and some of the new features in WebLogic Server 6.1 and EJB 2.0 in an upcoming column.

Message Driven Beans

THE MOST POWERFUL NEW SERVICE AVAILABLE

BY **SAM PULLARA**



WOW. THAT'S WHAT I HAVE TO say when I look back at where WebLogic began and where it has gone since then. When I started working at WebLogic, the only core people there were the founders, the president, an administrative assistant, and a lone sales person. Of course, back then our big moneymakers were the JDBC drivers. We always said, "You know, in two months, Oracle/Sybase/Microsoft is going to come out with a great native database driver and we won't be able to make any money in this business. We have to start selling our server." Fortunately for us, we're still able to sell those JDBC drivers, although they're a very small fraction of sales. WebLogic has come a long way in becoming a standards-based, fault-tolerant, scalable platform for building enterprise Java applications. Even our tag line back in those days was prophetic: "WebLogic, Elevating Java to the Enterprise."

So, now developers can buy our server off the shelf, use any tutorial or book that describes J2EE, and build incredible applications without having to build the infrastructure. There were times in the development of WebLogic where I'd step out of server-development mode and sit down and write an application. Inevitably it would lead me to trying to add more services to the platform so others didn't have to build it themselves. It's not true anymore. All the services to write real applications are handed to you – now if only it would architect the application!

Now, I'm sure all of you have used servlets and JSP. Many others have also used EJB. But I'm going to focus on what I believe is the most powerful new service available in WebLogic, Message Driven Beans (MDBs). Ah, the MDB is a beautiful thing. You create a destination (queue or topic), deploy a class that implements MessageDrivenBean and MessageListener, and BLAM! You have a unique application programming model that gives you very powerful messaging infrastructure. The combination of the EJB component model and the JMS messaging model gives you a tool that can be used effectively in a wide variety of applications. For instance, using this programming model, it was very easy to use JavaMail to build an e-mail–JMS bidirectional gateway. Drop a mime message on a queue, it gets sent out as an e-mail message, with retry! Receive a message in your mailbox, it appears as a mime message on a JMS queue in your server. Imagine the possibilities. Message forwarding, filtering, alerts, acknowledgments, all straight to your J2EE application. Combine it with a JSP tag for putting e-mail on a queue, and you get a pretty effective SPAM machine, not that I suggest you do that. E-mail address verification, e-mail–based monitoring, and server control. The best part about it is that the MDB gives you the power to implement gateways like this easily and with all the guarantees that JMS can provide. Another exciting usage of the message driven bean is integration with the all-new and shiny Web service. Business-to-business Web services will have to rely on constructs like the MDB to ensure that messages queued on one side arrive safely on the other in the face of network outages and error conditions. Combined with our out-of-the-box support for destination-based Web services, you can get a system like this up and running in no time, including the integration with partners that are using .NET, Apache, and many more.

In each issue I'm going to focus on a specific WebLogic technology and an associated cutting-edge technology, and how you as a developer can integrate the two and what you can expect in the future. I hope that today's rant from the soapbox encourages you to investigate the new features in WebLogic and to view them in the context of current technology.

AUTHOR BIO...

Sam Pullara has been a software engineer at WebLogic since 1996 and has contributed to the architecture, design, and implementation of many aspects of the application server.

CONTACT: sam@bea.com

wireless EDGE conference&expo

INTERNATIONAL
WIRELESS BUSINESS & TECHNOLOGY
CONFERENCE & EXPO
CONFERENCE & EXPO
CONFERENCE & EXPO

Plan to Exhibit

Provide the Resources to Implement Wireless Strategy

The conference will motivate and educate. The expo is where attendees will want to turn ideas into reality. Be ready to offer solutions.

Shaping Wireless Strategy for the Enterprise

Santa Clara, CA | **May 7-9, 2002**

Wireless Edge will provide the depth and breadth of education and product resources to allow companies to shape and implement their wireless strategy. Developers, *i-technology* professionals and IT/IS management will eagerly attend.

Plan to Attend the 3-Day Conference

WHO SHOULD ATTEND

Mobile & Wireless Application Professionals who are driving their enterprise's wireless initiatives:

- Program Developers
- Development Managers
- Project Managers
- Project Leaders
- Network Managers
- Senior IT and Business Executives

Conference Tracks

Track One: Development	Track Two: Connectivity	Track Three: Wireless Apps	Track Four: Hardware	Track Five: Business Futures
WAP	Smart Cards	Education	Cell Phones/ WorldPhones	Wireless in Vertical Industries
i-Mode	Wireless LANs incl. Bluetooth	Health Care	PDAs	The WWWW
Bluetooth / 802.11	UMTS/3G Networks	Entertainment	Headphones/ Keyboards / Peripherals	Unwired Management
Short Messaging	Satellite Broadband	Transport	Transmitters Base Stations	From 3W to 4W: Issues and Trends
Interactive Gaming		Financial Services	Tablets	"Always-On" Management
GPS / Location-Based		Supply Chain Management		Exploiting the Bandwidth Edge
Wireless Java				Unplugged Valueware
XML & Wireless Technologies				Wireless Sales & Marketing

FOR INFORMATION CALL

201 802-3084

SPEAKER PROPOSALS INVITED

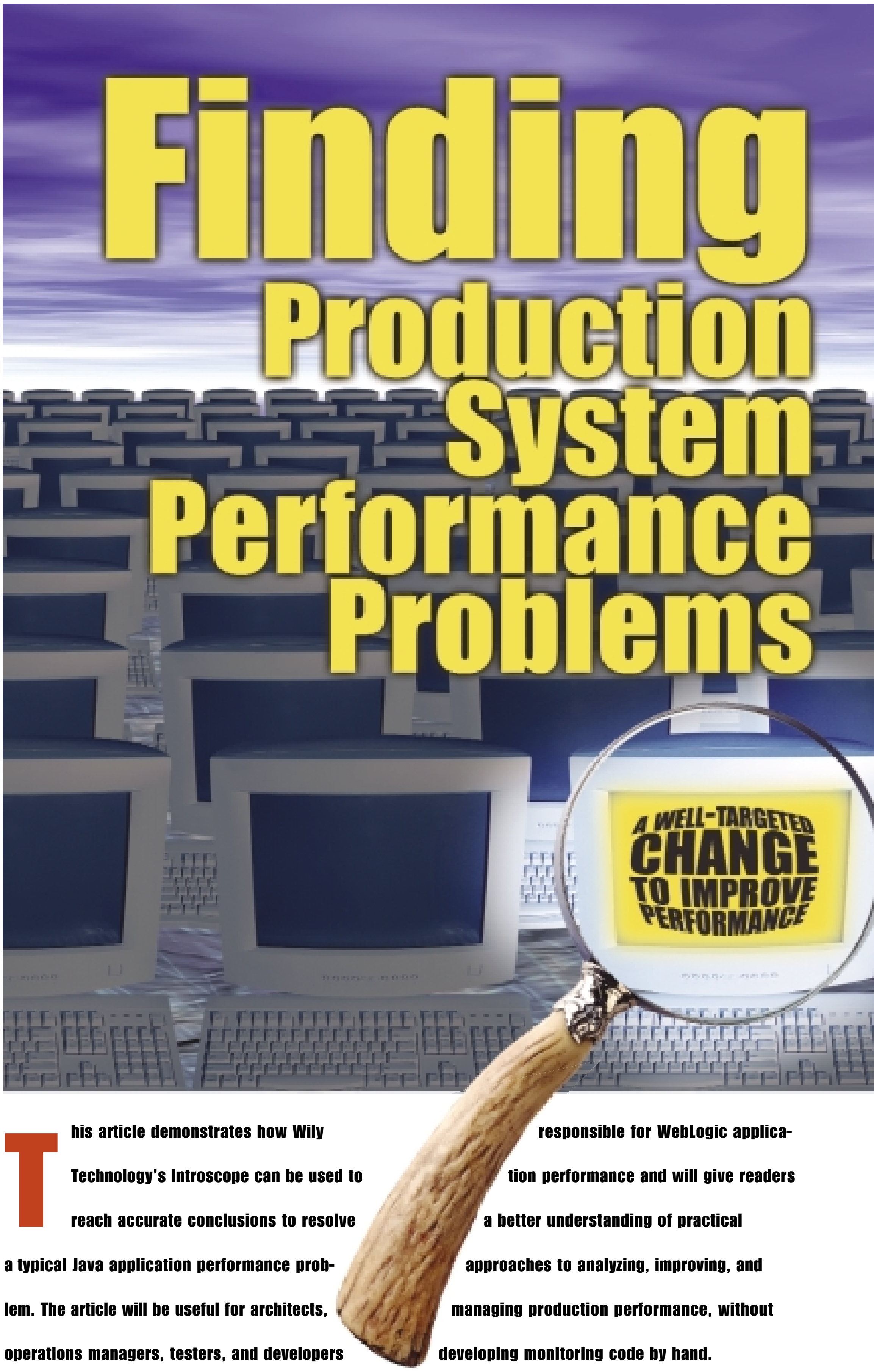
WWW.SYS-CON.COM

SHAPE YOUR WIRELESS STRATEGY! SAVE THE DATES!

EXCLUSIVE SPONSORSHIPS AVAILABLE

Rise above the noise. Establish your company as a market leader. Deliver your message with the marketing support of

SYS-CON MEDIA
SYS-CON EVENTS



Finding Production System Performance Problems

BY
CARL SEGLEM

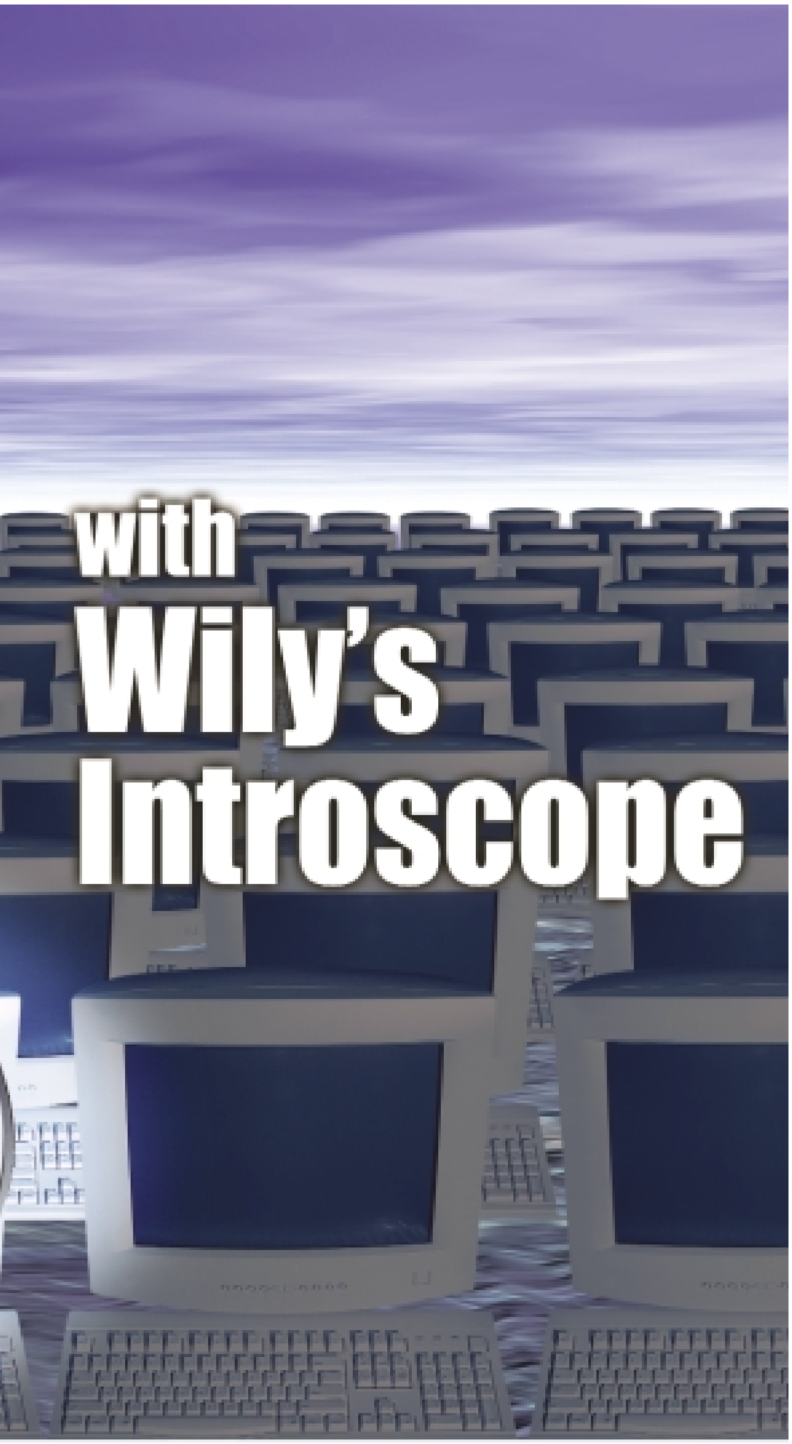
AUTHOR BIO...

Carl Seglem is a member of the Wily Technology Services team and has worked with Introscope at dozens of Fortune 1000 customers. Before joining Wily, he worked on information systems development and management at Scudder Kemper Investments and KPMG.

CONTACT...

carl.seglem@wilytech.com

This article demonstrates how Wily Technology's Introscope can be used to reach accurate conclusions to resolve a typical Java application performance problem. The article will be useful for architects, operations managers, testers, and developers responsible for WebLogic application performance and will give readers a better understanding of practical approaches to analyzing, improving, and managing production performance, without developing monitoring code by hand.



A business-to-business catalog and ordering system had been running in production for several months without much promotion and had served increasing numbers of customers reliably and quickly.

Recently, though, new features of the system, such as the ability to evaluate alternative items while browsing the catalog, had been promoted by the marketing organization, which caused more intensive use of the system. Unfortunately, users complained about slow performance of the application while searching and browsing for items as well as while building an order.

Operations and business managers came to the development organization with a critical mission to resolve these performance problems quickly. While developers had many hypotheses about the source of the problem, they were frustrated that they had no effective way to find it. The company chose to use Wily's Introscope to tackle their problem.

Using Introscope in the load-testing environment the development group identified a number of bottlenecks and eliminated other possible causes of the performance problems. With this information, developers could quickly implement focused changes in the application code to improve the system's performance and eliminate the performance problems. The operations and development groups also realized that using Introscope to monitor applications in production would make both groups more productive, allowing them to avoid and fix performance problems before they adversely affected their customers.

The System and Its Symptoms

UNDERSTANDING THE SYSTEM

This business-to-business system is a moderately complex application developed by a third party as a work-for-hire and turned over to the company to manage and maintain. The System Architecture diagram (Figure 1) gives a graphic representation of the parts of the system and how they interact. All of the components in the diagram are Java components.

The **Controller** receives requests for application services from the system and routes them to the appropriate components. It also manages user permissions and profiles, relying on an external **Authentication Service**.

The **Catalog Browser** implements business logic relating to searching for and browsing items. It relies on the **Customers** subsystem to retrieve information related to customers, such as preferred brands and pricing models; the **Item Catalog** subsystem for detailed item information, including pictures and descriptions, pricing, and combinations; and the **Inventory** system to show item availability while reviewing product information.

The **Order Builder** might also be called the shopping cart component. It combines user selections of products, quantities, and destinations with the Customers' pricing models, Item Catalog's prices and combinations, and the Inventory's on-hand information to create an order.

When an order is built, the **Order Placer** does the work of committing the order. It checks a Credit Verification Service if the customer's profile indicates it, updates the Customers and Inventory subsystems with the new order information, and has the Orders subsystem begin the process of fulfilling the order and billing the customer.

The **Customers**, **Item Catalog**, **Inventory**, and **Orders** components map the system's Java representations onto the back-end systems' representations and act as clients of the back-end systems.

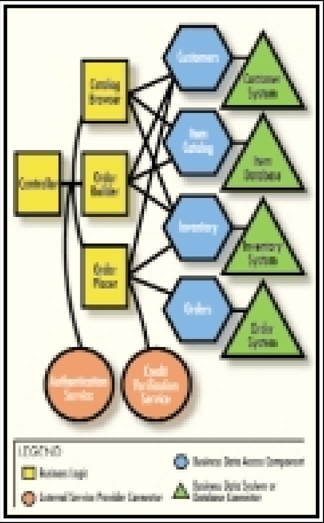
The **Customer System**, **Item Database**, **Inventory System,** and **Order System** exist in the Java system as connectors to those back-ends provided by their suppliers. The actual back-end systems run on different platforms and are shared by different systems, such as point-of-sale systems.

SEEING THE SYMPTOMS BUT NOT THE CAUSE(S)

The development organization has a load-testing environment, which allowed them to reproduce the problems that only appear under heavy load. However, because the system has many interacting components, which could not be measured directly, they could not isolate the cause of the problem, only verify its existence under certain conditions.

The back-end systems used by the production application are

FIGURE 1



System Architecture diagram

also used by the load-testing environment, as well as by many other systems, such as point-of-sale and telesales systems. Because they are shared, moderately used, and performing reliably, monitoring them directly does not provide useful information to troubleshoot the Java application's problems.

Individual developers frequently use a profiler to get very detailed information about the code they have written. However, because running the application with the profiler slows the application tremendously and produces enormous amounts of trace data, it has not proven useful to understand this production performance problem under load.

The development organization considered writing their own logging code into the application, but they decided not to for several reasons. Foremost is the great cost to the development organization in taking their developers' time away from creating new code to provide business value and occupying it with writing troubleshooting code, and building the infrastructure to store and present it. Other reasons included the risks of introducing and managing code changes in so many parts of the application and the damage to developers' morale from being assigned "grunt" work.

At the same time, there were many hypotheses about the source or sources of the performance problems. Some said it was the back-end systems' slow response, others believed that faster server hardware was needed, and others attributed the problem to the Web server.

In short, the development organization needed a way to get component-level performance information from the application while running under load without substantially changing its performance characteristics and without having to write it themselves. They also needed all this right away, since customer complaints were increasing daily. They discussed these needs with their BEA representatives who suggested that they contact Wily Technology (a BEA partner) about their product Introscope.

Introscope

COMPONENT-LEVEL MONITORING

Introscope provides component-level performance information about live production Java applications as well as applications running under load in a testing environment. It monitors any Java application running in any contemporary JVM (JDK 1.1.3 or later) on any hardware and operating system platform. Moreover, installation and administration of Introscope with WebLogic Server 5.1 and later is particularly easy with a feature called AutoProbe Integration.

Out of the box, Introscope monitors many common Java and J2EE components such as servlets, JSPs, EJBs, and JDBC and Socket activity. In addition, users can configure Introscope to monitor any class or method that they have built themselves or integrated from a third party using the Custom Tracing features. Users can also change the components they are monitoring even after the application is deployed, as their needs for performance information change. More importantly, monitoring choices are made without the need to access or change source code.

Introscope measures the average response time and the responses per second for most of the components it monitors. For other components, measurements include the bytes per second coming into or leaving the Java system and CPU utilization. In addition to each component's individual performance information, it keeps track of component interaction and attributes the performance of each to the component that caused or called it (a feature called "Blame Technology"). These performance measurements are useful for understanding how an application's components are performing while under load. The "Blamed" measurements make bottlenecks in component interactions easy to identify.

Introscope uses a number of techniques to ensure that the overhead of collecting performance information remains low. Introscope is selective about the components it monitors, and places lightweight monitors on relatively heavyweight component activity. The Introscope Agent collects summary information about component performance and reports that information asynchronously to a separate Enterprise Manager component, which handles more CPU-intensive tasks such as storing the data and making data available to the Workstation, Introscope's GUI.

HISTORICAL DATA STORED FOR ANALYSIS AND REPORTS

Introscope stores performance data in a JDBC-accessible database and/or comma-separated value (CSV) text files. The user controls exactly which data is stored and the frequency at which it is recorded. Once stored, the historical data can be viewed in the Workstation, or by using any technique that can query or report on the JDBC database or CSV files. Introscope includes sample component performance, service-level, and capacity-planning Crystal Reports.

ALERTS FOR OPERATIONS

Since Introscope is designed to manage Java systems in production, it can perform actions when performance measurements cross user-defined thresholds. Actions commonly triggered by Alerts include sending an e-mail, showing a dialog box in the Workstation, sending a message to a pager, writing to a log file, reconfiguring or restarting an application, and sending a message to another enterprise management system. In addition, an Introscope Alert can trigger any executable or shell script.

CUSTOMIZABLE VIEWS

The Workstation is an application that allows users to view and manage their systems' component performance. Particularly useful is the ability for users to create customized Dashboards to present performance data graphically for different users' needs. One Dashboard might show an overview of a system with colored lights indicating system status, while another might show detailed performance information for a component and the services it uses.

MONITORING CLUSTERS

Application instances can be monitored individually or as a cluster. Many Agents (whether on one machine or many and whether in a cluster or working as different tiers) can report performance information to the same Enterprise Manager. The data from each Agent is handled separately but can easily be monitored by an Alert or displayed together in a Dashboard. Additionally, aggregates for a cluster can easily be set up, for example, to provide the combined average response time for a Servlet in all instances in a cluster.

The Approach

The development and operations groups arranged for a Wily performance consultant to come in for five days and work with Introscope on their system in their environment. The goals of the work were to improve the system's performance and understand how Introscope can manage this and other systems in production.

INSTALLING INTROSCOPE WITH AUTOPROBE

On the first day, Introscope was set up quickly on the WebLogic machines in the load-testing environment by using the AutoProbe Integration feature with WebLogic Server. The Enterprise Manager was installed on a separate, shared, low-end box in the testing environment. An existing database server had database structures and a user added for Introscope to use. The Workstation was installed on several machines, including one in the testing environment, one in the operations management environment, and two in the development organization. That afternoon, the team was already viewing live component information with Introscope.

CUSTOMIZING THE MONITORING ENVIRONMENT

As with most systems, this one is made up of both J2EE components (which Introscope monitors out-of-the-box) and a number of custom components (which must be configured for Introscope to monitor). Introscope uses text files to configure which custom components to monitor, referencing the package, class, and method names of the primary ways the components are accessed. Based on discussions with the company's system architect, the package, class, and method names for Business Logic, External Service Provider, and Business Data Access Components were collected and used to create directive files for Introscope. This code snippet is a line from a custom directives file:

```
TraceOneMethodOfClass:
com.company.onlinesales.logic.OrderBuilder addItem
BlamedMethodTimer "Business Logic|Order Builder:Average
Response Time (ms)"
```

The application server was restarted with this updated configuration and the performance information about these components became visible in Introscope.

MONITORING COMPONENTS AND INTERACTIONS UNDER LOAD

Moderate load was run against the system to begin to understand what performance information could be displayed and how it is represented.

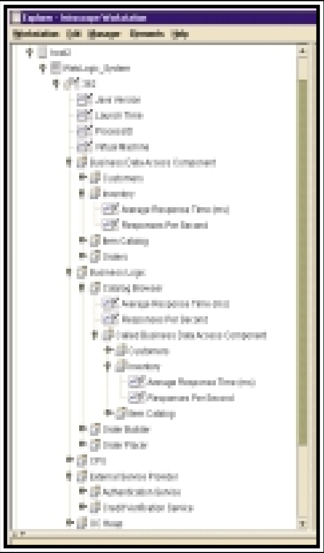
From the load generator's point of view, the application behaved the same as before Introscope was introduced. There was no discernible difference in the performance of the application running under load with Introscope. This finding was crucial. First, the development organization needed to be confident that analysis with Introscope would not change the nature of the performance problem. Second, the operations group and system business sponsor would balk at the possibility of introducing large overhead, which would require additional server investment.

In Introscope, a component hierarchy of the application is visible in the Explorer window. It shows both the performance of individual components specified earlier in the configuration, as well as the performance of related components that are involved during the course of a component's work.

Figure 2 shows the performance information for the Catalog Browser and the Inventory components by themselves, as well as the Item Catalog performance *when it is working on behalf of* the Catalog Browser. As previously mentioned, Introscope's ability to associate performance information about one component with other components is called Blame Technology. Because the Inventory component works for several other components, it is extremely useful to be able to differentiate the performance of each component by its context. Without this feature, it would be difficult to find problems and bottlenecks that are caused by particular components' interactions rather than their aggregate performance. Another useful aspect of the Blame Technology is that Introscope does not have to be configured in advance as to which interactions to monitor.

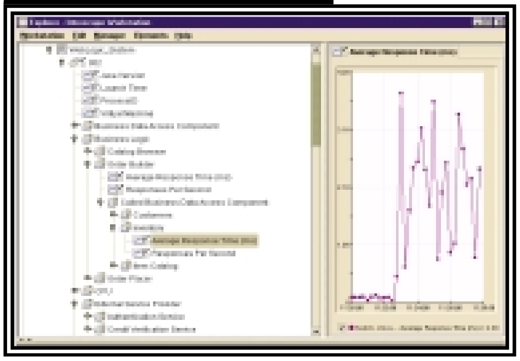
Browsing Introscope's Explorer tree confirms that under heavy load the Catalog Browser and Order Builder components respond slowly. New information is now apparent: the Item Catalog and Inventory components are busier and slower,

FIGURE 2



Performance information for the Catalog Browser and Inventory components

FIGURE 3



Inventory Average Response Time

while other components do not appear to slow down much. Figure 3 shows the Inventory Average Response Time with moderate and heavy load.

Looking at the performance information for the monitored components, it is evident that in order to understand the performance of the Item Catalog and Inventory components, it is also important to monitor the Business Data System or DB components to see how the Item Catalog and Inventory components are using them.

Being able to view different performance information side-by-side would make this correlation and analysis easier than browsing in the Explorer. Introscope's Dashboards show selected performance information on the same screen. That customization is discussed below.

HOMING IN ON THE PROBLEMS

On day two of the project, Introscope's directive files were updated to include the Business Data System or DB components and the application was restarted and run under load again. Looking in the Explorer, the newly-configured components were shown as top-level components, as well as called resources under the Business Data Access Components.

To create Dashboards to conveniently show this information side-by-side, component metrics were dragged from the Explorer tree onto new Dashboards, automatically creating graph views, which were labeled and organized in Panels. The Business Logic components overview is shown in Figure 4. It shows the average response times and responses per second of the four Business Logic components. In this Dashboard, which shows the transition from moderate to heavy load, the much slower response times of all the Business Logic components except the Order Placer is clearly evident.

On the Catalog Browser Dashboard, shown in Figure 5, it appears that under higher load the Catalog Browser responds more slowly because it relies more frequently on the Item Catalog and the Inventory components, which are also much slower.

The analogous pattern is evident on the Order Builder Dashboard: the Item Catalog and Inventory components are both busier and much slower under heavier load.

From the Item Catalog Dashboard, it appears that under higher load, the Item Database component is also used much more frequently, but responds quickly. This implies that the bottleneck is not in the back-end system but in the Item Catalog component or the way it is used. A corresponding pattern is evident on the Inventory Dashboard, which shows the Item Database is used more often while still responding quickly.

The Results

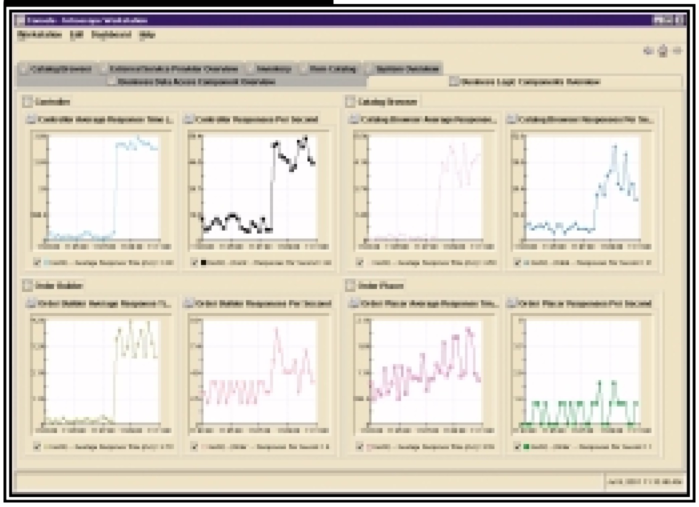
IDENTIFYING AND FIXING PROBLEMS

On day three of the engagement, conclusions about the sources of the performance problems became clear.

External circumstances

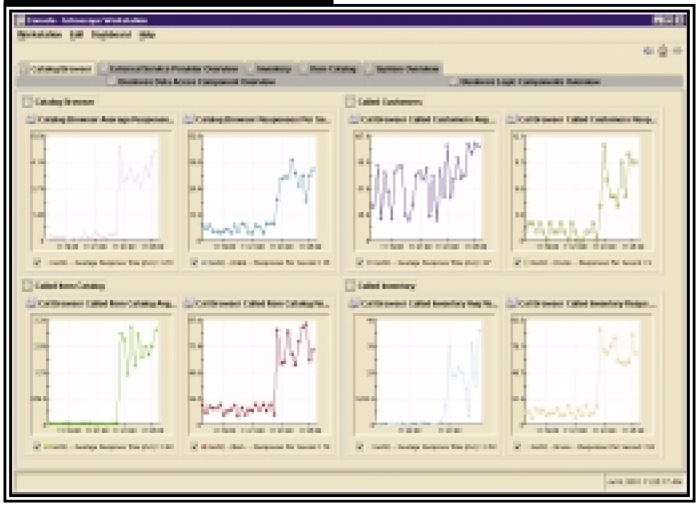
As suggested initially, there were two primary external contributing circumstances causing the slowdown: more users and more searches for related items by the Catalog Browser. The relationship between the number of users and the

FIGURE 4



Business Logic components overview

FIGURE 5



Catalog Browser Dashboard

activity of the two bottleneck components was expected to be linear, and previous log analysis suggested this was true. However, with promotion of the "find related" feature, usage of the Inventory and Item Catalog components increased at a much greater rate than that of the number of users, which led to stress on the system and general application slowdown.

Inventory and Item Catalog component bottlenecks

The analysis showed that the Inventory and Item Catalog components were accessed every time the Catalog Browser returned product description information for an item and every time the Order Builder added an item to an order. The combination of additional users and their use of the "find related" feature meant that there were many more Inventory and Item Catalog look-ups. The number of Inventory System and Item Database lookups was also greater, but the Inventory System and Item Database themselves did not slow noticeably. This suggested that the Inventory and Item Catalog components should be used less often, or they could cache their results in order to respond more quickly.

Eliminating alternative explanations

Many other possible sources of problems (other components, back-end systems, networking, memory) were quickly eliminated as suspects, reducing both the time taken to come to conclusions and the amount of work needed to reach them.

NEW RELEASE MOVED INTO PRODUCTION

Based on the conclusions made possible by Introscope, useful, localized changes could quickly be made in several components with a high degree of confidence that those changes would have a substantial beneficial effect on the application's performance under load.

Operations considers Introscope

On day four (while development worked on system code changes), Wily's performance consultant worked with operations line and management personnel to understand how Introscope could be deployed and used in the company's production environment to monitor their various Java systems. Sample Alerts and Dashboards were set up and test events were sent into the company's existing management framework.

Figure 6 shows one of the Alerts that operations set up. Figure 7 shows a resulting Alert message and the detailed information that Introscope provides when thresholds are crossed.

The operations group has an operations center in which monitoring consoles run all the time. Figure 8 shows an example of a Dashboard that might be displayed in the operations center, providing system status information at a glance.

Operations also spent some time understanding the database structure in which Introscope stores historical data. Some sample reports were run which showed how Introscope could be used both as a source of benchmarking reports during performance testing and for service-level and trend analysis reporting over time.

With Introscope's functionality, the operations and development group expect to be able to better understand how their systems are performing, respond to problems quickly when they occur, and avoid involving the development group except in exceptional circumstances. When these circumstances do occur, operations and development are confident that they will be able to share a view into the running application and avoid guessing and finger pointing about what the causes of problems might be.

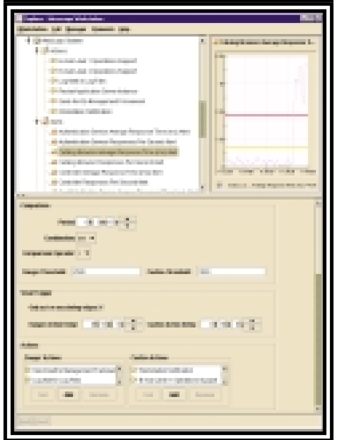
Improved performance

By the end of the week, development had made the indicated changes and began to test the performance under load. Preliminary results indicated that the large slowdowns had disappeared. Preparations were begun to deploy the updated application to production with Introscope.

Conclusion

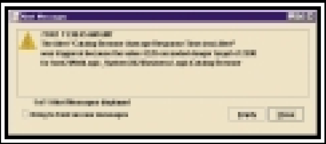
The causes of the performance problems were quickly identified. Much time-consuming investigation and potentially expensive purchases of server hardware were avoided. The development group could promptly make well-targeted changes to improve the application's performance for their customers. To learn more about Introscope call 1-800-GETWILY or visit www.wilytech.com

FIGURE 6



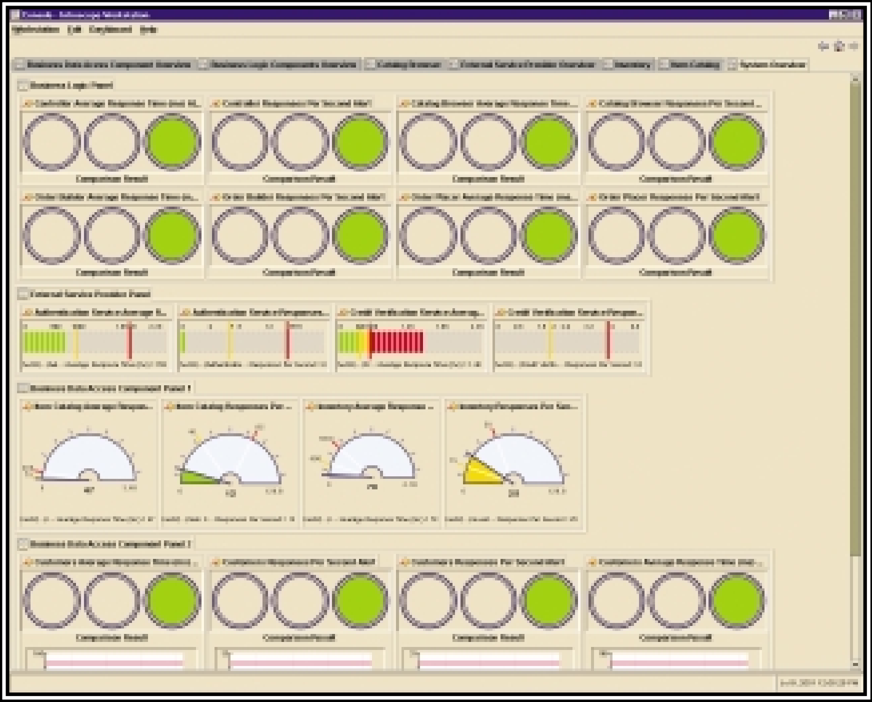
Alert set up by operations

FIGURE 7



Resulting Alert message

FIGURE 8



Operations Dashboard

Applications offer the advantage of using server-side code

Asynchronous Messaging with WebLogic Server, Server Sessions, and MDBs

BY
BILL KEMP

AUTHOR BIO...

Bill Kemp is a developer relations engineer with BEA in Colorado Springs with 14 years of experience in software development and support. He is a Sun Certified Java Programmer and BEA WebLogic Certified Developer. Bill focuses on EJB, JMS, and JDBC.

CONTACT...

wkemp@bea.com

Asynchronous message processing is a requirement of many applications that need to send and receive messages in real time or near real time. The JMS 1.0.2 Specification defines the MessageListener interface that allows an application to receive messages asynchronously from a JMS Destination when a message is sent to that destination. An implementation of the MessageListener interface is typically a client application to a JMS provider that runs in a remote JVM.

The JMS Specification also defines an optional set of APIs, JMS Application Server Facilities, that may be implemented by an application server vendor to allow for asynchronous, concurrent processing of a subscription. The EJB 2.0 Specification provides the Message Driven Bean API (MDB) that allows a developer to create and deploy a component into an EJB container, and have the container invoke the MDB when a message is sent to a JMS Destination that the MDB is deployed to service. Both of these APIs provide a way to consume messages from a JMS Destination within the JVM and framework of an application server rather than as an external JMS client to a JMS provider. Consumers deployed in this fashion can take advantage of services provided by the application server, such as transaction and thread management.

WebLogic Application Server provides an implementation of the MDB API described in the EJB 2.0 Specification and the optional JMS Application Server Facilities described in Chapter 8 of the JMS 1.0.2 Specification. I'll discuss these APIs, and support for them within WebLogic Server, with a store and forward example that uses MDB and ServerSessionPool. The example code was developed and tested using WebLogic Server 6.0 Service Pack 2. A general discussion of JMS isn't provided here, but an excellent three-part article, authored by Dave Chappell, on the JMS API can be found in the March, April, and May 2001 issues of ***Java Developer's Journal*** (Vol. 6, issues 3-5).

JMS ServerSessions

Chapter 8 of the JMS 1.0.2 Specification defines three additional optional interfaces, implemented by an application server vendor, that provide a special facility for concurrent message processing. It further delineates the roles of JMS provider, Application Server, and Application regarding the implementation and use of these interfaces. The interfaces found in the javax.jms package that make up this part of the API are ServerSession, ServerSessionPool, and ConnectionConsumer. The spec also defines how an application server will use the javax.jms.Session interface to implement these special facilities.

A Session object provides a setMessageListener() and getMessageListener() method to allow a ConnectionConsumer to associate a MessageListener with a Session. The ConnectionConsumer can set the MessageListener with a message selector. It also provides a run() method that allows an application server to start a Session once messages are available for consumption on a Destination.

A ServerSession is implemented by the application server to provide the runtime support for a JMS Session. It maps a thread to a Session and executes the Session's run() method through its start() method. In addition to the start() method, the ServerSession defines a getSession() method that returns the JMS Session associated with the ServerSession to the caller.

The application server also provides a javax.jms.ServerSessionPool object that's used by the ConnectionConsumer to get a Session for message processing. The ConnectionConsumer uses the only ServerSessionPool method, getServerSession(), to acquire a ServerSession for consuming messages that have been produced at the Destination.

Finally, javax.jms.TopicConnection and javax.jms.QueueConnection provide a method, getConnectionConsumer(), that creates a ConnectionConsumer that's used by the application server to load messages into a Session, and start the Session indirectly by calling the ServerSession's start() method. Construction of the ConnectionConsumer associates it with a ServerSessionPool and a Destination.

WebLogic provides an additional class that isn't part of the specification, weblogic.jms.ServerSessionPoolFactory. A ServerSessionPoolFactory is a factory pattern used to create a ServerSessionPool and associate it with a Connection, a MessageListener, set the Session's acknowledgment type, and define the transaction semantics of the Session. Implementation of a ServerSessionPool is subject to interpretation and will have vendor-specific features, and as such, its life cycle is well-suited to the factory pattern should the implementation change or should multiple implementations be made available. The WebLogic Server ServerSessionPool factory is looked up through JNDI using the name "weblogic.jms.ServerSessionPoolFactory:<name>" where <name> is the name of the JMS Server where the ServerSessionPool will be used.

MDBs

Message Driven Beans are defined by the EJB 2.0 Specification and exist for the exclusive purpose of consuming messages from a JMS Destination. WebLogic Server 6.0 provides full support for the MDB portion of the EJB 2.0 API.

If you are a seasoned EJB developer, MDBs are surprisingly simple, and even if you are new to EJB development you can create and deploy MDBs to consume messages from JMS Destinations with little difficulty. This simplicity of development and deployment is, in my opinion, the strong suit of MDBs.

Unlike session beans and entity beans, an MDB doesn't require the development of an extension to the EJBHome interface or the EJBObject interface. The reasons are that an MDB's home interface isn't located by a client application through a JNDI lookup, its life cycle isn't controlled by a client application through the factory methods on a home interface, and no methods need to be exposed to a client application through a remote interface. All the bean developer needs to do is implement the MessageDrivenBean interface with its two methods, setMessageDrivenContext() and ejbRemove() and implement the MessageListener interface providing an onMessage() implementation.

The spec adds some new elements to the ejb-jar.xml deployment descriptor for deploying MDBs. Here's the element definition from the ejb-jar_2.0.dtd:

```
<!ELEMENT message-driven (description?, display-name?,
small-icon?,
large-icon?, ejb-name?, ejb-class, transaction-type,
message-selector?,
acknowledge-mode?, message-driven-destination?, env-
entry*, ejb-ref*,
security-identity?, resource-ref*, resource-env-ref*)>
```

The <message-driven> element is an optional element contained in the <enterprise-beans> element. It uses many of the elements previously defined for describing an EJB, such as <ejb-name> and <transaction-type>. The <transaction-type> element describes the transaction semantics of the bean. The allowable values are "Bean" and "Container", analogous to session beans. A value of "Container" tells the EJB container to call the MDB onMessage() method in the context of a transaction. This is recommended. When using container-managed transactions, a <container-managed> section in the <assembly-descriptor> is required to tell the container what transaction attributes are needed for the MDB when it's invoked. When using container-managed transactions, the allowable values for <trans-attribute> are "Required" and "NotSupported".

The new elements defined are:

- <message-driven-destination>
- <message-selector>
- <acknowledge-mode>

The <message-driven-destination> element is a stanza used to describe the characteristics of the destination the MDB is deployed to consume. It must contain <destination-type> and may optionally contain <subscription durability>. The allowable values for <destination-type> are "javax.jms.Queue" and "javax.jms.Topic". The allowable values for <subscription-durability> are "Durable" and "NonDurable", with "NonDurable" being the default.

Listing 1

```
<!DOCTYPE ejb-jar PUBLIC
"-//Sun Microsystems, Inc.//DTD Enterprise
JavaBeans 2.0//EN"
"http://java.sun.com/dtd/ejb-jar_2_0.dtd">

<ejb-jar>
 <enterprise-beans>

    <message-driven>
      <ejb-name>wldjAlarmReaderBean</ejb-name>
      <ejb-class>wldj.mdb.AlarmReaderBean</ejb-
class>
      <transaction-type>Container</transaction-
type>
      <message-driven-destination>
        <jms-destination-
type>javax.jms.Queue</jms-destination-type>
      </message-driven-destination>
    </message-driven>

 </enterprise-beans>

 <assembly-descriptor>

  <container-transaction>
    <method>
      <ejb-name>wldjAlarmReaderBean</ejb-name>
      <method-name>*</method-name>
    </method>
   <trans-attribute>Required</trans-attribute>
  </container-transaction>

 </assembly-descriptor>

</ejb-jar>
```

Listing 2

```
<?xml version="1.0"?>
<!DOCTYPE weblogic-ejb-jar
PUBLIC "-//BEA Systems, Inc.//DTD WebLogic 6.0.0
EJB//EN"
"http://www.bea.com/servers/wls600/dtd/weblogic-
ejb-jar.dtd">

<weblogic-ejb-jar>

  <weblogic-enterprise-bean>
    <ejb-name>wldjAlarmReaderBean</ejb-name>
    <message-driven-descriptor>
      <pool>
        <max-beans-in-free-pool>20</max-beans-in-
free-pool>
        <initial-beans-in-free-pool>20</initial-
beans-in-free-pool>
      </pool>
      <destination-jndi-
name>weblogic.wldj.AlarmQueue</destination-jndi-name>
    </message-driven-descriptor>
    <jndi-
name>weblogic.wldj.AlarmReaderBean</jndi-name>
  </weblogic-enterprise-bean>

</weblogic-ejb-jar>
```

Listing 3

```
package wldj.mdb;

import weblogic.rmi.RemoteException;
import java.util.Hashtable;
import javax.ejb.CreateException;
import javax.ejb.MessageDrivenBean;
import javax.ejb.MessageDrivenContext;

import javax.jms.*;

import javax.naming.Context;
import javax.naming.InitialContext;
import javax.naming.NamingException;

public class AlarmReaderBean
  implements MessageDrivenBean,
             MessageListener {

 final static String JNDI_FACTORY=
     "weblogic.jndi.WLInitialContextFactory";
 public final static String JMS_FACTORY=
     "weblogic.wldj.TopicConnectionFactory";
 public final static String CriticalTopic=
     "weblogic.wldj.CriticalAlarmTopic";
 public final static String NormalTopic=
     "weblogic.wldj.NormalAlarmTopic";

 protected TopicConnectionFactory tconFactory;
 protected TopicConnection tcon;
 protected TopicSession tsession;
 protected TopicPublisher tpublisher;
 protected Topic topic;
 protected InitialContext ic;

 public void onMessage(Message msg) {

  TextMessage tm = (TextMessage) msg;
  try {
```

A <message-selector> may be specified using message selector syntax described in the JMS Specification. Message selection with MDBs is discouraged, as it can be an expensive operation when there are many messages in a Queue or Topic. Message selectors on Queues are not recommended.

Message Driven Beans do not use the JMS message acknowledgment API. The container provides message acknowledgment for the MDB based on the value of <acknowledge-mode> in the deployment descriptor. The default mode is AUTO_ACKNOWLEDGE.

The weblogic-ejb-jar.dtd also defines a new element, <message-driven-descriptor>, that's used to specify WebLogic Server–specific properties of the bean at deployment time. It's defined as:

```
<!ELEMENT message-driven-descriptor (
 pool?,
 destination-jndi-name?,
 initial-context-factory?,
 provider-url?,
 connection-factory-jndi-name?
)
>
```

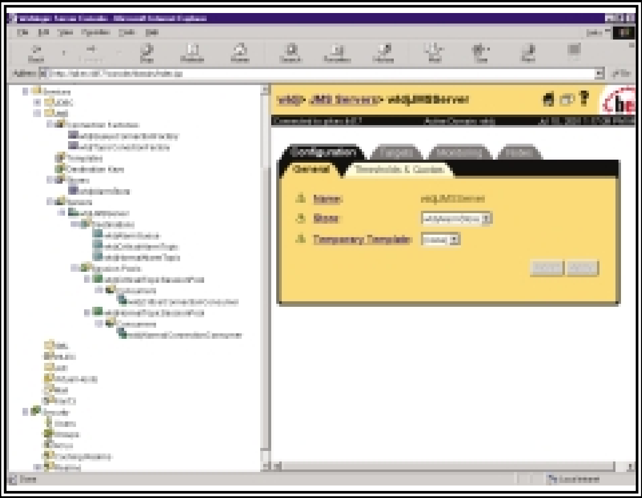
The <pool> element is used to tell the container how to pool instances of the MDB. The <destination-jndi-name> is the name of the Destination that the MDB will consume messages on. The <initial-context-factory> is the name of the JNDI InitialContext factory that will be used to get an InitialContext for looking up a ConnectionFactory. The <provider-url> is the URL of the InitialContext provider, and the <connection-factory-jndi-name> is the name of the ConnectionFactory that will be looked up.

Store and Forward Example

I created a small store and forward example to demonstrate the use of these two server-side techniques of asynchronous messaging. I developed the example by taking sample code provided with WebLogic Server and changing it slightly, and by using the WebLogic Server Admin Console to create the necessary JMS elements (see Figure 1).

The example uses a Queue and two Topic(s) to simulate a store and forward architecture used to route alarm messages. The Queue is consumed by an MDB that examines the header of a TextMessage for information that the

FIGURE 1



Console view of JMS server elements

```
    String level = tm.getStringProperty("level");
    String text = tm.getText();
    log("Received alarm message : " + text);
    log("Criticality = " + level);
    topic = null;

    if( level.equals("Critical") ) {
      log("Processing Critical Alarm");
      topic = (Topic)ic.lookup(CriticalTopic);
    } else if( level.equals("Normal") ) {
      log("Processing Normal Alarm");
      topic = (Topic)ic.lookup(NormalTopic);
    } else {
      log("Tossing Invalid Message");
      return;
    }

    tsession = tcon.createTopicSession(
              false, Session.AUTO_ACKNOWLEDGE);
    tpublisher = tsession.createPublisher(topic);
    tpublisher.publish(msg);

    tpublisher.close();
    tsession.close();
    }
    catch(Exception ex) {
      ex.printStackTrace();
    }
  }
}
```

Listing 4

```
package wldj.mdb;

import weblogic.rmi.RemoteException;
import java.util.Hashtable;
import javax.ejb.CreateException;
import javax.ejb.MessageDrivenBean;
import javax.ejb.MessageDrivenContext;

import javax.jms.*;

import javax.naming.Context;
import javax.naming.InitialContext;
import javax.naming.NamingException;

public class AlarmReaderBean
  implements MessageDrivenBean,
             MessageListener {

 final static String JNDI_FACTORY=
     "weblogic.jndi.WLInitialContextFactory";
 public final static String JMS_FACTORY=
     "weblogic.wldj.TopicConnectionFactory";
 public final static String CriticalTopic=
     "weblogic.wldj.CriticalAlarmTopic";
 public final static String NormalTopic=
     "weblogic.wldj.NormalAlarmTopic";

 protected TopicConnectionFactory tconFactory;
 protected TopicConnection tcon;
 protected TopicSession tsession;
 protected TopicPublisher tpublisher;
 protected Topic topic;
 protected InitialContext ic;

 public void ejbCreate () throws CreateException {
  log("ejbCreate called");
  try {
   ic = getInitialContext();
   tconFactory =
    (TopicConnectionFactory)ic.lookup(JMS_FACTO-
RY);
   tcon = tconFactory.createTopicConnection();
   tcon.start();
  } catch (Exception e) {
   log( e.toString() );
   throw new CreateException();
  }
 }
}
```

Listing 5

```
package wldj.jms;

import java.io.*;
import java.util.*;
import javax.transaction.*;
import javax.naming.*;
import javax.jms.*;
import weblogic.common.*;
import weblogic.jms.ServerSessionPoolFactory;

public class CriticalTopicSessionPool
  implements T3StartupDef
{
 private final static String
  SESSION_POOL_FACTORY=

"weblogic.jms.ServerSessionPoolFactory:wldjJMSServe
r";

  private T3ServicesDef services;
```

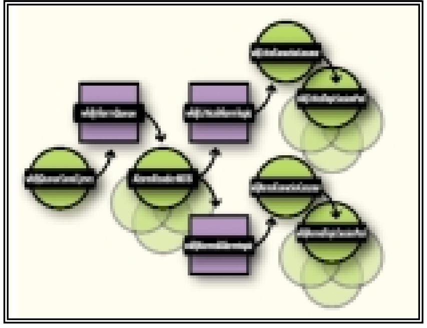
MDB uses to forward the message to the appropriate Topic. The header contains a criticality level for the alarm message used for routing. The two Topic(s) represent forward destinations for the two levels of alarm criticality. One Topic receives "Critical" alarms and the other Topic receives "Normal" alarms (see Figure 2).

Alarm Routing MDB

The "Alarm" Queue, wldjAlarmQueue, is a persistent Queue that's consumed by the AlarmReaderBean class that implements MessageDrivenBean and MessageListener. The AlarmReaderBean is deployed with container-managed transactions and a <transaction-attribute> of "Required". The ejb-jar.xml and weblogic-ejb-jar.xml deployment descriptors are shown in Listings 1 and 2.

The onMessage() method of this consumer is responsible for examining a user property named "level" in the message header and forwarding the message to the wldjCriticalAlarmTopic or the wldjNormalAlarmTopic (see Listing 3).

FIGURE 2



Schematic of alarm messages flow

Since the onMessage() method has to route the message to the appropriate Topic, it must have access to a TopicConnection to do so. To avoid looking up a TopicConnection every time a message is processed, the ejbCreate() method of the AlarmReaderBean performs a JNDI lookup on the wldjTopicConnectionFactory, creates a TopicConnection, and starts the TopicConnection. The ejbCreate() method is called only when the WebLogic Server EJB container creates the bean instance (see Listing 4).

When the onMessage() method has decided where to route the message, it creates a nontransacted AUTO_ACKNOWLEDGE TopicSession using the cached TopicConnection. It uses the TopicSession to create a TopicPublisher and then publish the message to the found Topic. The TopicPublisher and TopicSession are closed at the end of the method.

Consumer ServerSessions

Once the alarm messages have been routed to the wldjCriticalAlarmTopic and the wldjNormalAlarmTopic, they are processed by a MessageListener that has been associated with the Sessions in the ServerSessionPool. When the message arrives, the ConnectionConsumer assigned to the Destination uses the ServerSessionPool to acquire a ServerSession and a Session, loads the Session with a message (one or more), and starts the ServerSession. Let's examine the ServerSessionPool and ConnectionConsumer for the wldjCriticalAlarmTopic.

A ServerSessionPool and ConnectionConsumer can be created programmatically or by using the WebLogic Server Admin Console. Programmatically creating the ServerSessionPool and ConnectionConsumer can be done from a WebLogic Server client application or from a WebLogic Server Startup class.

Example code of creating and initializing a ServerSessionPool and a ConnectionConsumer in a Startup class is provided with the WebLogic Server distribution in WL_HOME/samples/examples/-jms/sessionpool and WL_HOME/samples/examples/jms/startup where WL_HOME is the directory where the WebLogic Server installation is located. A Startup class is a class that implements weblogic.common. T3StartupDef is deployed in a WebLogic Server server instance and is invoked by WebLogic Server at the end of the startup sequence to perform any task that logically needs to be performed at server startup, such as ServerSessionPool initialization. See listing 5 for the example modified for our application.

First, a Connection to the JMS Server must be acquired through a ConnectionFactory. I used a Topic, a TopicConnection, and a TopicConnectionFactory for this example. A default implementation of TopicConnectionFactory is provid-

```
 private TopicConnectionFactory tconFactory;
 private TopicConnection tcon;
 private TopicSession tsession;
 private Topic topic;
 private ServerSessionPoolFactory
         sessionPoolFactory;
 private ServerSessionPool sessionPool;
 private ConnectionConsumer consumer;

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

 public String startup(String name, Hashtable
args)
      throws Exception
 {
  String connectionFactory =
         (String)args.get("connectionFactory");
  String topicName = (String)args.get("topic");

  if (connectionFactory == null ||
      topicName == null) {
      throw new IllegalArgumentException(
         "connectionFactory="+connectionFactory+
         ", topicName="+topicName);
  }

  Context ctx = new InitialContext();
  tconFactory =
    (TopicConnectionFactory)ctx.lookup(
                       connectionFactory);
  tcon = tconFactory.createTopicConnection();
  topic = (Topic) ctx.lookup(topicName);
  tcon.start();
  sessionPoolFactory = (ServerSessionPoolFactory)
  ctx.lookup(SESSION_POOL_FACTORY);
  sessionPool = sessionPoolFactory.
  getServerSessionPool(tcon, 5, false,
                      Session.AUTO_ACKNOWL-
EDGE,

"wldj.jms.AlarmListener");
  consumer = tcon.createConnectionConsumer(topic,
"TRUE" ,

sessionPool, 10);
  return "Ready To Receive Messages";
 }
}
```

Listing 6

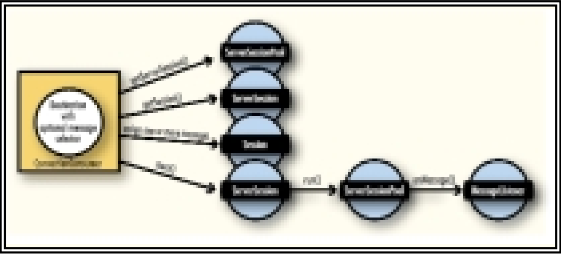
```
 public void send(String message)
      throws JMSException
 {
   msg.setStringProperty("level", "Critical");
   msg.setJMSPriority(9);
   msg.setText(message);
   qsender.send(msg);
 }
```

ed by WebLogic Server and can be found through a JNDI lookup. I created my own TopicConnectionFactory, wldjTopicConnectionFactory, with the same characteristics as the default.

Next, a JNDI lookup is performed on the Topic that the ConnectionConsumer will service. A JNDI lookup is performed to locate the ServerSessionPool-Factory, and from the factory a SessionPool is acquired using the getServerSessionPool() method. It's important to note that the SessionPool is created by passing a Connection, the maximum number of Sessions in the pool, a class that implements the MessageListener interface, a Boolean that specifies if the Sessions are transacted or nontransacted, and an acknowledgment mode to the getServerSessionPool() method. Finally, the ConnectionConsumer is created using the createConnectionConsumer() method on the Connection object, and is passed to the Topic, the ServerSessionPool, a message selector String, and the maximum number of messages that can be loaded into a Session. The message selector specified when the ConnectionConsumer is created using a String containing the message selector syntax defined in the JMS Specification. My example uses '"TRUE" as the message selector that selects all messages. More on this later.

The Startup class will execute at the end of WebLogic Server initialization and prepare a ConnectionConsumer that's ready to load a JMS Session with messages from the Topic and start the Session so that the messages can be consumed by the specified MessageListener. When a message arrives at the Destination, the ConnectionConsumer gets a ServerSession, uses the ServerSession to get the JMS Session, loads the JMS Session with one or more messages, and calls the start() method on the ServerSession. The ServerSession calls the run() method on the JMS Session, which then calls the onMessage() method of the associated MessageListener, passing it the Message that needs to be consumed. The role of the ConnectionConsumer is diagrammed in the JMS Specification and reproduced in Figure 3.

FIGURE 3



Role of ConnectionConsumer in ServerSessionPool, reproduced from JMS Specification

I find the programmatic technique of creating a ServerSessionPool and ConnectionConsumer much more complicated than using the WebLogic Server Admin Console. They can be created in the Admin Console after the JMS Server has been created. Use the SessionPools tab under JMS->Servers>yourServer NameSessionPools. Select "Create a new JMS Session Pool…". That will take you to the frame

WLDJ ADVERTISER INDEX

ADVERTISER	URL	PHONE	PAGE
Altoweb	www.altoweb.com	650-251-1500	19
BEA	http://developer.bea.com	800-817-4BEA	2-3, 30-31
BEA e-world	www.bea.com/events/eworld/2002/	404-240-5506	9
Corda Technologies	www.corda.com	801-805-9400	13
Java Developer's Journal	www.sys-con.com/java	201-802-3020	54
JavaOne	http://java.sun.com/javaone/	888-886-8769	45
Sitraka	www.sitraka.com	800-663-4723	60
Softwired	www.softwired-inc.com	41-1-445-2370	59
Sonic Software	www.sonicsoftware.com	781-999-7000	11
SYS-CON Media	www.sys-con.com	201-802-3020	45
WebLogic Developer's Journal	www.sys-con.com/weblogic	201-802-3029	49
Web Services Edge	www.sys-con.com	201-802-3069	49
Web Services Journal	www.sys-con.com/webservices	201-802-3023	56-57
Wily Technologies	www.wilytech.com	650-227-7600	23
Wireless Business & Technology	www.wbt2.com	201-802-3029	39, 55
Wireless Edge	www.sys-con.com	201-802-3069	53

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

NEXT MONTH

Building SOAP Web Services with WebLogic 6.1
by Paul Elisil

Ant and WLS
by Michael Gendelman

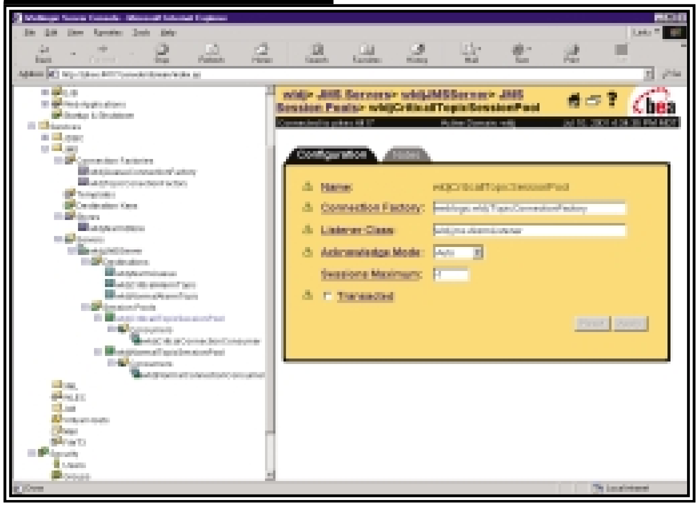
Clusterable EJBs (part 2)
by Tyler Jewell and Lawrence Kaye

J2EE Application Migration–From 5.1 to 6.1
by Dmitriy Yakubovich

Effective Customer/DRE Communication
by John Greene

BEA WebLogic DEVELOPER'S JOURNAL

FIGURE 4


wldj> JMS Servers> wldjJMSServer> JMS Session Pools> wldjCriticalTopicSessionPool

Configuration | Notes

Name: wldjCriticalTopicSessionPool

Connection Factory: weblogic.wldj.TopicConnectionFactory

Listener Class: wldj.jms.AlarmListener

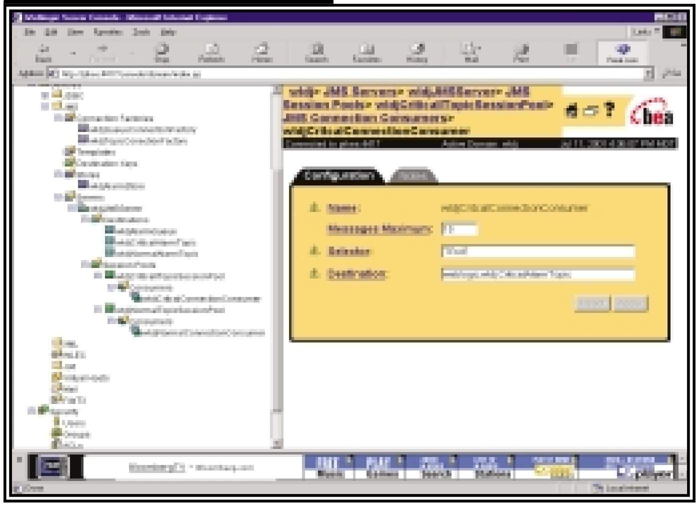
Acknowledge Mode: Auto

Sessions Maximum: [illegible]

Transacted

Console view of creating ServerSessionPool

FIGURE 5


wldj> JMS Servers> wldjJMSServer> JMS Session Pools> wldjCriticalTopicSessionPool> JMS Connection Consumers> wldjCriticalConnectionConsumer

Configuration | Notes

Name: wldjCriticalConnectionConsumer

Messages Maximum: [illegible]

Selector: [illegible]

Destination: weblogic.wldj.CriticalAlarmTopic

Console view of creating a ConnectionConsumer

that allows you to specify the characteristics of the ServerSessionPool (see Figure 4).

The wldjCriticalTopicSessionPool specifies the wldjTopicConnectionFactory as the ConnectionFactory to use, the fully qualified classname of the MessageListener, wldj.jms.AlarmListener, the Acknowledgment Mode as Auto, the maximum number of Sessions, and the transaction semantics of the Sessions. All that must be provided programmatically is the MessageListener, which has to be provided if you create the ServerSessionPool programmatically as well.

Next, create the ConnectionConsumer by navigating to the Consumers tab under the newly created ServerSessionPool (see Figure 5). The ConnectionConsumer is configured with the maximum number of messages that it should load into a Session, a message selector, and the Destination for which it is consuming messages.

When messages arrive at the wldjCriticalTopicConnection, the ConnectionConsumer gets a JMS Session from a ServerSession in the ServerSessionPool, loads the Session with a message, and starts the ServerSession. The ServerSession then calls the run() method on the Session, which activates the onMessage() method of the associated MessageListener and passes it a Message.

QueueSend Client

This application passes a TextMessage from a "store" queue, wldjAlarmQueue, to "forward" Topics, wldjCriticalAlarmTopic, and wldjNormalAlarmTopic. To get the TextMessage to the wldjAlarmQueue, a JMS client that uses a QueueSender was created. The client sets the JMS MessagePriority, sets a String header property named "level", and sets a String into the TextMessage (see Listing 6).

Transactions

Transaction semantics in the JMS Specification are associated with the Session object. A Session can be created as transacted or nontransacted. Transacted Sessions have their transactions committed or rolled back with the commit() and rollback methods. Nontransacted Sessions are created with an acknowledgment mode specified.

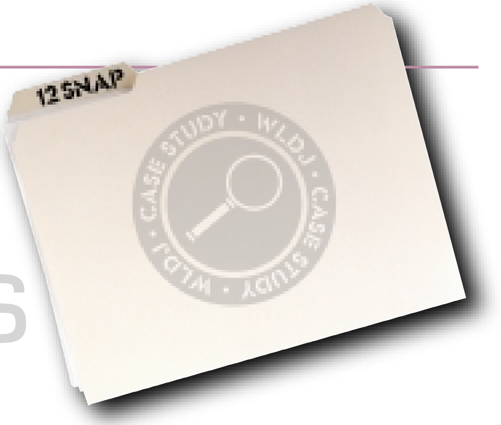
When using server-side asynchronous message consumers, MDBs or ServerSessionPools, using nontransacted Sessions with an acknowledge mode of AUTO_ACKNOWLEDGE allows the JMS server to manage the transactions for the message consumer by calling its onMessage() method in the context of a transaction, and then committing the transaction when the onMessage() method returns. If you use transacted Sessions, your MessageConsumer is responsible for committing or rolling back the transaction.

JTA User Transactions aren't supported with MDB. A JTA User Transaction started in the onMessage() method of a MessageConsumer called by a Session in a ServerSessionPool will be outside of the context of the transaction started when the onMessage() method is called.

Summary

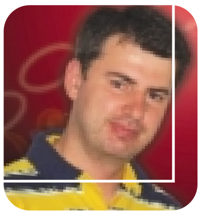
An asynchronous message-processing application can take advantage of the JMS and EJB APIs to perform message processing using just server-side code that runs in the framework of WebLogic Server. This releases client code from having to manage threads, concurrency, and transactions. The ease of creating this store and forward example is a testament to the robust implementation of the JMS and Message Driven Bean APIs by WebLogic Server.

ServerSessionPools and ConnectionConsumers are presented here as an alternative to using MDBs, but are more involved in their implementation. MDB is a simpler choice with the restriction that only one MDB can be deployed per Destination. Multiple ServerSessionPools and ConnectionConsumers can be created to service one Destination, and each ConnectionConsumer can be deployed with a message selector to filter messages based on Message header properties. Message selectors are expensive and not recommended on Queues, but may be useful to filter messages on a Topic that may have more than one interested subscriber or that may be consumed by more than one ConnectionConsumer. Message filtering on a Queue by an MDB should be performed programmatically in the MessageListener's onMessage() method, as was shown by the AlarmReaderBean.



Mobile Auctions with 12snap

USING COMMON SENSE AND A TOUCH OF J2EE



BY
PETER ZADROZNY
AND
JEAN-PAUL DE VOOGHT

AUTHOR BIO...

Peter Zadrozny is the founding editor of **WebLogic Developer's Journal.** When he is not working on the magazine he spends his time on his real job as BEA's chief technologist for Europe, the Middle East and Africa.

Jean-Paul de Vooght is head of technology at 12snap AG in Munich, Germany. After working at Cambridge Technology Partners, Inc. on B2C sites such as Swissair and Robeco Advies, he joined the German wireless company and now focuses more on B2B activities around mobile marketing.

CONTACT...

z@bea.com
jp@12snap.com

In October 1999 12snap's main goal was the development and commercial implementation of the world's first wireless shopping platform for existing mobile devices as well as WAP–enabled phones. The first application allowed mobile phone users across Germany to participate in auctions through a combination of voice, cell broadcast, and short messages, all based on the GSM technology so popular around the world (except in North America).

This auction application went into production in February 2000 in Germany, the United Kingdom, and Italy. Additionally, there were other applications such as fixed-price shopping. In June 2001 a wireless betting application was added for the UK market. At the time of this writing, 12snap has removed the auction application from production until new market opportunities arise. However, we find the application interesting and will discuss it here.

How It Works

The idea is that anybody can participate freely in any of the auctions available. Ideally the participants are people who have already registered with 12snap; non-registered users can still participate in an auction, but if they win they have to register.

The registration process is purposely very easy. The participants can either contact 12snap's call center to verbally provide all their information or they can go to the 12snap.com Web site and type in all of the required information. The important thing is that 12snap ends up with all the payment information to properly close the transaction.

Auctions are announced via Cell Broadcast Services (CBS). The CBS message is received by any users, who activate the feature in their mobile phones. The activation allows configuration and reception of a certain number of channels. Each channel is identified by a three-digit identification and can be labeled. 12snap broadcasts on Channel 123, which is closer to omnicast than multicast.

Every day the application sends several CBS messages to the mobile operator, who then distributes the messages to the cell broadcast–activated handsets. Each message announces the product to be auctioned on that day (see Figure 1), including a basic description of the product, the auction start price, and a phone number pointing to the voice response unit of 12snap (see Figure 2).

As other users participate in the same auction, the system uses the Short Message Service (SMS) to inform previous participants in the auction with "bid topped" messages (see Figure 3). The SMS message carries enough information for the

12snap auction teams

user to participate again and place a higher bid (see Figure 4).

Once the auction is over, the application notifies the winner with a new SMS message (see Figure 5). When the winner is not a registered user, the SMS message also invites the user to register and provide payment information. In this case, the auction application waits for several hours for the unregistered winners to provide their information before proceeding with payment and fulfillment activities. If an unregistered winner does not provide the payment information within that period, he or she is blacklisted and prevented from participating on future auctions.

At last count there were about 80 auctions a week running in Germany. These occurred between noon and 8:00 p.m. on weekdays and between noon and 10:00 p.m. on weekends. The average duration of an auction was about 15 minutes and one hour each day was dedicated to a specific type of product.

Although the auction engine was designed to handle concurrent auctions for the various regions of Germany, this feature was never used.

Administration

The auction application is administered via a browser interface. From the various administrative tasks we can get a look at how an auction is created. This requires the following information:

- ***Item, supplier number, and name*** from the dropdown menu refers to the actual product to be auctioned.
- ***Product Category*** forces an auction to be scheduled at specific times of the day based on categories, such as "events," where tickets to a music concert could be auctioned, or "electronics" for items such as mobile phones.
- ***Number of items*** to be auctioned affects the actual number of winners. If two items are to be auctioned, then two customers will win and will end up paying two different prices according to the strategy followed by 12snap's auction model.
- ***Starting price***
- ***Warning price*** is the price above which the customer receives a warning indicating that the amount entered is too large.
- ***Minimum bid***
- ***Speaker name*** identifies the set of sound files or voice prompts used by the IVR to speak out the welcome messages, instructions, and product descriptions.
- ***Item type*** affects the timing parameters of the auction, particularly those affecting the closing of the auction and processing of payment data for winning customers. For example, an item could be perishable.
- ***Payment methods***

After this, other screens are used to add more information related to a new auction:

- ***The date*** and the starting time
- ***The time*** the auction will last and extension time
- ***Assign a time*** scheduler to the auction.
- ***Assign a slot*** to the auction begin time.

The Whole Picture

Putting it all together, we have all of the components shown in Figure 6, where the participants supporting the process described earlier are:

- The customer using a handset or a browser
- The public land mobile network (PLMN) of the mobile operator
- The mobile operator
- Public switched telephone network (PSTN) for voice hand-over to the TSP
- Packet data network for connection to the SMS center
- The telephone service provider for SMS and IVR capability
- 12snap systems
- The payment partner
- Logistics for delivering goods to the winners
- Address-checking facility for maintaining high quality on data from registrations

The Solution

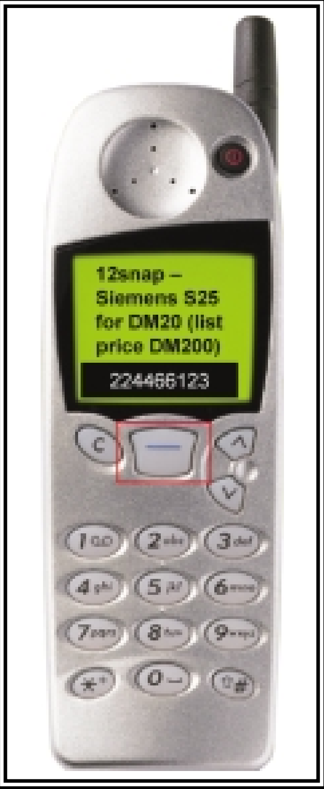
The 12snap team had a major challenge in trying to design, develop, test, and implement the application in about four months, which is more or less equivalent to the infamous "yesterday" deadline that we are all too familiar with. Additionally, the application needed a cost-effective integration solution capable of supporting the connection needs to systems, such as voice response units, SMS gateways, and third-party payment services.

Java, along with the Enterprise Java Beans specification, was quickly identified as the basic technology that would allow a quick development and implement a clean service-based solution. WebLogic was then selected as the application server platform mainly because it worked correctly on the first try.

The design stage was divided into the following parts:

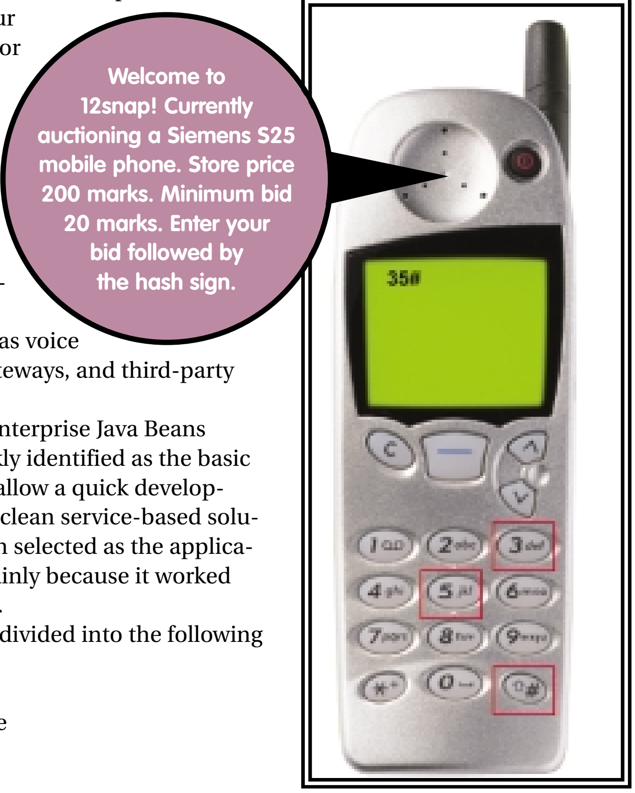
- Auction logic
- Software architecture
- Data modeling
- Front-end design
- Call-flow and IVR integration

FIGURE 1



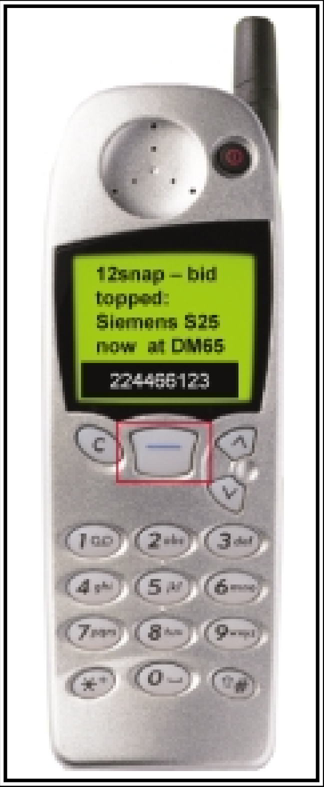
Announcing the product

FIGURE 2


Welcome to 12snap! Currently auctioning a Siemens S25 mobile phone. Store price 200 marks. Minimum bid 20 marks. Enter your bid followed by the hash sign.
35#

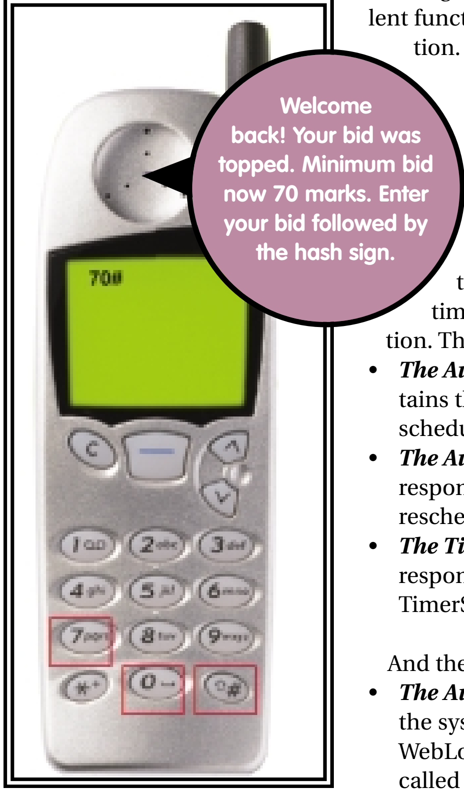
A bid is entered

FIGURE 3



SMS with bid-topped message

FIGURE 4


Welcome back! Your bid was topped. Minimum bid now 70 marks. Enter your bid followed by the hash sign.
70#

Request for another bid

- SMS and CBS integration
- Development setup
- Production setup

Timed Auction Logic

The initial aspects of the design addressed the auction logic itself. There are a variety of auction models that can be classified based on three phases: setup, bidding, and resolution. For example, in Dutch auctions the setup phase initiates a very high opening bid. During that phase the participants bid to lower the initial price. This will run until demand rises to match supply or the goods are exhausted.

12snap adopted a very simple auction model. In the setup phase the number of goods is defined. If there is more than one item, the winners will pay different prices. During the bidding phase the participants bid to increase the price. This process stops when no participant places a bid during a period called the *extension time*, which is defined in the setup phase. When there is only one item, the resolution phase assigns the item to the highest bidder. When there is more then one item (N > 1), they are assigned to the *N* topmost bids.

In addition to the basic strategy, a number of states were defined for the auction, which indicate what methods need to be triggered. The actual transitions are timer-based and use WebLogic timer classes. Although these are WebLogic proprietary classes, there is no equivalent functionality present in the J2EE specification.

The problem is that these timer classes are not cluster-aware, that is, a WebLogic timer is not aware of other timers running outside of its JVM. Since the application would run in a WebLogic cluster, this was going to be a show-stopper. The 12snap team had to perform some magic to allow the timer to work in a WL cluster configuration. This magic was based on three EJBs:

- ***The AuctionTimer:*** An entity bean that contains the scheduling for an auction. The scheduling is a vector of events.
- ***The AuctionTimerMgr:*** A session bean responsible for creating, cancelling and rescheduling the AuctionTimer
- ***The TimerSchedulerMgr:*** A session bean responsible for accessing the WebLogic TimerService

And there were also some normal Java classes:

- ***The AuctionTimerScheduler:*** The kernel of the system. It implements some specific WebLogic Timer interfaces and is the one called by WebLogic for scheduling and triggering events. Each instance is associated with an AuctionTimer instance. For scheduling, it calls the AuctionTimer to get the next activation time (but not for the first one, to avoid transaction locking). For triggering, it gets the event from the AuctionTimer, and removes it from the list of next events. Then it gets the action for this event, which is the name of the method to call on the AuctionMgr.
- ***The TimerSchedulerHelper:*** Provides utility methods for registering and cancelling AuctionTimer. It also provides methods for sending multicast messages for scheduling and cancelling timers.
- ***The TimeServiceHelper:*** Provides some utility methods to access the WebLogic Time Service.
- ***The TimerSchedulerMulticastThread:*** A thread whose role is to listen for multicast messages and execute the associated action.
- ***The SchedulingValue class:*** Describes an event.

There are other classes used, but they are not worth discussing. You can download this package from www.wldj.com.

FUNCTIONAL OVERVIEW

In a cluster, all the members will have a local auction timer. Every event for every auction will be scheduled on all the machines. When a timer wants to execute an event, it first checks to see if the action has already been executed. If so, it does nothing; otherwise, it executes the action.

EXECUTION OF AN EVENT

When an event is triggered, the database row concerning the particular auction is locked by using a "select for update." This is done on the t_auction_event table.

It then gets the next event to be executed, first

TABLE 1

OFFSET	DESCRIPTION
0	Magic number, a rather poor way to detect false requests
1	Action to be executed: 0 for cancel, 1 for schedule
3 to N	The message string as follows: <auction ID>[;<first time>] where: <auction ID> is the reference of the associated auction, and <first time> is the first activation time in milliseconds since the Epoch. It is not required for cancellations.

Byte-buffer format

checking that the event has not already been executed. To do so, it compares the date of this event with the next activation date stored in the AuctionTimerScheduler for when this event was scheduled (previous call to the schedule method). If these two dates match, then it can be executed. If not, another machine must already have executed this action, so the method just returns.

By locking the row in the database, it prevents two instances of WebLogic from executing a sequence at exactly the same time.

The transactions are handled by using a "user demarcated" transaction using JTS.

SCHEDULING ON ALL THE MACHINES

This is the trickiest part of the system. As there was no way to know all of the instances of WebLogic servers that make up a cluster, 12snap was forced to implement a solution to communicate with all the members of the cluster.

This was done by using multicast and a dedicated socket. There is a thread that listens on this socket for messages and executes the commands. On the other side, when you want to schedule or cancel a timer you simply send a message on this multicast socket and every member of the cluster will receive it.

The TimerSchedulerMulticastThread listens for messages. The TimerSchedulerHelper contains utility methods to send these messages. The TimerSchedulerMgr bean, which is responsible for scheduling or cancelling timers, sends only the multicast messages.

MESSAGE FORMAT

The message is a byte buffer with the format shown in Table 1.

PROPERTIES INVOLVED

The system used the following properties:

- ***timerScheduler.multicastIp:*** Multicast IP for the timer system
- ***timerScheduler.multicastPort:*** Multicast port for the timer system
- ***timerScheduler.sessionTimeout:*** Transaction timeout for the execution of an event

RESTARTING THE TIMERS

This process is different from the basic scheduling because when it is executed the multicast socket may not be open. Basically, every machine registers the StartTimers startup class. But now, when you update and register the timer, you lock the row in the database with a "select for update" and a user-demarcated transaction. Then you update, that is, you remove the events that could not be executed, and you register the timer locally. To be able to schedule a timer locally or cluster-wide, some methods have been added to AuctionTimerMgr.

State Transitions

Equipped with this sophisticated timer, the application can control state transitions for various auctions running concurrently.

In terms of modeling the auction engine, failures should receive special attention; they did not consider, for example, an "error" state for failures related to payment.

Software Architecture

Implementing the core auction logic with WebLogic and EJB 1.0 led 12snap to make a few key decisions. It chose a now very common pattern that allowed it to deal with services for later reuse. Services would be implemented using stateless session beans. For persistence, they chose entity beans using container-managed persistence – at least for most tables. Servlets would be used as access method for its services. This is illustrated in Figure 7.

Presentation Layer

When the project was started, JSP had barely made it to 1.0. Considering the difficulties in keeping up with the specification and the known pitfalls in maintaining production JSPs, 12snap chose a more conservative template approach to the presentation layer and decided to use FreeMarker, which comes in the form of a library with template classes.

Short Message Services

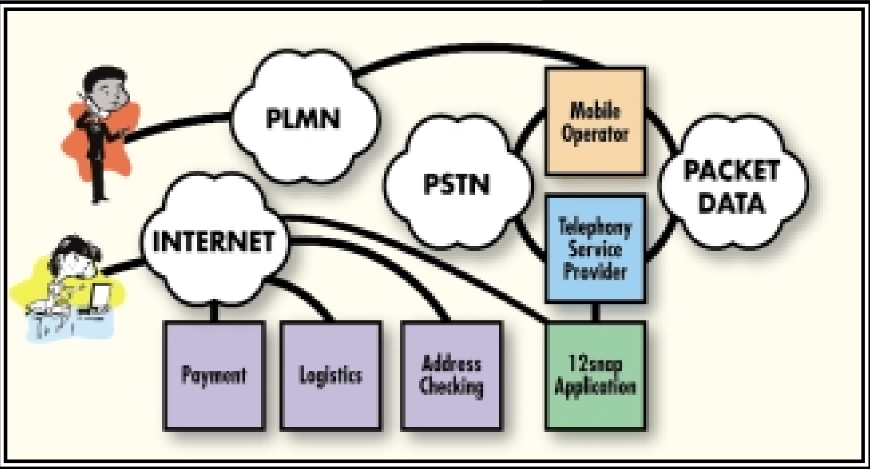
The Short Message Service (SMS), as defined within the GSM digital mobile phone standard, has several unique features. A single short message can be up to 160 characters of text in length and consists of words, numbers, or an alphanumeric combination. Non-text– based short messages (for example, those in binary format) are also supported.

The Short Message Service is a store and forward service. In other words, short messages are

FIGURE 5

Notification that bid was accepted

FIGURE 6



All the participants supporting the process

“The 12snap team had a major challenge in trying to design, develop, test, and implement the application in about four months, which is more or less equivalent to the infamous ‘yesterday’ deadline that we are all too familiar with”

not sent directly from sender to recipient but through an SMS Center (SMSC) instead. Each mobile telephone network that supports SMS has one or more messaging centers to handle and manage the short messages.

The Short Message Service features confirmation of message delivery. Unlike paging, users do not simply send a short message and trust that it gets delivered. Instead, the sender can receive a return message notifying them whether the short message has been delivered or not.

Short messages can be sent and received simultaneously with GSM voice, data, and fax calls because whereas the other calls take over a dedicated radio channel for the duration of that call, short messages travel over and above the radio channel using the signaling path. As such, users of SMS rarely, if ever, get a busy or engaged signal.

Sending and receiving SMS messages from an application requires a gateway solution that implements the native protocol supported by the SMS Center. On the application side, SMS gateways tend to offer standard access protocols such as SMTP, RMI, HTTP, or file access to interact with the mobile message infrastructure. Other formats are also available.

Initially, 12snap decided to outsource the SMS gateway technology to one of the mobile operators in Germany and agreed on an SMTP interface. At the WebLogic level, they used JavaMail to prepare and send e-mail messages to the SMS gateway that corresponded to the SMS notifications the auction engine needed to send to the auction participants.

Cell Broadcast Services

12snap initially implemented a gateway to the Cell Broadcast Center (CBC) of one of the main operators in Germany by using its Cell Broadcast Entity (CBE). It was a solution based on a Web server hosting forms processed in the back-end by an application server that would, in turn, pass requests to the CBC. That *n*-tier solution was hosted at the operator premises. Eventually, the solution was migrated to 12snap's own WebLogic servers by implementing the ASN.1 protocol required to directly access the CBC (in agreement with the operator as there are some very sensitive load and regionality issues to consider).

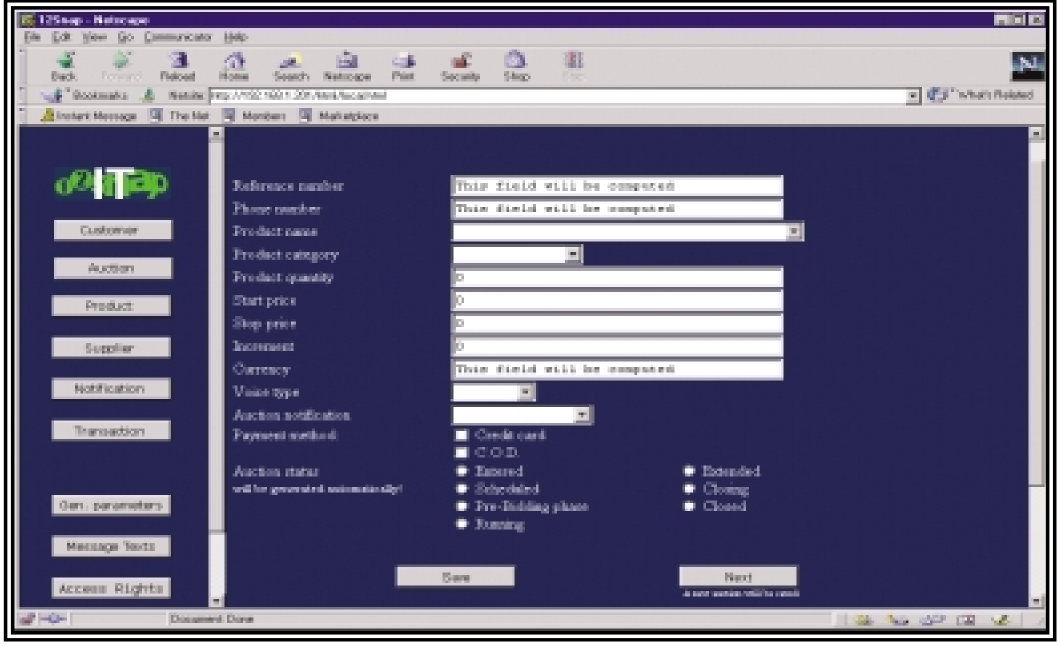
The implementation of the ASN.1 protocol was made using JavaCC and a reference CBE tool from the CBC manufacturer.

Interactive Voice Response

Interactive Voice Response servers (IVRs) are curious animals when you have an application server view of the world. After a 20-minute discussion with IVR specialists, 12snap discovered that the application server and the IVR can easily compete for the business logic decision in processes. From the very beginning they tried to confine the business logic decisions to the WebLogic cluster and keep the IVR as “lean” as possible. They were in fact lining up for the upcoming VXML, VoxML, and VoiceXML standards without really knowing it.

An IVR has connections on one side to the telephony network (e.g. ISDN Q.931) and on the other side to the data network. Its DSP boards and software provide the necessary glue between

FIGURE 7



Reference number	This field will be computed
Phone number	This field will be computed
Product name	
Product category	
Product quantity	0
Start price	0
Stop price	0
Increment	0
Currency	This field will be computed
Value type	
Auction notification	
Payment method	Credit card / C.O.D.
Auction status (will be generated automatically)	Entered / Scheduled / Pre-Bidding phase / Running / Extended / Closing / Closed

The access method for its service

javaone

http://java.sun.com/java

both. When a call arrives at the IVR a script is immediately executed and two key pieces of information are given to that script: ANI and DNIS. ANI stands for *Automatic Number Identification* and represents the number of the person calling the IVR. *The Dialed Number Identification Service (DNIS)* carries the number called by the person. In essence, these numbers are equivalent to routing information that can be used by any script to make decisions.

The scripts running on the IVR are commonly referred to as call-flows and link voice prompts together. It was decided, for performance reasons, to keep the actual sound files on the IVR itself. Call-flow decision points ask for a link to the middle layer for consistency across multiple channels. For example, when a caller dials into a specific auction, the IVR will need to contact the WebLogic server to find out if that auction is available or not.

12snap auctions began by hosting its equipment at an ISP that offered managed IVR solutions. In order to avoid technology dependencies it integrated the IVR with WebLogic through HTTP by implementing a basic thread-safe browser library on the IVR. For each line the IVR picks up, a browser thread was activated and used by the call-flow scripts to place HTTP requests against WebLogic. Servlets such as GetBid were implemented on the WebLogic cluster to pass information back to the call-flows.

TABLE 2

CATEGORY	SELECTION	NOTE
Workstations	Intel PC running SuSE Linux	Cheaper than the laptop option running X server
App server	2 Sun Ultra10	WebLogic 4.51 on Solaris
Database server	1 Sun Ultra30	Oracle 8.0.6 on Solaris
App server proxy	1 Sun Ultra5	iPlanet Enterprise Server 4.01
Web server	1 Sun Ultra5	iPlanet Enterprise Server 4.01
SMS gateway	N/A	outsourced to TSP, accessible via e-mail
IVR	N/A	outsourced to TSP

Development hardware

The External Server

Initially, the application requirements defined the support of various payment methods (credit card, direct debit, and cash-on-delivery) as well as address verification and fulfilment notifications to be sent to its logistics partner. Integrating with the business process of such service providers depends on the technology available. The payment- and address-checking services were initially available from the same provider in the form of shared objects for SPARC. The fulfilment partner used a much simpler interface: e-mail. It decided to implement, for those third-party libraries, a special component dubbed "external server," which would run in a separate JVM and host the JNI-wrapped libraries.

This component supports modules, in this case Java objects that access native libraries. Accessing these libraries from the WebLogic JVM was considered too risky due to JNI-related issues, such as instability.

The external server registers a Factory with WebLogic's JNDI tree so that they can access the modules from the application server. A "check" thread acts like a watchdog and verifies that the connection to the Weblogic server is still alive. If not, it re-registers itself into the JNDI tree. A console allows operators to administer the server via telnet.

Development Environment

In order to identify the most suitable platform for its system, 12snap compared NT and Solaris systems. Based on the anticipated transaction estimations (accumulated values across IVR, Web, and WAP), it opted for Sun Solaris as the development and production platform. Since this is expensive equipment it had to be a little creative to minimize costs and agreed on the configuration shown in Table 2.

Linux Workstations Setup Procedure

Once configured with basic software and networking, they downloaded the customization script for the workstation from the repository server using FTP and ran the install shell script located under the install directory. This script performs the following actions:

1. Creates special directories
2. Backs up configuration files (profile, hosts, fstab)
3. Installs skeletons
4. Installs Java Development Environment (JDE) for Emacs
5. Mounts file systems
6. Installs JDK

The tools related to the installation script are:

- Libraries (e.g. FreeMarker)
- Application server (e.g. weblogicaux.jar and Service Packs)
- JDK

Additional tools are shown in Table 3.

When it started the service, it identified a mobile operator capable of supporting it with hosting, IVR, SMS gateway, and call center capability. It was left to bring in its production hardware as shown in Table 4.

Figure 8 shows how the components are laid out on the network segments.

JDK 1.3 and WebLogic 5.1 Migration

A few months ago 12snap decided to revisit the WebLogic cluster configuration and set up some tests under new conditions – namely upgrading the JVM from 1.2.2 to 1.3, upgrading to WebLogic 5.1, and checking out the performance of two instances of WebLogic per server instead of one. These tests were conducted by benchmarking the performance of four of the most used servlets in its application:

- ***GetAuctions:*** Used on the Web site to display a list of auctions
- ***GetCategories:*** Used by the IVR to present a menu of auctions running
- ***GetCurrentPrice:*** Used by the IVR to tell the latest price for one auction
- ***SetBid:*** Used by the IVR to set a customer's bid

These benchmarks showed an increase in performance coming not only from the migration to the latest JVM but also from having two instances of the WebLogic server on every Sun machine in production. This effectively created a cluster of four WebLogic instances on two Sun machines. The benchmark was done by simulating 120 simultaneous users generated by The Grinder. This is an open source Java-based load generation tool. It can be downloaded for free from http://grinder.sourceforge.net.

Day-to-Day

The following is a high-level list of some processes that take place on a day-to-day basis to run the wireless auctions:

1. Product scouting (at the beginning of the month)
2. Order fax sent to supplier
3. Production and uploading
   - Get product pictures for Web site.
   - Get short and long description of products.
   - Produce the IVR voice prompts of the long and short description.
   - Upload the files.
4. Finalize production
5. Auction planning
6. Auction scheduling
7. Last-minute changes (e.g. new products)

Common Scheduling Problems

The most common error in scheduling auctions was the "wrong category for this hour." Someone already scheduled an auction for a specific hour and the new auction is in a different category from the first one of that hour.

The second most common error was scheduling auctions which should start as soon as possible. The auctions were coupled to the Cell Broadcast channel and specifically to the property CBCDelay and the value "Preceding Time for Hourly Reminder." A large value here allows the broadcast to be sent early (which is better from the CB Center perspective), but this also means that once this time had passed it was too late to schedule a new auction for that hour.

After scheduling an auction, all CB messages were sent to the CB Center immediately. In the case of an error a delete-message was sent. At this moment, CB is available only in Germany. It implemented an SMS-based subscription and notification module for use in Italy and UK.

Operational Information

Before closing the auction application, 12snap had about 80,000 subscribers in Germany, 50,000 in Italy, and approximately 10,000 in the UK. These numbers represent the number of customers for which 12snap has full information to complete payment transactions.

As we mentioned earlier, the system did not require the users to be registered to participate in an auction, so the potential users were actually higher than the number of registered users.

TABLE 3

CATEGORY	SELECTION	NOTE
Compiler	Sun JDK 1.2	First used blackdown on Linux and then Sun on used both Linux and Sun machines
IDE	emacs with JDE, NetBeans/Forte, JBuilder, vim	
Build Tool	Jakarta Ant	Powerful alternative to using XML and extensible with Task objects
Debugger	N/A	
Unit Testing	JUnit 3.2	
Load Testing	The Grinder 1.x	Before The Grinder was made public, we used Microsoft Stress Testing Tool
Documentation	javadoc tool	
Bug Tracking	BugZilla	Requires MySQL and Perl knowledge to maintain
Version Control	CVS	Only problem is lack of locking which can bring complication during large builds

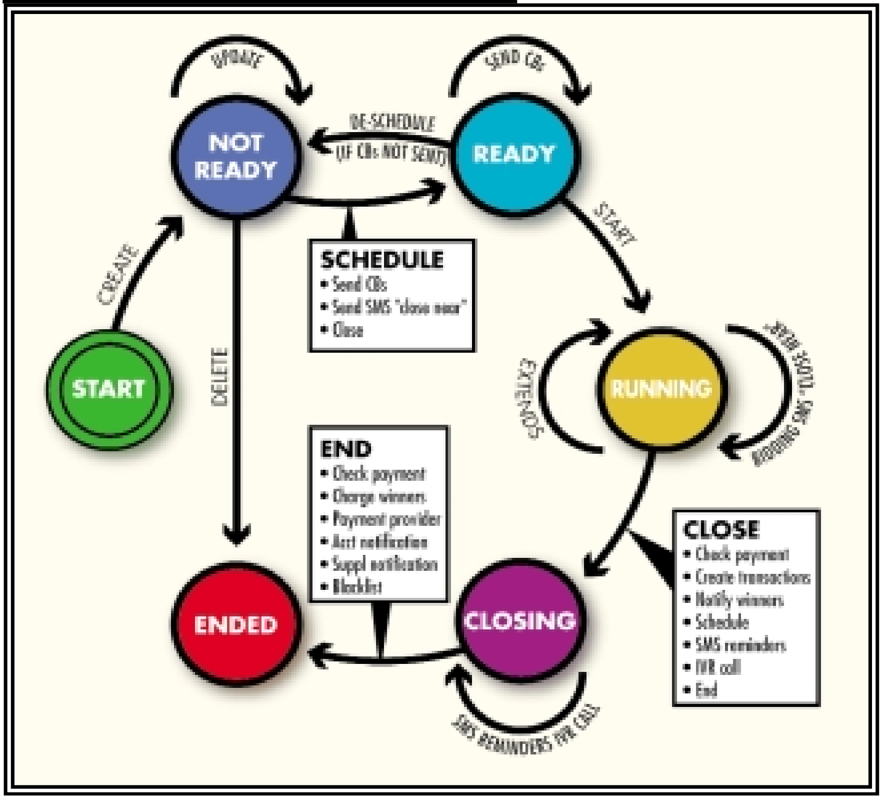
Tools

TABLE 4

CATEGORY	SELECTION	NOTE
App server	2 Sun E250 2 CPUs 1 Gb RAM	WebLogic 4.51 on Solaris One of the servers was installed with Oracle as standby and connected to the array
Database server	Sun E450 2 CPUs 1 Gb RAM	Oracle 8.0.6 Enterprise Edition
App server proxy	Sun Ultra5	iPlanet Enterprise Server 4.01 with WL NSAPI Plug-in
RAID array	Sun A1000	
Web server	Sun Ultra5	iPlanet Enterprise Server 4.01 with WL NSAPI Plug-in
SMS gateway	N/A	outsourced to operator
IVR	N/A	outsourced to operator
Backup	Internal DLT drives	Using Solaris tools

Production Hardware

FIGURE 8



How components are laid out on network segments

All servers for all countries are hosted at an ISP in the UK. Each country has its own combination of WebLogic proxy and cluster servers. Database servers are shared for two of the three countries. The ISP provides connectivity to the 16 ISDN E-1 lines to service a total of 480 lines.

There were about 80 auctions per week starting at noon and, interestingly enough, the peak times were at noon and at around 6 p.m.

Lessons Learned

In the day-to-day operations, one of the first things 12snap learned was that whenever there was a failure, it had to look in the WebLogic log file. A careful analysis of this file solved 90% of the problems it encountered. This is a hint that is placed conspicuously in all of the "Hint" documents.

One of the first comments from its software engineers was, "We need code-beautifying, because code is partly not readable." This was an annoyance that was never resolved. It would be helpful to have some sort of beautifying tool for Java and specifically for J2EE.

Within its code, exception handling suffered from poor design. 12snap limited its analysis to splitting exceptions into Recoverable and Unrecoverable from the user standpoint, e.g., wrong input: recoverable, database down: unrecoverable. This falls short when it comes to choosing which type is needed by the developer.

12snap now realizes it should have adopted nested exceptions based on a Runtime/Checked exceptions categorization. The output of exceptions should only be done in servlets or timer-triggered methods.

The proper naming of variables was another area that caused trouble and 12snap concluded that it should always be carefully considered during code reviews. For example, at some point 12snap had the following "pathological" variable names:

- available meaning pre-bidding
- maxPrice meaning stopPrice
- ready meaning scheduled
- closed meaning ended

For the ERP system 12snap chose Navision Financials running on NT. In order to integrate the WebLogic cluster of a specific country and the Navision server, it installed and configured a transfer server, (which was a dedicated Sun server), with Samba. This server was installed and configured to belong to the same domain as the ERP server. As soon as the session beans completed successful payment and fulfilment transactions, a file was written into that special directory using the WebLogic File Services. This was too simple.

Writing the file could fail for several good reasons (e.g., disk full), and information would not pass to the ERP system although the application would flag it as complete. In retrospect, this is something that 12snap reckons it should have implemented using an asynchronous mechanism, such as the Java Message Service. This way there can be some level of guarantee that the transactions will arrive correctly in the ERP system.

From Navision's side, a dedicated client runs the Navision data ports, which are timer-controlled scripts that would fetch or deposit files to/from Navision.

Conclusion

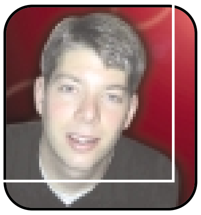
Although the application does not use all of the J2EE APIs, mainly because of its availability at the time it was developed, 12snap was able to develop and put it into production in record time. It ran for more than 18 months on version 4.5.1 with a relatively large number of users taking full advantage of WebLogic clustering. One of the neat tricks was getting the timers to work in a clustered environment–and we have shared that experience here.

As you can see, nothing magical happened in order to have a stable application running. Just a lot of common sense and a few mistakes that we hope everyone can learn from.

Entity Enterprise JavaBeans (EJBs) are a convenient means to map persistent data to Java components. Container-Managed persistence (CMP) provides rapid development since the EJB container automatically handles loading and storing the persistent data. However, along with their many advantages, Entity EJBs can lead to very slow performance when used incorrectly. This column details a few common pitfalls which trip up EJB programmers and hinder the performance of their Entity beans.

Avoiding Performance Pitfalls with Entity EJBs

BY **ROB WOOLLEN**



AUTHOR BIO...

Rob Woollen is a senior software engineer on the WebLogic Server development team at BEA Systems. He holds a bachelor's degree in computer science from Princeton University

CONTACT...

rwoollen@bea.com

Primary Key Classes

Like a database row, entity beans have an associated primary key. This primary key may be a single entity bean field. In this case, the entity bean can use the field’s class as the primary key.

It’s also possible to provide a custom primary key class. This is necessary for a compound primary key, one that maps to multiple entity bean fields.

With a custom primary key class, the developer must implement the hashCode and equals methods. Since the EJB container often uses the primary key class in its internal data structures, this class must implement hashCode and equals correctly and efficiently (see Listing 1).

Implementing hashCode

The hashCode method must return the same value for two objects which are equal, and it should attempt to distribute the hashCode values relatively evenly. Our first implementation, shown below, is efficient and correct, but it does not distribute the hashCode values at all. This hashCode implementation transforms any hash table into a list and forces linear searches. Clearly, this defeats the whole purpose of having an indexed data structure.

```
private int hash = -1;
public int hashCode() {
  if (hash == -1) {
    hash = str.hashCode() ^ i ^ b;
  }
  return hash;
}
```

The hashCode implementation above computes the Exclusive OR (XOR) of the string's hashCode and the primitive fields. XOR should be preferred to other logical operators such as AND or OR since it gives a better distribution of hash values. This implementation also caches the hashCode value in a member variable to avoid recomputing this value.

Implementing Equals

The equals method compares the current object with the passed parameter and returns true if the Objects have the same value. The default java.lang.Object.equals compares the reference (pointer) values and returns true if they are equal. For most primary key classes, this needs to be overridden to compare the values within the primary key class (see Listing 2).

The first line of the optimized equals implementation compares the passed reference against this. While this appears strange at first, it is a common case when the EJB container checks whether a primary key already exists in its data structures.

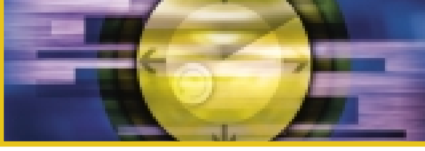
Next, we have replaced the getClass().equals with a much more efficient instance of check. The instance of operator returns true if the passed parameter’s class is MyPk or one of its subclasses. Making the MyPk class final allows the create method to safely use the instance of operator since there cannot be a subclass.

Finally, the hashCode and member variables are compared. And expressions are short-circuited in Java which means that if the first expression is false, the second is not evaluated. Our equals method takes advantage of this by ordering the and statement with the cheapest comparisons first. The hashCodes are compared first since our implementation caches this value, and it should be very rare for both objects to have the same hashCode but not be equal. Next, the primitive fields are compared, and finally the more expensive java.lang.String.equals is called.

Loading and Storing Entity Beans

The WebLogic Server's EJB Container offers two distinct entity bean types: Read-Only entity beans and Read-Write entity beans. Read-Only entity beans support advanced clustered caching, and will be discussed them in another column.

Entity beans are transactional objects, and it is important to understand the relationship between transactions and persistence. When an entity bean instance is first used in the transac-



tion, its state is refreshed from the database. Any modification to the entity bean state is flushed to the database immediately before the transaction commits (in the beforeCompletion callback, for those of you familiar with the javax.transaction.Synchronization interface).

Let's take a look at an example Employee entity bean that has the Required transaction attribute. You can assume that we created the bean in a separate transaction and currently have a reference to an Employee instance.

```
Employee e = ...
String name = e.getName();
e.setSalary(e.getSalary() * 2);
e.setLevel(e.getLevel() + 1);
```

The key observation from the code snippet above is that the caller has not started a transaction. Since the bean is deployed with the Required transaction attribute, each method runs as an individual transaction and causes loads and stores to the database. In this example, every getXXX method forces a database read while every setXXX method produces a database read and a database update. This simple example has five separate transactions with five database reads (SELECTs) and two updates.

```
Employee e = ...
tx.begin();
String name = e.getName();
e.setSalary(e.getSalary() * 2);
e.setLevel(e.getLevel() + 1);
tx.commit();
```

The code example above wraps the calls to the Employee entity bean within a single transaction. In this sample code we use a UserTransaction reference to manually begin and commit the transaction. This could also be done with a session bean with a container-managed transaction that then calls the entity beans within the container transaction.

When all of the business methods run within a single transaction, there are only two database accesses. The first getName call produces a SELECT statement to load from the database. The subsequent getXXX and setXXX methods do not cause any database access because they are within the same transaction, and the data is already loaded. When the transaction commits, an UPDATE statement is issued to write the new salary and new level to the database.

Using the Mandatory Transaction Attribute

Many EJB programmers deploy their beans with the Required transaction attribute. Required works in many cases because it inherits the caller's transaction if one exists and otherwise starts its own transaction. As we've seen earlier, developers must understand how transaction attributes can greatly affect performance. A new programmer on your team might be reusing the Employee entity bean and mistakenly call every getXXX method in an individual transaction. One way to prevent this is to deploy your entity bean with the Mandatory transaction attribute. Unlike Required, a Mandatory bean will not start its own transaction. If it is called with a transaction, the Mandatory bean participates in that transaction. If a Mandatory bean is called without a transaction, the EJB container immediately throws an Exception to the caller.

Deploying entity beans with the Mandatory transaction attribute is an easy way to indicate that the operations should be grouped together in a calling transaction.

The CMP Advantage -- Finders Loading Beans

Like business methods, entity bean finder performance depends on setting and using transactions appropriately. Let's consider another example with our Employee Bean.

```
Collection employees =
  home.findEmployeesWithSalariesGreaterThan(30000);
Iterator it = employees.iterator();
while (it.hasNext()) {
  Employee e = (Employee) it.next();
  double salary = e.getSalary();
}
```

This simple example executes a finder (database query) to return employees whose salary is greater than the passed parameter, in this case 30,000. It then iterates through this collection and retrieves the salary from each. In the case where N employees are returned from

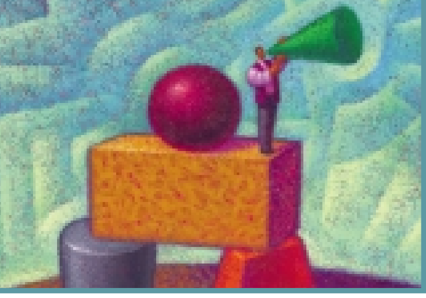
—continued on page 21

Listing 1:
An inefficient, but correct, primary key class

```
public class MyPk
  implements java.io.Serializable
{
  public String str;
  public int i;
  public byte b;
  public MyPk() {}
  public int hashCode() { return -1; }
  public boolean equals(Object o) {
    if ((o != null) && (MyPk.class.equals(o.getClass()))) {
      MyPk other = (MyPk) o;
      return other.str.equals(str) && other.i == i && other.b == b;
    } else {
      return false;
    }
  }
}
```

Listing 2:
An efficient equals implementation

```
public final class MyPk ...
  public boolean equals(Object o) {
    if (o == this) return true;
    if (o instanceof MyPk) {
      MyPk other = (MyPk) o;
      return other.hashCode() == hashCode() &&
        other.i == i && other.b == b &&
        other.str.equals(str);
    } else {
      return false;
    }
  }
```



BEA Systems, Inc., the makers of the WebLogic Server for which this magazine is named, empowers its customers with comprehensive online self-help services.

Customer Self-Help for the 21st Century

THE BOTTOM LINE – IMPROVED SATISFACTION

BY **KALA JARUGUMILLI**

AUTHOR BIO...

Kala Jarugumilli is currently working as a manager in the Customer Support Organization of BEA Systems, Inc. She is responsible for Customer Support's Knowledge Management Strategies at BEA Systems, Inc. Kala has more than four years of experience working in Customer Support organizations.

CONTACT...

kala.jarugumilli@bea.com

The introduction of BEA's WebLogic Server product into the marketplace produced a dramatic growth in the customer base that resulted in increased demand for customer support. Over the last few years, the explosive growth of WebLogic Server has helped BEA achieve market leadership in the application server space. Sales people are happy, executives are thrilled, and developers beam with pride. But what about technical support?

How can any support group possibly support 10,000+ customers with the same focus, technical expertise, and personal touch as it did when there were only 100? Clearly, scaling the number of engineers by two orders of magnitude isn't the answer. Therefore, in order to scale the organization and better serve its customers, BEA looked for an innovative way to extend support to this customer base. BEA turned to *self-help on the Web* as a way to allow customers to help themselves address technical problems. **AskBEA**, the company's powerful natural language processing engine, provides an efficient and fast means for Web-savvy customers to find the answers to their issues.

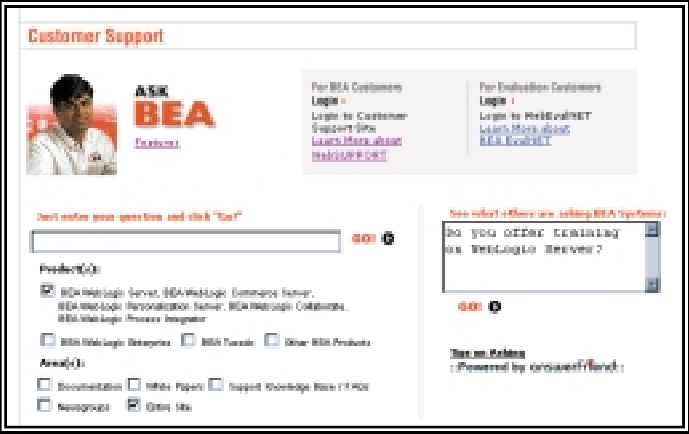
Customers desire information in an efficient, fast, and reliable mode. Early on, as the knowledge resource base for BEA products was being built, access to this information was spread across various site locations. This required customers to search the various sources separately in order to find all the information they were searching; not an easy and convenient task for customers. In an effort to provide a centralized, efficient, fast, and one-stop solution that addressed customers' needs, BEA implemented the AskBEA service.

AskBEA, available at www.bea.com/support is an online self-help service with a natural language processing engine that focuses on getting accurate answers to customer queries through a single platform that searches a number of resources containing the wealth of BEA product knowledge. Using AskBEA, customers can often find answers to their questions without filing a support case. We're not just talking about simple questions either. Imagine asking about an EJB-deadlock situation and getting an answer directly from the developer who implemented the EJB container. Of course, BEA's technical support team is stronger than ever, so if your AskBEA search isn't fruitful and you file a support case, one of BEA's highly skilled DREs (Developer Relations Engineers) will be able to assist you in a timely fashion (see Figure 1).

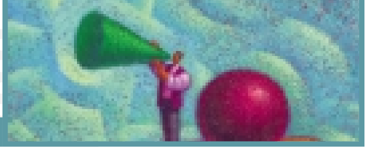
Implementation and Challenges

AskBEA started in July 2000 using the AskJeeves conventional search engine functionality. It was soon obvious that a less maintenance-intensive

FIGURE 1



AskBEA screen



technology with an advanced natural language processing engine was needed to better manage the process of updating rapidly changing content and result in improved answer performance. BEA embarked on improving AskBEA by using answerFriend's advanced natural language processing software.

PHASE I

The phase I implementation of AskBEA based on AskJeeves technology proved beneficial to customers as well as to BEA Systems. It allowed users to find answers to their questions online and covered product documentation, a support knowledge base, newsgroups, and white papers on the developer center and bea.com sites through a single interface. It also resulted in a decline in the number of queries coming into BEA customer support from evaluation customers.

However, the phase I implementation had its share of challenges; primarily due to inefficiencies in providing fast turnaround of dynamic content. It required labor-intensive steps to create predefined questions that were stored in a database and linked to sections of product documentation that contained possible answers. This resulted in enormous amounts of manual work and hindered the publication of new content every day, including newsgroup postings and solutions in the support knowledge base. Incorporating a new product release also required a laborious process of framing predefined questions and answers. Therefore, the phase I solution did not scale. Additionally, the integration for newsgroup articles was not an easy and seamless process.

Perhaps more importantly, the engine worked more like a keyword search engine than a true natural language processing engine that understands semantics and variations of word meanings. This resulted in answers that were less precise than desired. Because the implementation required excessive person-hours, the AskBEA phase I rollout was limited to BEA's WebLogic Server and WebLogic Commerce and Personalization Server products only.

PHASE II

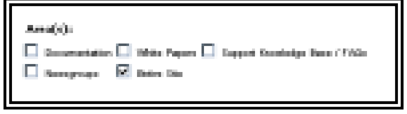
Soon BEA identified answerFriend, Inc. as a better match to address the challenges at hand. AnswerFriend's technology offers an advanced natural language processing engine that understands the meaning of a user's query and results in precise, highly accurate answers that take the user to the source of the answer in just two mouse clicks.

BEA became answerFriend's first large-scale customer to use the technology in a production environment. Due to this, the initial implementation and deployment stages of the application faced some challenges and required customizations to suit BEA's needs.

Despite the early challenges, answerFriend's technology did not require BEA to expend many person-hours defining preframed question/answers and, therefore, gave BEA the ability to roll out support for its products in just two months. As new products are released, the existing database of BEA concepts (ontology), along with other high-tech and English language concepts, can be utilized to provide good performance with minimum tuning. Concepts unique to the new product features can then be added to fine-tune the engine performance. This implementation has a tighter integration with newsgroups and is based on Network News Transfer Protocol (NNTP). It gives the user the ability to go to the specific article within a news thread that contains the possible match to the answer, lets the user post new queries and replies to existing articles from within the interface, and browse other posts and newsgroups. New articles on the newsgroups and other new documents on the site are picked up on a nightly basis, resulting in access to the most up-to-date information.

The answerFriend implementation of AskBEA is hosted in-house by BEA, runs

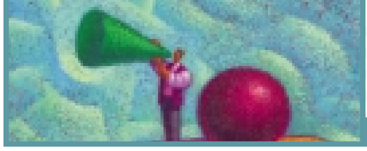
FIGURE 2



Area(s):

☐ Documentation ☐ White Papers ☐ Support Knowledge Base / FAQs
☐ Newsgroups [✓ Entire Site]

Knowledge/Data sources



on BEA WebLogic Server, and provides an easy-to-use interface with scalable, reliable, and fast access to the most accurate and highly relevant information from a single portal.

Customer Usability

AskBEA is a one-stop solution for searching disparate knowledge sources including extensive BEA product documentation, BEA customer support knowledge databases, the entire www.bea.com Web site, and the highly active BEA Usenet newsgroup community. With AskBEA, users can choose to search all of the information sources or restrict searches to only the sources they select. Customers have the opportunity to ask questions simply, such as "How do I set up connection pools in WebLogic Server?" or "What platforms does WebLogic Server support?" or "How do I update my license in WebLogic Server 6.0?" and be immediately directed to highly relevant answers.

FINDING WHAT YOU WANT

To ensure that the engine returns precise answers to the user's query, questions must be phrased appropriately. If you don't really know what you're looking for, or you don't know much about the content beneath the engine, you will receive a smattering of superfluous techno-substance. For example, if you go to Google at www.google.com and search for "computer help," you will notice that a search like this doesn't yield many useful results about computers. Also, how many clicks will you need before you really find what you're looking for? By combining a good question with the right keywords and specific data sources, you will be able to take advantage of BEA's always-improving knowledge base.

USING THE KNOWLEDGE SOURCES

AskBEA supports the following knowledge/data sources: Product Documentation, White Papers, Newsgroups, and Support Knowledge Base/FAQs. Each has a checkbox on the main AskBEA page. You can search as few as one or as many as all data sources. Let us briefly examine each source (see Figure 2).

Documentation

You can use AskBEA to search all available WebLogic documentation (the same documentation available at http://e-docs.bea.com.) If you have a question on a core function of WebLogic Server, the answer is usually in the documentation. In addition, a documentation search is especially useful if you are looking for help on a feature/topic that has changed extensively between the various versions of the product, as AskBEA will return documents from multiple documentation trees.

White Papers

These are the white papers on the Developer Center, http://developer.bea.com/docs/wp.jsp. This section currently has only a small set of documents, but it is getting bigger every day. All of these documents are indexed.

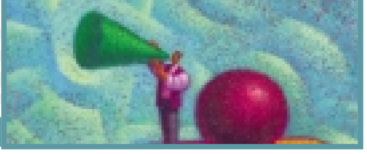
Newsgroups

As you probably know, there are more than three dozen WebLogic-specific newsgroups available at newsgroups.bea.com (http and news protocols supported). AskBEA indexes all of these posts as well since newsgroups are often monitored by WebLogic developers, product managers, and DREs. These folks answer complex WebLogic questions as well as questions related to application architecture, industry topics, and miscellaneous J2EE issues.

Support Knowledge Base/FAQs

BEA's Support Knowledge Base (SKB) for WebLogic Server is comprised mainly of "Solutions" authored by DREs. DREs are charged with creating Solutions based on recurring customer issues and commonly misunderstood topics involving WebLogic Server. A team of knowledge engineers verifies the correctness of these Solutions and, once accepted, they are published to the Web.

Internally, a Solution is also used to connect a customer support case to



an engineering change request (CR). When a CR is fixed, or a workaround is discovered, the DRE fills in the resolution field of the Solution and sends it to the Knowledge Engineering team for review and publication.

In addition to Solutions, this Knowledge Source contains other FAQs maintained by BEA as well as product updates and news available only to customers with a WebSUPPORT ID (if you do not have a WebSUPPORT login, you can get one here: www.bea.com/support/Web.shtml), This is the probably the most valuable knowledge source since Solutions are the direct result of a previously-solved problem involving both support and development.

When choosing which Knowledge Sources to search initially, you might think that it's best to just leave the "Entire Site" box checked. Frequently, though, searching specific sources is the better way to go. The following examples illustrate this.

THE RIGHT QUESTIONS

As explained on the *Tips for Asking page*, www.bea.com/support/askbea_tips.shtml, the easiest way to ask customer support questions is to use simple English and AskBEA will provide relevant answers. Other basics include: keep questions concise; include only relevant concepts; don't use boolean operators. By combining these techniques with proper WebLogic-specific keywords, you can hone in on the knowledge you seek.

Examples

Let's take three real-world WebLogic server examples, from simple to complex, so we can see how useful AskBEA really is. I will try to come up with a good question and identify the proper resource(s) to search and take you through my thought process. *Note*: These searches were performed in May 2001. AskBEA searches may return different results now.

Example 1

User wants end-of-life information about WebLogic. He's using 4.0.4 (uh oh!) and he wants to know when his support will run out.

The right question: clearly "end-of-life" is the important phrase here. Should we include "4.0.4" as well? Probably not necessary. If we find the "end-of- life" section, we'll probably find information on specific versions. "4.0.4" might clutter up the result set with 4.0.4-specific stuff. How about: **I need end-of- life information for WebLogic Server** (note that the question doesn't have to be a question, nor does the phrase "end-of-life" need to be in quotes.)

The right knowledge sources: chances are, product documentation isn't going to talk about this, but it might. You might find people talking about it on the newsgroups, but you probably want the official BEA answer on this one, so let's use the **Documentation** and **Support Knowledge Base/FAQs.**

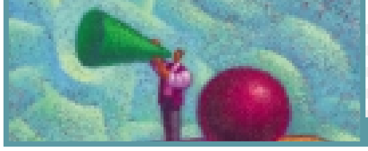
The result? The first link in the SKB/FAQ section is the one we want. It's labeled, "BEA Product News Updates: Product News Update: BEA WebLogic Server and WebLogic jDriver End-of-Life Information". The link takes us to a Web page (behind WebSUPPORT) which outlines the end-of-life for WebLogic Server and jDriver products. Note that this user is in trouble: 4.0.x was retired in April 2001.

Example 2

User has a question regarding the feasibility of proxying to multiple WebLogic servers with IIS. She hasn't been able to figure out how to set it up, so she's also considering using multiple instances of the IIS plug-in. She's tried to get that going too, but IIS keeps crashing.

This question has actually come up several times, and it turns out that there is a Microsoft limitation. Neither of the things she wants to do is possible (a workaround is to use Apache, which supports virtual hosting, or NES, which allows proxying to different WebLogic Server instances). Let's go through the process of identifying the right question so we can see how AskBEA can be of service.

The right question: any question will do as long as we get the right keywords: multiple IIS instances. The word *multiple* could match either case (trying to proxy to *multiple* WebLogic Servers or using *multiple* instances of IIS.) So we



could use: What are the known problems with multiple IIS instances?

The right knowledge sources: the answer could really be anywhere. If there is a known limitation it should be in the documentation. If it's not, there's probably a newsgroup discussion about it, or perhaps there's a Solution in the SKB. Let's search the documentation first.The result? Not much.

Try the newsgroups next. The result? A nice thread: "Subject: Re: Multiple WebLogic instances using IIS to proxy the requests" was returned. This post is especially enlightening:

We tried to do this & WebLogic doesn't support this config. We opened up a support call with them regarding this, but there is a problem with the way that IIS & NT proxies .JSP pages to the WebLogic servers. It's very unstable this way & I wouldn't recommend this config. We switched to Apache with virtual hosts to do this & it works fine.

It's probably not the answer you want to hear, but it works very well.

An SKB search returned nothing, so it looks like the newsgroup posting was the winner here. A support engineer will probably write a Solution for this though.

So you still might be thinking, "Why not search all the resources at once?" While you can always search "entire site," there are two main reasons not to, assuming you have an idea of the best resource:

- The more resources you include, the slower the search. It's often quicker to search one source, click back to your browser, select a different source, and search again.
- The more resources you search, the fewer results are displayed per resource per page (three for multiple resources versus five for a single resource). This will affect the next example.

Example 3

User is experiencing a LockTimedOutException and deadlocking with EJBs in WebLogic 5.1.

The right question: Why do my EJBs deadlock with LockTimedOutException?

The right knowledge sources: you could make a strong argument for newsgroups as well as SKB. Let's try the latter this time.

The result? The fourth link is: "WebLogic Server: S-06714 This deadlock is detected and after a certain timeout (five minutes by default), the deadlock is removed and one of the clients gets a LockTimedOutException." This Solution (anything in the form of S-0#### is a Solution) is exactly what we want. It contains an in-depth explanation of some of the problems that can occur because of the pessimistic locking scheme of EJBs in WebLogic Server 5.1. It also gives some good tips, workarounds, suggestions, etc. Note that if we had included additional resources in the search S-06714, the fourth result, would not have been shown on the first results screen. You would have had to click "More >>" to see it. In this case, three (duplicate) FAQ items (from various docs trees) were returned before the solution.

The answerFriend questionEngine on which AskBEA runs understands the meaning of the user's question and serves up precise, sentence-level answers with a single click that leads to the source documentation. This is a sophisticated, fast, and easy-to-use tool that gives customers the ability to get answers to their questions quickly, accurately, and painlessly. To further facilitate customer access to the final solution to their question, AskBEA includes three important features:

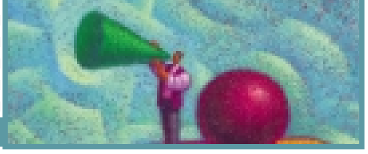
1. It returns convenient download icon locations for users to recognize the availability of downloadable software including service packs, etc.
2. Answers that display '*' indicate to customers that access to this information requires WebSUPPORT login.
3. Answers for multiple versions of the product are made available through a drop-down list.

For details on features and tips on using the service, please see:

- ***Features:*** www.bea.com/support/features.shtml
- ***Tips:*** /www.bea.com/support/askbea_tips.shtml

AskBEA helps to resolve most of the common issues customers run into and provides appropriate references on other issues. However, there may still be a need to obtain expert help from customer support. Some of the steps you take as an end customer before opening a case with BEA cus-

RECEIVE $150
DISCOUNT OFF FULL CONFERENCE
WEB SERVICES EDGE REGISTRATION
web services EDGE conference&expo
2002WORLDTOUR
Learn How to Develop
SOAP Web Services NOW!
at a One-Day Tutorial...Coming to a City Near You!



tomer support will ensure a fast, efficient, and effective resolution process to your issue. How long should you search before deciding to contact customer support? The recommendation is that you search three distinct questions per problem – and make sure to search all of the knowledge resources. If you do file a support case, give the DRE a head start by telling him/her what questions you have already asked AskBEA. Frontline support is actually tasked with asking AskBEA as its first means of finding a solution to your support case. Your specifying the relevant details will eliminate further lag caused due to DRE searching the AskBEA engine for results.

Increased AskBEA Usage

Today, the AskBEA online customer self-help service enables over 10,000 BEA customers (an estimated 250,000 developers) to efficiently access, from one place, precise and relevant answers to their questions. Since its initial launch in July 2000, AskBEA has shown significant results in the usage of the site and the impact on call volumes in customer support. Site activity is increasing steadily with the total daily number of queries on the site for weekdays ranging from 300 in July 2000 to nearly 3,000 in April 2001. The number of weekday queries averages around 2,500 per day. The total number of monthly queries has ranged from 11,000 in July 2000 to over 56,958 in May 2001, a 518% increase in usage of the site!

Positive Impact on Support Resources

As BEA has grown, the number of visitors to www.bea.com and other BEA Web sites has risen dramatically and the Web downloads of BEA products have also increased rapidly. However, the number of cases coming into support from customers evaluating BEA products declined from 20% in June, 2000, before the launch of AskBEA, to 8% in May 2001.

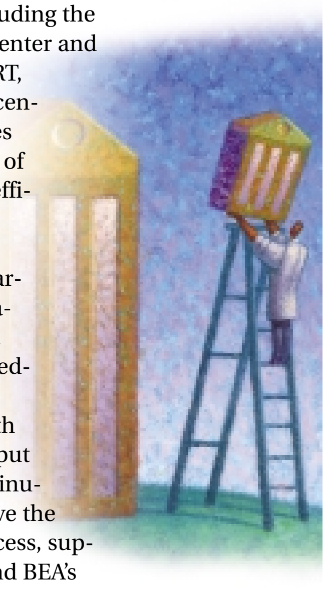
As user experience has grown with AskBEA, so has BEA's ability to improve the engine's performance. By capturing information on questions that return limited or no answers to the user, BEA knowledge workers are able to focus on and analyze those identified knowledge gaps and make continuous improvements in its knowledge sources. Combined with quick turnaround of rapidly changing content that occurs from newsgroup posts and documentation on new product versions, customers are viewing AskBEA as a very reliable source for their support needs.

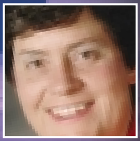
Site Locations

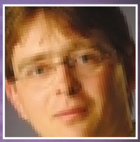
AskBEA can be accessed through the BEA Customer Support portal (www.bea.com/-support/index.jsp), the Developer Center (http://developer.bea.com/index.jsp), and the BEA Product Documentation center (http://e-docs.bea.com/).

The Bottom Line... 'Improved Satisfaction for BEA Customers'

AskBEA has been featured in many industry reports, including *Information Week*, *Network Computing*, and *ServerWatch*. This service is based on a solid foundation of scalable, reliable, and J2EE standards-based BEA WebLogic Server technology. AskBEA, along with other online services including the Developer Center and WebSUPPORT, offers user-centered features such as ease of use, speed, efficiency, and reliability to access the various information sources. Customer feedback is also solicited, with customer input used to continuously improve the support process, support sites, and BEA's products. Use of these services has improved customer satisfaction, reduced call volumes from evaluation customers, and stabilized the incoming rate of customer cases through increased use of online self-help service. This solution has proved an effective and efficient self-service strategy for BEA's customer support to handle the increasing demand for BEA products while continuing to provide world-class support.







WebLogic NEWS

Scient and British Air Deliver First Class Online Experience

(London) – Scient, Inc., and British Airways have announced the delivery of www.britishairways.com, an online experience tailored to the needs of travelers. British Airways partnered with Scient to rebuild its existing Web presence into one that offers travelers personalized service, better control over transactions, and easier navigation.

To help British Airways realize its goal , Scient assisted in the implementation of Interwoven's TeamSite Content Management System and BEA's WebLogic Personalization Server. www.scient.com, www.britishairways.com

Novell's Secure Partner Portal Solution Extends Strategic Partnerships

(Provo, UT) – Novell Inc has made available the Novell Secure Partner Portal, a solution aimed at helping companies build reliable and secure relationships cost-effectively with partners across the Net.

This portal is the second "Jump Start" solution launched by Novell and its Cambridge Technology Partners eServices division. It builds on Novell eDirectory, a secure and scalable identity management technology to manage partner access, and on Cambridge's expertise in the creation of business portals. BEA's WebLogic eBusiness Platform is the underlying infrastructure for the Secure Partner Portal. www.novell.com, www.ctp.com

BEA Systems and Andersen Ally to Leverage E-Business Platform

(San Jose, CA; Chicago) – BEA Systems, Inc., and Andersen, the global integrated professional services firm, have allied to accelerate the deployment of Web-based business capabilities to clients worldwide. Andersen will provide built-on-BEA solutions to companies across a range of industries to optimize complex transaction processing and to integrate legacy and Web-based applications on a single, standards-based platform.

The WebLogic E-Business Platform is an application infrastructure for technology solutions tied to business strategy. It enables Andersen to speed the delivery of solutions for Web-based revenue models and improved enterprise-wide efficiencies. www.andersen.com

BEA Systems and BMC Software Announce Global Alliance

(San Jose, CA; Houston, TX) – BEA Systems, Inc., and BMC Software, Inc., a provider of enterprise management software, have announced a global strategic alliance to provide mutual customers with the tools that ensure peak performance and reliability of critical e-business applications and processes. BMC Software selected WebLogic as the infrastructure provider for its GuardianAngel because BEA demonstrated advantages in meeting BMC Software's performance and reliability requirements.

BMC Software's new Web-based monitoring solution, GuardianAngel, is built on WebLogic. SiteAngel, a Web-based service that enables companies to manage customer experiences on e-commerce sites, is a built-on-BEA application that is platform- and browser-independent and can be accessed from any location via the Internet. www.bmc.com

Accenture's Open Technology Solution Speeds Implementation

(New York) – Accenture has announced a new Java application framework for BEA WebLogic E-Business Platform that makes it easier for companies to build and manage their e-business infrastructure.

Accenture's General and Reusable Netcentric Delivery Solution (GRNDS) application framework is based on J2EE and leverages WebLogic's powerful combination of portal, application, and integration servers. It is an extensible and upgradable architecture with a robust toolkit that is designed to speed application rollouts. The toolkit is designed to simplify the J2EE environment through consistent use of practical design patterns, enable multichannel access for new capabilities, and reduce architecture development costs by 50 percent or more. www.accenture.com

Sonic Software Provides Secure Messaging Infrastructure for WebLogic Customers

(Bedford, MA) – Sonic Software has integrated SonicMQ, its award-winning Java Message Service technology product, with BEA WebLogic. The integration combines the power of BEA's application server with a reliable, secure, and scalable Java-compliant messaging middleware.

BEA customers can also increase the reliability and security of Web services with SonicXQ, which combines standards-based, any-to-any connectivity and support for business processes on a message-based platform. www.sonicsoftware.com

Goodlot.com Chooses WebLogic to Power Global Online Lottery

(San Jose, CA) – BEA Systems, Inc., has announced that online charity lottery Goodlot.com has chosen the BEA WebLogic E-Business Platform to host its new global lottery. Goodlot.com is a charity lottery that will raise funds for globally recognized non-profit causes such as the Red Cross, UNICEF, and the World Wildlife Fund. It's backed by some of the world's most respected ambassadors, including Nelson Mandela, the retired president of South Africa and Anna Roosevelt, chair of the Roosevelt Institute.

The BEA WebLogic E-Business Platform was selected because it can scale to handle huge volumes of secure transactions. Goodlot.com estimates that in 2002 it will host more than 20,000 transactions a day. www.goodlot.com.

CocoBase Accelerates WebLogic 6.1

(San Francisco, CA) – THOUGHT Inc., a leader in Object to Relational (O/R) Mapping Technology, has announced the release of CocoBase Enterprise O/R, Dynamic Mapping for the Enterprise, Version 3.1 Service Release 13 with automatic transactional cache memory persistence. Previously, customers had to manually configure CocoBase caching to integrate properly with BEA. Now, BEA customers can create high-performance distributed J2EE applications automatically, reducing WebLogic's overhead and significantly increasing its performance.

The CocoBase caching system is a fully extensible design that allows customers to choose between an in-memory, nondistributed or distributed memory and/or disk caching implementation. Since the CocoBase caching system can disk persist, it doesn't require the startup overhead associated with typical caching systems. www.thoughtinc.com softwired

www.softwired-inc.com

sitraka

www.sitraka.com