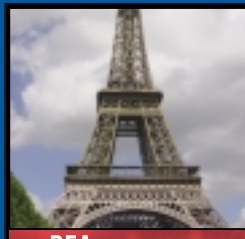


BEA WebLogic

DEVELOPER'S JOURNAL

AUGUST 2002 - Volume:1 Issue:8

weblogicdevelopersjournal.com



BEA world 2002

TURNING TECHNOLOGY INTO PROFIT
by Andy Winskill page 50

FROM THE EDITOR

The BEA Slayer?

by Jason Westra
page 5

GUEST EDITORIAL

Diagnosing Tough Performance Problems

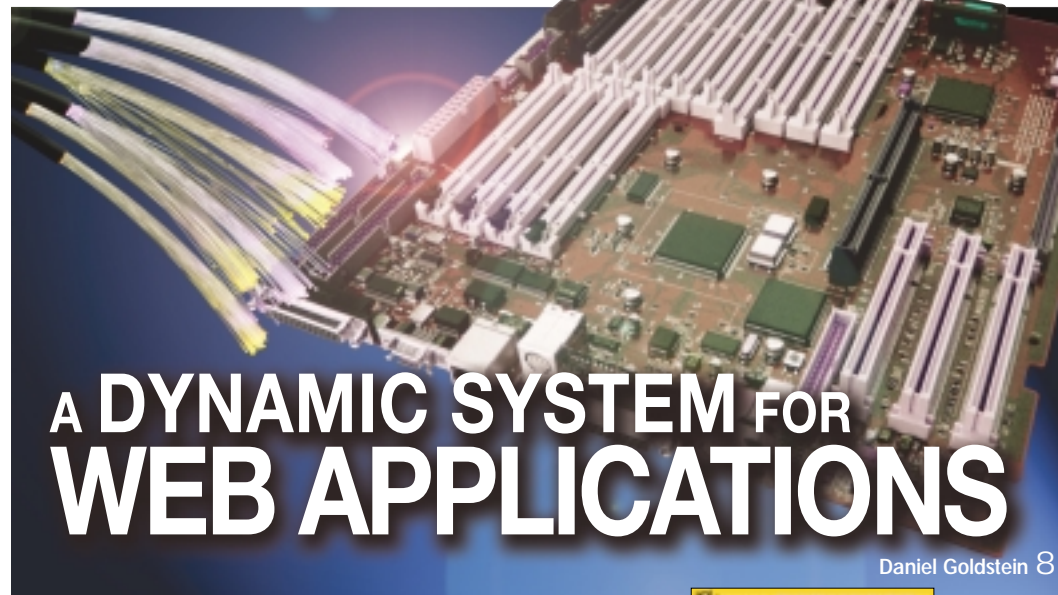
by Ashutosh Tiwary & Przemyslaw Pardyak
page 6

TRANSACTION MANAGEMENT

Show Me Some Commitment

by Peter Holditch
page 24

SYS-CON MEDIA

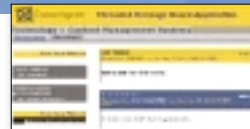


A DYNAMIC SYSTEM FOR WEB APPLICATIONS

Daniel Goldstein 8

J2EE: Large-Scale Financial Apps & Service-Oriented Architectures

Building with the BEA WebLogic Platform



12

Anwar Ludin

INTEGRATION: Advanced JMS Design Patterns for WebLogic Server Environments

Addressing the issues



18

Hub Vandervoort & Jake Yara

PERFORMANCE: The Art of Capacity Planning

Guidelines for WebLogic Server



26

Srikant Subramaniam & Arunabh Hazarika

EJB QL: Implementing WebLogic QL

Eliminate code and simplify your development



32

Michael Gendelman

CONTENT MANAGEMENT: BEA WebLogic & Interwoven

Two packages with a simple integration path



36

Travis Wissink

WLS CERTIFICATION: Making the Grade

JDBC, transactions, and clustering



40

Dave Cooke

WLDJ FEATURE: WebLogic on the Mainframe, PART 2

Install, configure, and deploy

44

Tad Stephens & Eric Gudgion

BEA
<http://developer.bea.com>

Wily Technology
www.wilytech.com

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 EDITORS
JAMIE MATUSOW, JEAN CASSIDY

ASSISTANT EDITOR
JENNIFER STILLEY

WRITERS IN THIS ISSUE
DAVE COOKE, MICHAEL GENDELMAN, DANIEL
GOLDSTEIN, ERIC GUDGION, ANURABH HAZARIKA,
PETER HOLDITCH, ANWAR LUDIN, PRZEMYSŁAW
PARDYAK, TAD STEPHENS, SRIKANT SUBRAMANIAM,
ASHUTOSH TIWARY, HUB VANDERVOORT, JASON
WESTRA, TRAVIS WISSINK, JAKE YARA

SUBSCRIPTIONS
For subscriptions and requests for Bulk Orders,
please send your letters to Subscription Department.
SUBSCRIPTION HOTLINE:
888-303-5282

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

COO/CFO
MARK HARABEDIAN

VP, BUSINESS DEVELOPMENT
GRISHA DAVIDA

SENIOR VP, SALES & MARKETING
CARMEN GONZALEZ

VP, PRODUCTION & DESIGN
JIM MORGAN

ART DIRECTOR
ALEX BOTERO

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

ASSISTANT ART DIRECTOR
TAMI BEATTY

VP, SALES & MARKETING
MILES SILVERMAN

ADVERTISING SALES DIRECTOR
ROBYN FORMA

ADVERTISING ACCOUNT MANAGER
MEGAN RING-MUSSA

ASSOCIATE SALES MANAGERS
CARRIE GEBERT • ALISA CATALANO
KRISTIN KUHNLE • LEAH HITTMAN

VP, SYS-CON EVENTS
CATHY WALTERS

CONFERENCE MANAGER
MICHAEL LYNCH

REGIONAL SALES MANAGERS
MICHAEL PESICK • RICHARD ANDERSON

ASSISTANT CONTROLLER
JUDITH CALNAN

ACCOUNTS PAYABLE
JOAN LAROSE

ACCOUNTS RECEIVABLE
JAN BRAIDECH

ACCOUNTING CLERK
BETTY WHITE

WEBMASTER
ROBERT DIAMOND

WEB DESIGNERS
STEPHEN KILMURRAY • CHRISTOPHER CROCE
CATALIN STANCESCU

ONLINE EDITOR
LIN GOETZ

CUSTOMER SERVICE MANAGER
ANTHONY D. SPITZER

CUSTOMER SERVICE REPRESENTATIVE
MARGIE DOWNS

EDITORIAL OFFICES
SYS-CON Publications, Inc.
135 Chestnut Ridge Road, Montvale, NJ 07645
Telephone: 201 802-3000 Fax: 201 782-9637
SUBSCRIBE@SYS-CON.COM

BEA WebLogic Developer's Journal (ISSN# 1535-9581)
is published monthly (12 times a year)

Postmaster: Send Address Changes to
BEA WEBLOGIC DEVELOPER'S JOURNAL,
SYS-CON Publications, Inc.
135 Chestnut Ridge Road, Montvale, NJ 07645

© COPYRIGHT 2002 BY SYS-CON PUBLICATIONS, INC. ALL RIGHTS RESERVED. NO PART OF THIS PUBLICATION MAY BE REPRODUCED OR TRANSMITTED IN ANY FORM OR BY ANY MEANS, ELECTRONIC OR MECHANICAL, INCLUDING PHOTOCOPY OR ANY INFORMATION STORAGE AND RETRIEVAL SYSTEM, WITHOUT WRITTEN PERMISSION. FOR PROMOTIONAL REPRINTS, CONTACT REPRINT COORDINATOR: SYS-CON PUBLICATIONS, INC., RESERVES THE RIGHT TO REVISE, RE-PUBLISH AND AUTHORIZE THE READERS TO USE THE ARTICLES SUBMITTED FOR PUBLICATION. ALL BRAND AND PRODUCT NAMES USED ON THESE PAGES ARE TRADE NAMES, SERVICE MARKS OR TRADEMARKS OF THEIR RESPECTIVE COMPANIES. SYS-CON PUBLICATIONS, INC., IS NOT AFFILIATED WITH THE COMPANIES OR PRODUCTS COVERED IN WEBLOGIC DEVELOPERS JOURNAL.

BEA WebLogic Developer's Journal (ISSN# 1535-9581)
is published monthly (12 times a year)

Postmaster: Send Address Changes to
BEA WEBLOGIC DEVELOPER'S JOURNAL,
SYS-CON Publications, Inc.
135 Chestnut Ridge Road, Montvale, NJ 07645

SYS-CON
MEDIA



BY JASON WESTRA
EDITOR-IN-CHIEF

The BEA Slayer?

There are numerous news groups and discussion lists on the topic of J2EE. I follow several regularly to track trends from the industry's news and views. The mindset I've seen over the past six months has been one of "topple the giant," a modern-day "Jack the Giant Slayer." In "Jack the Giant-Slayer," Jack, a young villager, steps up to the task of ridding the countryside of evil giants. Against all odds, he successfully slays the evil giants in various clever ways. Interestingly, there were three giants in the story. The application server market has three giants as well – BEA, IBM, and Oracle. In our modern-day fairy tale the part of Jack has been played by many, including open-source efforts such as JBoss and JONAS, the HP Application Server, and Sun's new Sun ONE J2EE AppServer 7.

While Jack has been played by different application servers, they all have one thing in common – they're free. JBoss and JONAS are free, open-source implementations of the J2EE specification, while the HP Application Server and the Sun ONE J2EE AppServer 7 are commercial servers bundled free with each vendor's operating system.

With the maturity of the J2EE specification and the advent of free implementations, it seems that BEA is poised to lose a game it has been winning for years. Surely, server license revenue will falter when competing with free implementations, right? Not so fast! These products may support the specifications, but I say, "Who cares?" BEA is no longer competing on this level – the level of standards support. BEA has raised the bar up to the level of the platform. To compete in today's market, you need to deliver a platform that is integrated seamlessly across portal and content management, core server functionality, security, integration, and administration. The free implementations currently support the major releases of the specification, but they're generally six months to a year behind and aren't production-worthy for another three months after standards support. Also,

while open-source implementations can be made to play nice with each other, even when integrated they don't make up a platform. They don't have a single install, they don't have a single administration or security model, and each has its own form of administration, management, and configuration.

Bundled? Hah! A bundled server is one that nobody would buy in the first place, let alone one upon which you'd deploy a mission-critical application. Let me give you a historical perspective on bundling application servers. In my Year 2000 predictions in my column for *Java Developer's Journal*, I predicted there would be three application servers left standing by the end of 2000 – BEA, IBM, and Oracle. BEA would be standing because it's the best. IBM would hold a large market share because it is "Big Blue" and it had many loyal customers to convert from its heavy iron to new, sleek Web technologies. Oracle would be a close third because it was the most widely licensed database. By bundling its application server inside its database, Oracle would be able to persuade customers to use its app server rather than buy extra licenses from another vendor.

I was right about the giants, although Oracle didn't achieve that status until a year later when it licensed the Orion container. In any case, Oracle's bundling strategy didn't work. Even if a product comes bundled for free, no one is going to use it if it doesn't work well. Needless to say, I'm not sweating that HP and Sun are bundling their servers with their hardware.

This month in *WLDJ* we have more articles, content, and examples demonstrating BEA's WebLogic Platform, its easy administration, and its content management capabilities. This isn't an open-source effort discussion list about what WebLogic could become. These articles cover what WebLogic already is – a platform. At this stage of the game, BEA owns the platform, and I predict it'll be hard for the Jacks of the world to slay this giant.

AUTHOR BIO...

Jason Westra is the editor-in-chief of *WLDJ* and the CTO of Evolution Hosting, a J2EE Web-hosting firm. Jason has vast experience with the BEA WebLogic Server Platform and was a columnist for *Java Developer's Journal* for two years, where he shared his WebLogic experiences with readers.

CONTACT: jason@sys-con.com

Diagnosing Tough Performance Problems

J2EE APPLICATIONS UNDER REALISTIC LOAD

BY ASHUTOSH TIWARY
& PRZEMYSŁAW PARDYAK

Although many of the symptoms of performance problems (e.g., poor response time) are similar throughout the application life cycle, the underlying causes and the techniques used to diagnose them become more complex in later stages as the load increases and the configuration becomes more complex. In this article, we discuss the tools and techniques that are useful in diagnosing tough performance problems that occur under realistic high loads. We also illustrate why tools used in development or under limited load conditions are not suitable for finding such tough performance problems.

Reproducing Performance Problems Under Load

There are two principal kinds of performance problems: **persistent problems** that affect the performance at all times, and **transient problems** that occur intermittently and for a limited amount of time. The former are typically easy to reproduce as they can be triggered by high enough workload. The latter are harder to reproduce because the right configuration, state, and workload are required to reproduce them. To diagnose tough performance problems under high load, we need a reliable way of reproducing the problem and the ability to examine the internals of the application under load.

Persistent performance problems (e.g., the Empty-Cart transaction always has poor response time) are often reproduced using load-testing tools. In contrast, current techniques for reproducing transient problems (e.g., response time of the entire system quadrupled for 15 minutes at 2:00 p.m. yesterday) involve guesswork and ad hoc testing in order to approximate the original configuration and load in the production environment. Using current approaches, it may take anywhere from weeks to months to reproduce and diagnose transient problems. An alternative is to use **real workload technology**, which records the production workload (including all transaction requests and responses) when the transient problem occurs, and plays it back in the test environment in order to reproduce the problem. This reduces the time to reproduce and ultimately solve these problems from weeks and months to hours and days.

AUTHOR BIOS...

Ashutosh Tiwary has 12 years of software development and performance consulting experience at Boeing, Hewlett-Packard, and Teknekron Communications Systems. He is a PhD candidate in computer science at the University of Washington, where his dissertation work forms the basis for Performant's technology.

Przemysław Pardyak codeveloped Performant's core technology and has eight years of research and development experience including performance management. He is a PhD candidate in computer science at the University of Washington.

Diagnosing the Root Cause

Once the performance problem is reproduced, diagnosing the root cause requires drill-down analysis that correlates external symptoms with potential root causes. This involves:

- **Finding** the bottleneck or the transactions, beans, servlets, and methods that consume the most time
- **Breaking down** the aggregate time spent in a type of transaction across the servlets, beans, and methods it uses
- **Correlating** individual transactions from an HTTP request, through servlet and bean calls, and down to JDBC calls and SQL statements to find the chains of invocations that produce slow response time
- **Examining** individual thread execution profiles to see which threads were the bottleneck and where the time was spent

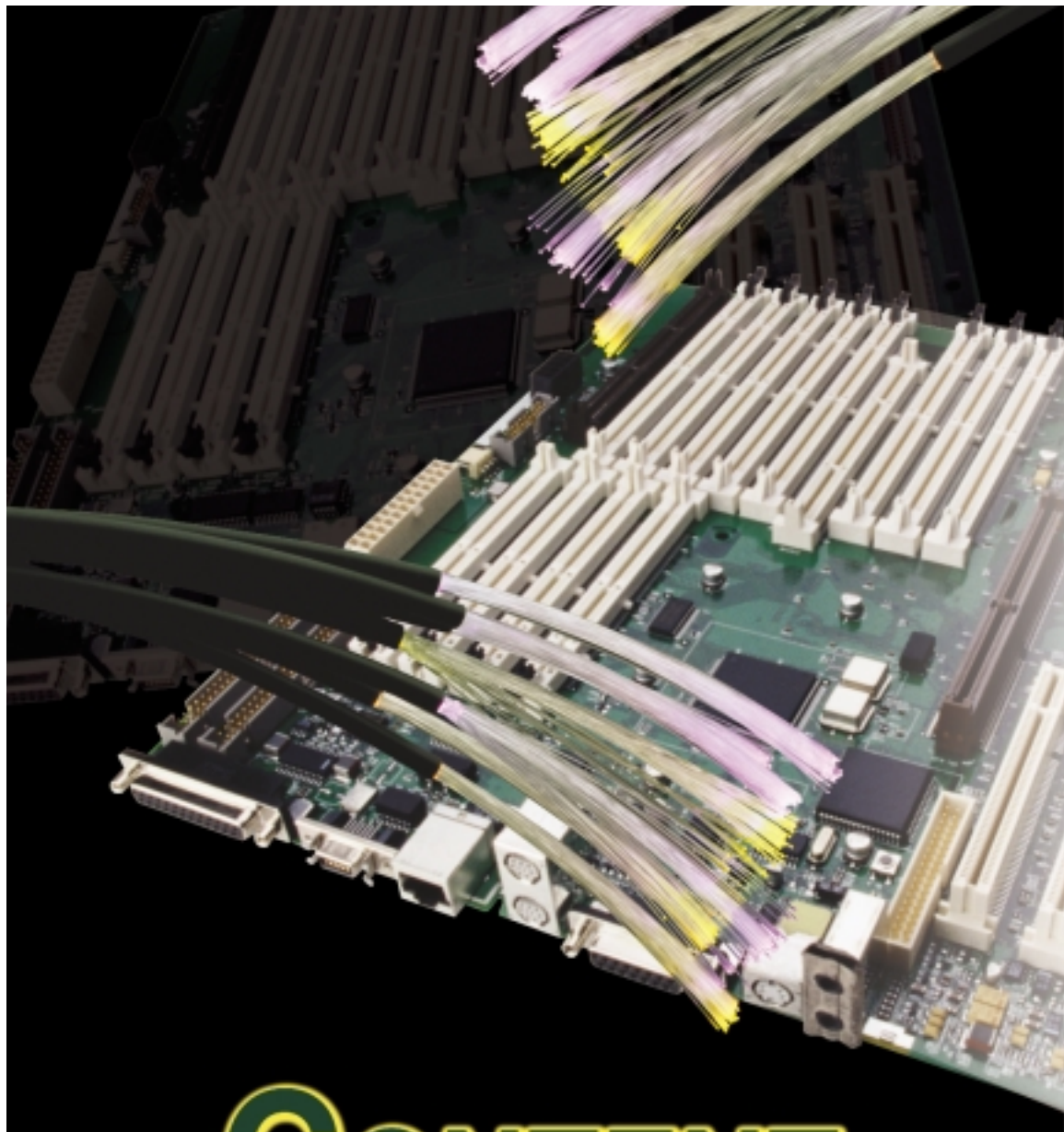
Profiling tools used by developers often provide aggregate information (e.g., statistical summaries of calls to methods) that helps in such analysis. However, they don't isolate individual calls or objects, nor do they gather information about arguments and results. Additionally, profiling tools that are based on the JVM profiling interface (JVMPPI) suffer from a significant overhead (between 200 and 1,000%). This performance impact makes loading the application nearly impossible and significantly skews results. Profiling tools based on JVM sampling have lower overhead (around 100 to 200%) but provide even less information. Such tools can be used under higher load but are less helpful in diagnosing tough problems. Byte code instrumentation technology can provide an arbitrary level of detail, and its overhead can be managed by limiting the scope of instrumentation without sacrificing the level of detail. When properly implemented, gathering data using byte code instrumentation can have overhead as low as 5 to 50%.

Production monitoring tools instrument a small subset of Java methods and report summary information on beans and methods that take the most time.

continued on page 43

CONTACT: atiwary@performant.com
ppardyak@performant.com

Panacya
www.panacya.com



BY DANIEL GOLDSTEIN

AUTHOR BIO...

Daniel Goldstein, PhD, director of developer marketing for FatWire Corporation, has held visiting scholar positions at Stanford University and Harvard University. Daniel's Internet background includes development work for America Online and Rolling Stone.

CONTACT...

goldstein@fatwire.com

CONTENT MANAGEMENT

WEBLOGIC AND A DYNAMIC CONTENT MANAGEMENT SYSTEM FOR WEB APPLICATIONS

This article outlines how to build a typical content-based Web application, a threaded message board. In recent years, threaded message boards

have gained popularity due to their ability to draw visitors back to a site with the appeal of fresh content. Message boards also make financial sense.

Unlike the costly process by which Web teams create their own content, a threaded message board automates publication and delegates it to site users. Additionally, companies have achieved significant cost reduction in the area of technical support by creating message boards where communities of users can answer one another's support requests.

Since a threaded message board sits atop a database of content, a dynamic or database-driven content management system drastically speeds the creation and operation of such a Web application. This article reviews the basics of building a threaded message board on the BEA WebLogic application server and the FatWire UpdateEngine content management system, which is database-driven and compatible with the Microsoft SQL Server, Oracle, and IBM DB2 databases. With WebLogic, UpdateEngine automates many of the steps involved in creating a threaded message board, such as defining database tables, creating edit screens, and making vendor-specific SQL and workflow calls.

Anatomy of a Threaded Message Board

The basic unit of the threaded message board is the message, or post, which is a short snippet of text with authorship information. Figure 1 shows the main interface of a message board.

Every message belongs to a thread, which is a post about a new topic and its subsequent replies. The left frame of Figure 1 displays the two threads called "New thread" and "Application Accelerators." From this main interface, users can create a new thread, select a thread to view its messages, or reply to a message within a thread. If the users begin a new thread or reply to an existing thread, they are presented with an HTML form to enter the message content.

Building the Dynamic Threaded Message Board Application

Building and running the message board consists of four steps:

1. Design data structures
2. Create tables in the database
3. Program template logic to run the board
4. Manage content

DESIGN DATA STRUCTURES

The first step in developing a dynamic Web application is designing the database structures that will hold the content. Surprisingly for a message board, we can get by with only one data structure (content class in UpdateEngine parlance): that of the basic message or post. The messages on this board will contain the fields shown in Table 1.

CREATE TABLES IN THE DATABASE

In the UpdateEngine GUI, the developer types in field names and selects data types (from the 22 available in a drop-down list). From this point, UpdateEngine's Content Definition wizard creates database tables automatically, eliminating the need to do this programmatically in a database-specific SQL dialect. If new fields need to be added or removed from a table at any stage of the development cycle, this can also be done on the fly. Creating the table definition for this message board takes about 5 minutes with the wizard. Figure 2 shows the wizard interface with the example fields entered.

PROGRAM TEMPLATES TO RUN THE MESSAGE BOARD

Templates are programs that control the display of the message board and move content into and out of the database. Templates are written in JSP and invoke the UpdateEngine API. This API allows the

developer to write high-level Java code to carry out queries, as well as content management and workflow operations, and saves development time by avoiding database-specific SQL or workflow operations. The same templates run on any database compatible with UpdateEngine. Four templates need to be written to power our message board.

Display_Threads

This template runs a database query to get all threads and display them by the order of how recently they were updated. Listing 1 shows a pseudo-code example of how the query could look in the UpdateEngine API.

The ordered list can now be displayed as a thread list, and clicking on a thread should call Display_Posts for that thread. Display_Threads should also present the user with the option to create a new thread. Doing so amounts to calling the Create_Post template, passing in information that a new thread (as opposed to a reply) is created.

Display_Posts

Like displaying threads, displaying messages comes down to a simple query. Run a query to get all posts that match a given thread (by checking the POST_PARENT field), and then order them by date authored. An example is shown below:

```
/*Create a query to find posts whose parent is the current thread,
/*pointed to by dbidCurrentPost */
Query q = new Query(POST_PARENT, Query.EQUALS, dbidCurrentPost);
/*Run the query over all posts. The result is a list of pointers to
records */
DBIDList dbidPost = getDBIDs(POST,q);
```

This template should also provide an option to reply to the thread. To do this, create a link that calls the Create_Post template, passing in the information that a reply (and not a new thread) should be created.

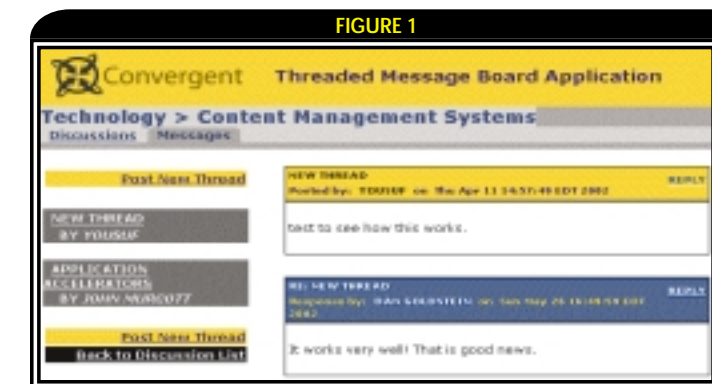


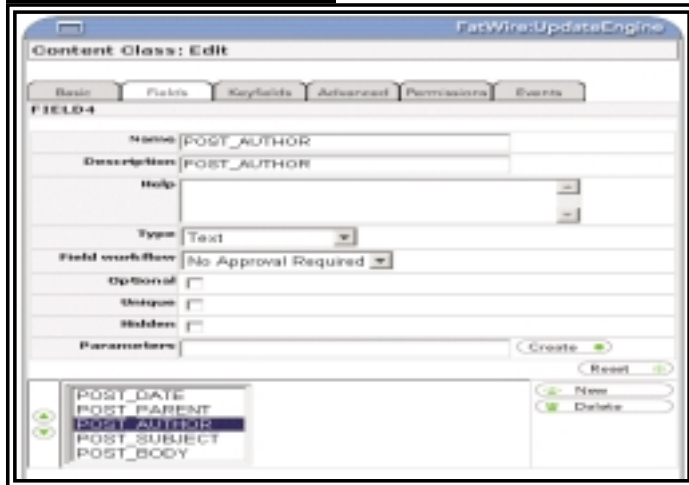
FIGURE 1 Main interface of message board

TABLE 1

CONTENT CLASS: POST		
FIELD NAME	FIELD DESCRIPTION	DATA TYPE
POST_DATE	Posting date	Date
POST_PARENT	A link to the initial post in the thread (or a NULL value if it the post defines a new thread)	Links to another instance of POST
POST_AUTHOR	Poster's name	Text
POST_SUBJECT	The subject of the post	Text
POST_BODY	The body of the post	

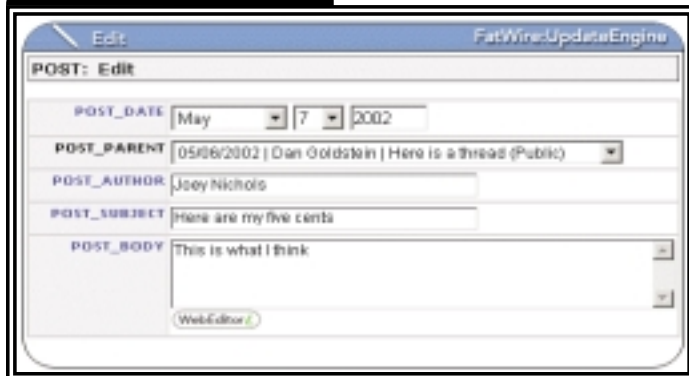
Fields in message board

FIGURE 2



Wizard interface

FIGURE 3



Automatically generated edit screen

FIGURE 4



UpdateEngine browse screen

reply to an existing thread, the POST_PARENT field is set to point to the initial message in the thread.

Once the database structure is created and the four JSP templates function as desired, the message board is up and running.

MANAGE CONTENT

No back-end CMS is needed to operate the message board; however, there are occasions where it's desirable to be able to manage board content to remove or edit objectionable posts, to change user names or thread names, or to append editor's notes. Usually, creating a form to edit messages would take additional programming; however, in UpdateEngine forms for entering and viewing content are automatically generated for each database table created by the Content Definition wizard. The automatically generated edit screen for our message board is shown in Figure 3.

To access a data-entry form, first locate the message to edit. Messages may be located from the UpdateEngine browse screen, shown in Figure 4, and they can be sorted on by a number of key fields.

Alternatively, the moderator can access a record using UpdateEngine's search or Power Search tools, which allow Boolean searches over any fields in the database. Once the desired record is identified, it can be checked out (to prevent other moderators from editing it simultaneously) and sent to the edit screen.

If board moderators want to have a higher level of control over what appears on their site, all new messages can be placed into a workflow process instead of being posted live. Workflows are defined using the UpdateEngine GUI. In this example, two stages, "unapproved" and "live," will suffice, but any number of levels of field- and item-level workflow can be created. Using the Java API, the Process_Post template should be modified so it sets the workflow stage to "unapproved," and the display templates should be modified to show messages only in the "live" stage. Now messages will only appear on the board after an authorized moderator has approved them from the UpdateEngine browse screen.

Conclusion

Developing content-rich Web applications without manually programming database tables, edit screens, and workflow processes saves time. Database-specific SQL commands and workflow operations are passed over in favor of interaction with a GUI and a vendor-neutral Java API. With WebLogic and UpdateEngine, a developer can extend the simple message board described here by adding and redefining database tables in order to create a threaded message board as complex as any found on the Web.

Listing 1

```

/* Define a query to find all threads, that is, posts without predecessors.*/
Query q = new Query(POST_PARENT, Query.EQUALS, NULL);

// Define an order by when the post was modified
Order o = new Order(CCCColumns.sModTimeColName);

/* getDBIDs runs the query and ordering over the POST content class, which holds all the messages. A DBID is merely a pointer to a record in the database, and a DBIDList is a list of such pointers.*/
DBIDList dbidlPost = getDBIDs(POST,q,o);
    
```

Create_Post

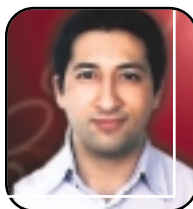
Create_Post is simply a form that calls Process_Post. If Create_Post is to create a new thread, a subject field should be displayed on the form. If it is being called in order to reply to an existing thread, the subject line should be hidden and its value should be set to that of the referring post with "Re:" as a prefix.

Process_Post

The Process_Post template receives the contents of the form displayed by the Create_Post template, as well as information about what is being created: a new thread, or a reply to an existing thread. In either case, a new message is created in the database and the content fields are instantiated. If the post is the beginning of a new thread, the POST_PARENT field is set to NULL. If it's a

Sitraka
www.sitraka.com/performance/wldj

LARGE-SCALE FINANCIAL APPLICATIONS & SERVICE-ORIENTED ARCHITECTURES



BY
ANWAR LUDIN

AUTHOR BIO...

Anwar Ludin is CTO of ExoSonic, which provides financial institutions with a reusable framework of design patterns and EJB components for developing and deploying multichannel, mission-critical distributed enterprise applications and Web services.

CONTACT...

anwar.ludin@exosonic.com

Financial institutions have heavily invested in J2EE and component-based architectures. WebLogic server is a key e-business platform for deploying innovative financial services to clients and trading partners. The emerging service-oriented architecture paradigm is a natural extension to J2EE, and WebLogic integrates it seamlessly.

Component and Business-Process Reuse

Component reuse is the key to building large-scale business-to-business (B2B) financial services; this article illustrates the benefits of doing so using the WebLogic platform. The most fundamental shift introduced by e-business is that the core business logic is now located and managed on the application server. Furthermore, business logic is now accessible from various presentation channels. WebLogic is a powerful platform for enabling centralized business-process reuse. If you provide the business processes to an Excel spreadsheet, you simply have to encapsulate it as a SOAP service, and WebLogic greatly simplifies the task.

Application Topology and Business Patterns

Taking a step back, it's interesting to note some of the essential e-business patterns emerging in finance. In its simplest form, a business pattern describes the interactions between users, businesses, and data. In that sense, a pattern-driven approach to building e-business architectures around WebLogic is very interesting. A J2EE architecture, in a nutshell, can be mapped to different business patterns such as the user-to-business, or "self-service," pattern; the user-to-data, or "information aggregation," pattern; and the B2B, or "extended enterprise," pattern.

Once you've identified the e-business pattern your financial service follows, the next step is to map it to an application topology, which in turn maps to a runtime topology. This article will concentrate mainly on the user-to-business pattern for financial services but will also address the extended enterprise with applications such as Web services.

For the user-to-business pattern, two application topologies have been identified. In topology 1, your application doesn't interface with legacy systems or third-party applications. This typically happens when you build a "pure" J2EE application that doesn't require the WebLogic Server to interface with legacy systems. In topology 2, your application interfaces with legacy systems or

third-party applications. This is most common in financial services that require access to real-time data feeds, ticker plants, and third-party portfolio management systems.

Runtime Topology

Once an application topology has been identified, it must be mapped to a runtime topology, which is eventually deployed as the e-business solution. The following is a nonexhaustive list of the essential financial services for the user-to-business pattern:

- Access to real-time financial news and data feeds
- Notifications and alerts on events such as trades or quote values
- Real-time trading of financial instruments
- Portfolio valuation and reporting

Not surprisingly, customers have come to expect these services to be delivered seamlessly on mobile devices with the same level of functionality they would have using a Web browser. Financial services have to be designed with a multichannel architecture from the ground up. Figure 1 illustrates a runtime topology that I've found effective when building large-scale, multi-tier financial applications with WebLogic.

Nodes

A runtime topology consists of several nodes, which identify specific functions such as firewalls, Web servers, and directory services like LDAP. Each node is important in building an enterprise trading system around WebLogic.

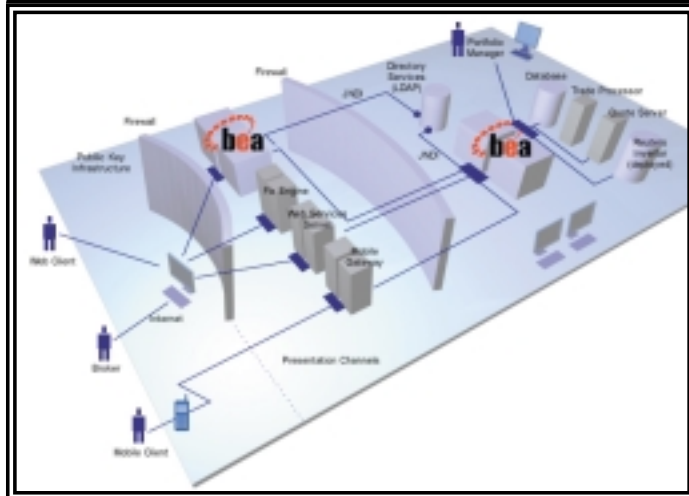
The nodes in the "outside world" represent different actors that interface with the financial business processes. An actor can be a private or institutional client, another application communicating through Web services/SOAP, a brokerage firm using the FIX protocol to book and acknowledge trade orders, or even a mobile client. One or several actors initiate most of the system use cases.

A first firewall node restricts access from the Internet and filters traffic based on a set of rules. For example, most banking applications use HTTPS encryption with a firewall enabling only certain IP addresses and Port 443. A second firewall node restricts access from the demilitarized zone (DMZ) to the internal corporate network. The presentation channels and other nodes, such as mobile gateways or FIX engines, are located in the DMZ.

The Web server node (WebLogic in this topology) hosts financial Web applications. In most cases, the applications provide customized front ends for services such as portfolio management,

BUILDING WITH THE BEA WEBLOGIC PLATFORM

FIGURE 1



Runtime topology

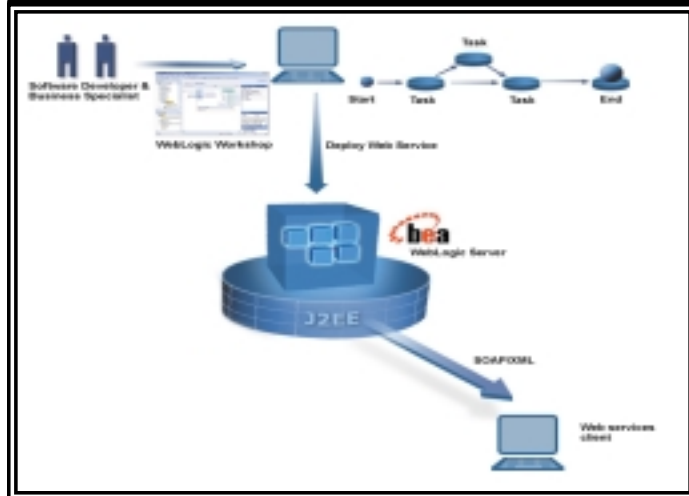
e-reporting, and access to financial data and news. Clients use the Web channel; however, most business logic processing is delegated through RMI/IIOP to EJB components located behind the second firewall.

The FIX engine node is a highly specialized messaging engine for producing and consuming messages encoded in FIX, an industry-wide open protocol developed specifically for real-time, seamless electronic exchange of securities transactions. FIX enables you to provide straight-through processing, which reduces transaction costs by virtually eliminating phone calls and faxes between traders and brokers. Usually, a FIX engine must meet the same high-availability criteria as the other runtime nodes; therefore, most engine vendors incorporate clustering and failover in their products.

The Web services server node provides the infrastructure to map SOAP invocations to EJB session facades hosted behind the second firewall. Do this using RPC-style invocations where an inbound SOAP message arrives at the Web services server. The server parses the message and invokes an EJB session facade using RMI/IIOP, which in turn manages local calls to session and entity beans hosted by WebLogic. This illustrates the benefits of packaging reusable financial processes as EJB components, which can be leveraged by Web applications and by SOAP clients. RPC-style invocations are ideal for providing pricing or portfolio optimization analytics as Web services, which offers many new opportunities to share processes internally or externally.

The mobile gateway node provides the bridge between enterprise financial processes and mobile platforms such as Symbian, Palm OS, and Windows CE. With the advent of more powerful devices and richer user interfaces, mobile users will expect the same level of functionality provided by desktop applications. WML hardly delivers in that sense. Using a mobile middleware infrastructure based on JMS is an interesting approach. Such a solution relies on three components: WebLogic as a JMS provider for the financial business logic, data synchronization, and security; a mobile JMS gateway; and a very lightweight JMS client library residing on the mobile device. iBus//mobile from Softwired, Inc., (www.softwired-inc.com) seamlessly integrates with WebLogic for building mobile services with JMS. Because mobile platforms have problems such as intermittent connections and broken communication links, providing queuing facilities and reliable messaging is a necessity – JMS messaging is the ideal candidate.

FIGURE 2



Software development process

The bulk of financial processes are located in nodes behind the second firewall. In this case, the WebLogic Application Server node is a central repository of reusable financial business processes packaged as session beans, entity beans, and message-driven beans. The ultimate goal, component reuse, is achieved by identifying the fundamental services recurring in all large-scale financial applications and encapsulating them as EJB components. Component reuse can be hard to achieve; however, today's fast-paced market leaves no choice other than relying on a solid, well-tested set of essential components.

The directory services node is essential for creating a single sign-on infrastructure. Single sign-on not only eliminates the hassle of managing multiple user logins and passwords, it greatly simplifies overall administration tasks. An LDAP directory can also store generic information such as e-mail addresses or, in the case of strong authentication, user certificates. From an application development perspective, centralized data stores for user management are great because the same authorization and authentication business logic aren't reproduced in each application. A slightly different runtime topology would add a read-only "slave" LDAP directory located in the DMZ. The write master stays behind the second firewall, and replication is used to synchronize master and slave nodes. This solution has the advantage of scaling better.

The quote server, or "ticker plant," node is a real-time market data distribution system like the Reuters Triarch. The ticker plant uses TIBCO's messaging technology for dispatching JMS quote messages in real time from TIBCO Rendezvous messages to the client application nodes.

Client application nodes are GUIs used for managing portfolios or entering trade orders. Except for the processing of real-time quotes, the bulk of the business logic behind the GUIs is actually implemented as EJB components hosted by WebLogic and reused by the Web tier.

The trade processor node is a back-office system used for settling and clearing the trades. Financial business processes hosted by WebLogic send JMS trade messages to the trade processor, which after acknowledging a trade returns a JMS message to a message-driven bean.

Finally, the Reuters Investor node (<http://rfs.reuters.com/investor>) provides financial data and news in an XML format to financial components hosted on WebLogic. The Reuters Investor service and the quote server differ in that the Reuters Investor service is used

Precise
www.precise.com/wldj

for building financial portals from the flexible Reuters data feed and the quote server targets real-time financial data streaming for trading.

Using clustering at the different nodes, such as the FIX engines, mobile gateways, and WebLogic servers, enhances reliability and availability. Horizontal scaling is achieved through clustering. Adding resources such as CPUs and memory to the server machines achieves vertical scaling. In this topology, WebLogic is used as a Web server in the DMZ and as an application server in the internal network.

Service-Oriented Architecture

Using WebLogic in the DMZ makes sense, especially with the additional features provided by WebLogic 7.0 and tools such as WebLogic Workshop. The runtime topology is also simplified because WebLogic simultaneously plays the roles of clustered Web server and Web services node. However, there's an even better reason for using WebLogic Server to build service-oriented architectures. If you've played around with WebLogic Workshop, you've probably seen how easily Web services can be built and published. It isn't a secret; we all know the next step in enterprise computing is service-oriented architectures. The real issue is how to get there as painlessly as possible. Many believe service-oriented architectures represent a fundamental shift in the design of enterprise software – with a tool like WebLogic Workshop, it's more a question of extending and adapting enterprise software than a fundamental shift. After all, you still have to develop J2EE components, and you still have to apply J2EE patterns. The difference is that now you have to consider service facades.

In most cases, business processes are complex and require multiple asynchronous steps. Even the simplest of financial processes, such as a trade order, might engage complex workflows spanning multiple days and requiring the intervention of many actors. As illustrated in Figure 2, the real fundamental shift happens when you visualize software architecture and development as a process-centric discipline. Most interesting is the ability to build models of processes and publish them as Web services.

Putting WebLogic Workshop to the Test

To evaluate WebLogic Workshop as a process-centric software development tool, I “redeveloped” an application I'd helped build.

Bank A wanted to sell funds to Bank B. Bank B would in turn sell those funds to its clients by providing tailored portfolios. The process involved several complicated steps and business rules. In addition, Bank A's IT infrastructure was totally incompatible with Bank B's.

We devised the relatively inelegant schema of exchanging information by sending and receiving daily batches of files. The business-logic tier then processed the files to validate trades and orders. The process required a lot of manual intervention, and I had an “I-could-have-done-better” feeling at the end.

Using Web services to solve the problem would have been great, but until recently it was still considered experimental technology. With WebLogic Workshop, this is no longer true. I was impressed with how fast I could prototype and redesign the application as a set of Web services and gradually visualize the entire workflow. I showed the results to a business specialist with little knowledge of software engineering. Thanks to WebLogic Workshop's visual interface, he was able to provide feedback, and we modified and adapted the business model interactively.

Separation of Roles

The goal of the runtime topology is not only to provide a secure, scalable, and reliable environment, but also to promote a separation of roles between developers. The runtime topology in Figure 1 provides a clean separation of the business logic components hosted by WebLogic and the presentation logic components. Developers with strong knowledge of EJB and WebLogic programming build the reusable business components. Developers working on Web services and presentation channels can then leverage and integrate them in the presentation tier.

An important issue is the performance considerations resulting from intertier method invocations and data transfers. J2EE design patterns are an invaluable source of information for avoiding such pitfalls. I recommend Floyd Marinescu's book, *EJB Design Patterns*, as a source of patterns and best practices for building distributed systems using EJBs.


Simple Use Case

Finally, let's look at a typical use-case scenario. Consider a user accessing a financial institution's Web site. A single sign-on procedure automatically authenticates the user. From this point on, he or she has access to resources according to group permissions. Personalized information such as reports, financial data, and news is provided through WebLogic Portal and portlets.

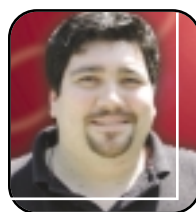
The portfolio portlet gives the user an instant overview of his or her current assets as well as an updated gain/loss situation. The recommendations portlet displays current recommendations according to the user's profile. The user reviews the recommendations and enters a trade order with a limit condition, thanks to the trading portlet. The user logs off, and the order is delegated to the business-tier EJBs.

At the same time, a trade notification JMS message is sent to the portfolio manager monitoring the user. When the portfolio manager logs in to the application, he or she acknowledges the trade order and another JMS message is dispatched from the client GUI application to the trade processor. After some processing, the trade processor returns a JMS acknowledgement message to the business tier, which in turn translates the message to another JMS message recognized by the FIX engine and dispatches it to it. The FIX engine then packages the trade order as a FIX message and sends it to the broker. After some processing, the broker returns a confirmation message, which is translated in a JMS message and returned to the business tier. A notification is sent to the user on his or her mobile phone.

Conclusion

Over the past year, financial institutions have heavily invested in J2EE architectures for adapting and opening their internal IT infrastructure to the Internet, trading partners, and clients. WebLogic Server has become an essential component of large-scale financial applications and essential services such as online trading or portfolio management. Service-oriented architectures are the logical next step for building highly decoupled service-centric and business process-centric solutions. However, current investments in J2EE should be leveraged, and service-oriented architectures should be an evolution of the J2EE component-based model rather than a revolution. Tools such as WebLogic Server and WebLogic Workshop not only preserve your current investments but also facilitate the transition to service-oriented architectures. 

Altaworks
www.altaworks.com



BY
HUB VANDERVOORT
& JAKE YARA

AUTHOR BIOS...

Hub Vandervoort is vice president of Professional Services for Sonic Software. He has more than 20 years of experience as a consultant and senior technology executive in the networking, communications software, and Internet industries.

Jake Yara is an independent software consultant working with Sonic Software to implement application server-specific solutions for the leading EJB application servers on the market. In February 2000 he founded Yara, Inc.

CONTACT...

hvanderv@sonicsoftware.com
jakeryara@alum.mit.edu

ADVANCED JMS DESIGN PATTERNS FOR WEBLOGIC SERVER ENVIRONMENTS

Addressing the issues

BEA WebLogic Server (WLS) is the most widely deployed J2EE application server and continues to be an attractive platform for many enterprise-class situations.

As part of its support for J2EE 1.3, WLS now embeds a Java Message Service (JMS). The implementation is straightforward; using many built-in WLS features, it proves worthy at moderate

scale in predominantly WLS environments.

Despite the inclusion of a new JMS implementation, there remain instances where heterogeneity, scalability, and availability requirements go beyond native support, calling for increased sophistication provided by third-party messaging solutions. For example, when multiple brands of application servers, C clients, or non-Java legacy applications are involved, a solution with broader platform coverage is required. If specialized features like client-side persistence or large-message support are necessary, niche solutions may be most suitable. If extreme throughput scalability or high availability via clustering, routing, and failover are critical, enterprise architects must turn to pure-play messaging vendors.

While implementing a third-party JMS has many advantages in highly scaled, mission-critical settings, it carries a bit more complexity than simple out-of-box configuration – especially when Enterprise JavaBeans (EJBs) and XA Container-Managed Transactions (CMT) are involved. The issues stem from two principal concerns: first, the manner in which WLS 6.x implements the Java Transaction API (JTA) and Java Transaction Service (JTS); and second, the way JMS and J2EE application servers interrelate with respect to runtime and development-time questions. On the first issue, cooperation from the JMS vendor is necessary for a solution (unless the JMS exposes its internal transaction APIs – read: open source). On the second issue, various design patterns can augment JMS vendor-provided tools for a complete solution.

This article explores third-party JMS integration with WLS and addresses the two issues outlined above by exposing Sonic Software's approach to implementing the SonicMQ WLSAdapter. We'll review how XA transactions implemented by the SonicMQ JMS are properly mapped to the nonstandard WLS JTA and illustrate how to deal with problems surrounding client reconnect, JNDI loading, exception handling, and complex EJB interactions.

WLS XA Transactions with Third-Party JMS

Global Transaction (XA) support, or CMT, presents unique, complex issues. When WLS EJBs use a foreign JMS to send or receive messages, and both attempt to participate in a common XA transaction, it usually fails. This is because the WLS 6.x JTA and JTS are implemented in a nonstandard way.

According to the JMS specification, a JMS provider must comply with the JTA specification and its rules for creation and use of XA resources, namely the XAConnectionFactory, XAConnection, and XASession objects. Similarly, the JMS must conform to the JTA standard for providing XAResource objects directly from an XASession object (for more details see Chapter 8 of the JMS 1.0.2.b Specification at <http://java.sun.com/products/jms/docs.html>).

Virtually all third-party JMS products meet

these criteria, and they expect the application server, which acts as a JMS client, to adhere to the same JMS specification for a successful XA-compliant operation. WLS addresses four issues dealing with XA transactions.

1. No XAConnectionFactory, XAConnection, or XASession

When a J2EE application server sets up an XA transaction, it's supposed to use the XAQueueConnectionFactory or XATopicConnectionFactory objects from an external JMS provider to create XAConnection and XASession objects. Instead, WLS creates only standard JMS connection and session objects. As a result, steps specific to XA transactions and "Chapter-8 compliance" aren't implemented.

2. XAResource Is Never Enlisted

In order for an XA transaction to take place, an XAResource must be enlisted to begin the transaction. Because WLS doesn't use the third-party JMS provider XAConnectionFactory objects, it can't properly obtain an XAResource object to enlist. Thus, in a message-driven bean (MDB), where the enlistment should take place automatically under control of the application server, the external JMS XAResources are never actually enlisted into the CMT. Likewise, a session bean can't explicitly enlist the XAResource either – an XAResource can't be created in the first place.

3. Proprietary Session Interface

WLS requires that the session object of the external JMS implement a specific proprietary interface, `weblogic.jms.extensions.MDBTransaction`. This interface contains a method, `associateTransaction()`, that will be called by WLS to join the JMS transaction manager in the CMT. Because this interface isn't standard, third-party JMS providers must include an adapter to implement the method. Internally, WLS checks to see that a session object supports the interface; if not, the container throws an exception.

4. XA Transaction Branches Aren't Serialized

According to the JTA specification, all branches of a common XA transaction must be in serial order. However, WLS manages transaction branches in a way that allows a second transaction branch, provided to the container by an outboard transaction manager, to begin before other transaction branches have completed. This occurs because WLS transactions start and end branches in different threads. The "start" thread may have completed its actions, allowing the next transaction branch to start, while the "end" actions from the first transaction branch may not yet have completed. JMS 1.0.2-compliant products expect XA branches to be in serial order according to the JTA specification and consider any overlapping transaction branches to be a violation of JTA protocol. Thus, compliant JMS implementations will throw



an exception when transactions are permitted to overlap, unless the JMS vendor provides an alternative nonstandard implementation.

Enter WebLogic Server Adapter (WLSAdapter)

Despite the issues, third-party JMS providers who intend to participate in the WebLogic market have clear incentives to provide integration solutions. Most vendors do this by including adapters to reconcile differences between JMS Chapter-8 compliance and the “isms” characteristic of the current WLS release. Adapter implementations are distinguished not only by the robustness with which they address XA functionality, but also by how they carry through to enable truly elegant and scalable solutions.

Enter the WLSAdapter from Sonic Software – a solution that resolves integration issues between WLS and SonicMQ, a leading enterprise messaging platform. This implementation addresses other key integration challenges as well, including:

- Automating client reconnect and JNDI loading
- Facilitating development-time failure injection for testing of exception-handling logic
- Providing a robust set of template design patterns for sophisticated JMS architectures

XA Transaction Integration

If an application server is to provide XA transactions alongside a third-party JMS, it must be able to:

- Create and manage JMS connections and sessions
- Create and manage XAConnections and XASessions
- Enlist the XAResource.

While WLS can create and manage JMS connections, it can't create XA transactions or enlist XAResources. To resolve this problem, WLSAdapter creates and manages all XAConnectionFactories, XAConnections, XASessions, and XAResources.

When the WLSAdapter creates an XASession, it also creates a single XAResource object inside a JVM (singleton), which is presented to the WLS transaction manager for each XA transaction. The real XAResources, provided within the JMS transaction manager, are created by WLSAdapter and registered with the singleton, which in turn delegates all WLS transaction manager calls to the appropriate SonicMQ XAResource. The adapter manages all SonicMQ XA branches internally. In MDBs, the WLSAdapter automatically enlists the XAResource on each transaction, while session beans explicitly enlist XAResources via the WLS TxHelper, `javax.transaction.Transaction tx = weblogic.transaction.TxHelper.getTransaction()`, and a direct call to the `enlistResource(myXAResource)` method.

The result is equivalent to the JMS adapter providing a small transaction manager to participate as a single resource for the WLS transaction manager. And, since an XA resource can now be obtained for the WLS transaction manager, issues 1 and 2 are resolved.

The WLSAdapter addresses issue 3 by implementing the BEA proprietary interface, `MDBTransaction`, in its session object and uses the `associateTransaction()` method to trigger automatic enlistment from MDB transactions. Because the method is now implemented in the adapter XA session object, the WLS exceptions are no longer thrown.

Finally, the WLSAdapter addresses issue 4 by managing XA transaction branches so they remain completely serialized. It does this by catching SonicMQ exceptions whenever WLS allows a transaction branch overlap, and holding the overlapping branch until the previous one has completed. Only then does it permit the next transaction branch to be started. Exceptions are masked, branch overlap is avoided, and XA transactions conform to the JTA Specification.

With the WLSAdapter for SonicMQ, all four of the WLS XA-compliance issues are resolved, providing a fully JTA-compliant Global Transaction solution within an enterprise-class JMS.

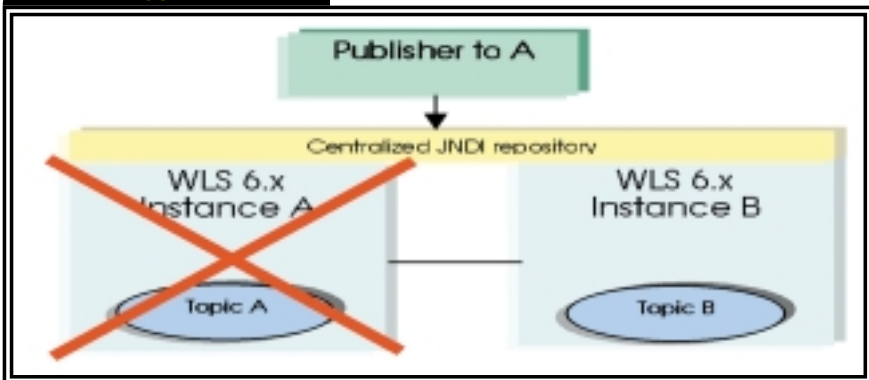
Other Integration Challenges

Runtime and Development-Time Considerations

The introduction of an adapter presents an opportunity to incorporate various value-added features that enhance the robustness of an enter-

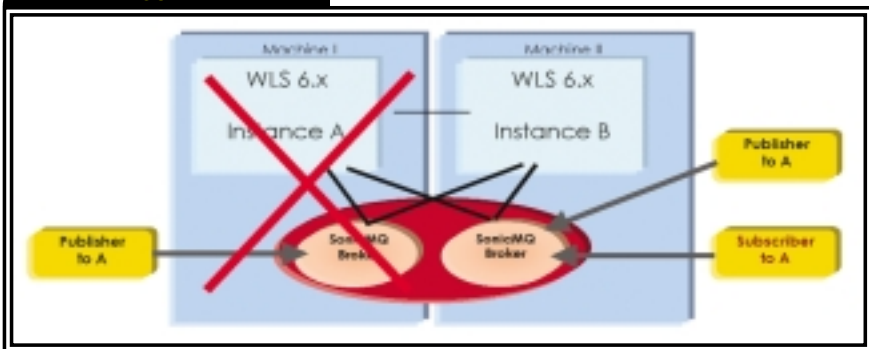
Sonic Software
www.sonicsoftware.com/websj

FIGURE 1



Typical clustering configuration

FIGURE 2



WebLogic cluster and SonicMQ cluster working together

prise solution. Key considerations include automating runtime reconnection around failed brokers or links and providing facilities for testing application-level exception handling (particularly connection- and transaction-level recovery logic).

AUTOMATIC CLIENT RECONNECT

Reconnection and failover capabilities are important to any large-scale distributed application. Both WebLogic and SonicMQ provide clustering and reconnection capabilities. Figure 1 represents a typical WebLogic clustering configuration with two servers in a cluster.

Topic A and Topic B can be accessed from any WLS in the cluster. However, the topics must be

created in advance, and topic information is tied locally to a particular server

In the event Instance A goes down, traffic will be routed to the remaining server in the cluster. However, Topic A will no longer be accessible.

Figure 2 shows a WebLogic cluster and a SonicMQ cluster working together to leverage the reconnect capabilities of WLSAdapter.

In this example, if either of the WLS instances fails, all topics will still be available through SonicMQ. In addition, if either of the SonicMQ brokers goes down, they'll automatically fail over to the backup server or servers, and topic information won't be lost. When WLSAdapter is used in a SonicMQ cluster, topics can be dynamically created and information is shared across the JMS cluster

To accomplish a reconnect in the first example (which doesn't use WLSAdapter), a producer or a synchronous consumer in an EJB would have to trap the exception inside the application logic when the broker goes down and explicitly reconnect and reconstitute all the connection and session objects

WLS does recreate a reconnection within an MDB; however, the listener isn't recreated and the messages can be lost. The solution provided by WLSAdapter:

- Completely hides the connection failure from the user
- Relies on the URL list to reestablish a connection
- Ensures the session is recreated as well as the producers and consumers

Since the connection failure and the reconnect are completely transparent to the user and business logic, the developer is freed from writing code to manage reconnect scenarios.

Failure Insertion Points

WLSAdapter can insert failure points for debugging and analysis. The advantages geared to development time address one of the most daunting aspects of integration testing – introducing controlled but realistic failures to test exception-handling pathways. WLSAdapter provides two types of failure injection: connection drop failure points and method failures.

CONNECTION-DROP FAILURE

A connection-drop failure pauses the application in a specified method. This way, an external action can be taken by the developer to have an effect at a specific point in the execution of the application.

For example, to test application behavior when a broker goes down or a connection fails, randomly pulling cables or “killing” a broker won't provide a reliably reproducible test case. However, using the SonicMQ WLSAdapter, you can isolate the failure occurrences by forcing a connection-drop failure to occur during the execution of a particular method, for example the onMessage() method of an MDB. When the

method is called, the application will be paused by the adapter. At that point you can force a broker to disconnect and simulate the problem. After releasing the failure point, execution of the application resumes to reveal how the exception logic copes with the situation.

METHOD FAILURE

A method failure is similar in design, but it simply forces a particular exception to be thrown from a specific method rather than requiring developer intervention (such as killing a broker or pulling a cable). When a method failure point is set, the application will pause when it reaches the method specified. There the option is offered to have that method raise an exception, or continue normally.

There are 10 methods in which you can set a connection-drop or method failure. Five of them are for JMS methods, and five are specifically for XA transactions. An application simply needs to implement the method in question for the WLSAdapter to execute it and inject a failure point; thus, no temporary test logic needs to be embedded within the application code. Table 1 lists the JMS and XA methods where failure insertion may occur.

Using failure point insertion in WebLogic is as simple as including the WLSAdapter package in the classpath of the WebLogic domain startup and adding the appropriate flags to the application server start-up:

```
-DSonicsw.WLSAdapter.Test.DoFail=true
-D Sonicsw.WLSAdapter.Test.FailType=%1
-D Sonicsw.WLSAdapter.Test.FailPoint=%2
```

The parameter %1 would be either “ConnectionDrop” or “MethodFailure”, and %2 would represent one of the methods listed in Table 1.

Enterprise-Class Design Patterns

Value-added features can be included with adapter packages in templates that implement advanced transactional design patterns. Often enterprise-level EJB scenarios employing JMS require several beans to interact transactionally. These complex scenarios necessitate clever construction for robustness and some form of session pooling for scalability.

For example, a common circumstance might be to have an MDB acting as a consumer to a Pub/Sub topic listening for RFPs. When an RFP message arrives, it might process the document into a local database via an entity bean, and then a session bean to produce a proposal for transmission over a point-to-point queue back to the proposal requestor. In a high throughput environment, the outbound queue transmission might pool session beans to improve scalability. In this complex arrangement, the MDB, entity bean, and session beans must all become part of

a single transaction, where two of the beans are interacting with the JMS transaction manager.

The WLSAdapter uniquely answers questions surrounding these enterprise-grade problems. It does so by including prebuilt code templates that implement design patterns suitable for supporting multibean interactions. Solutions are illustrated for consumers and producers, along with a robust, yet lightweight, pooling mechanism for stateless session beans.

Looking Ahead

There are five primary transactional-JMS design patterns included in the Sonic WLSAdapter (illustrated in Figure 3). These include:

- Message-Driven Bean
- Message-Driven Bean Router
- Message-Producing Bean
- Message-Consuming Bean
- Message-Consuming Bean Router

TABLE 1

JMS FAILURE POINT METHODS	XA FAILURE POINT METHODS
send	start
receive	end
onMessage	prepare
commit	commit
rollback	rollback

Methods where failure insertion may occur

Conclusion

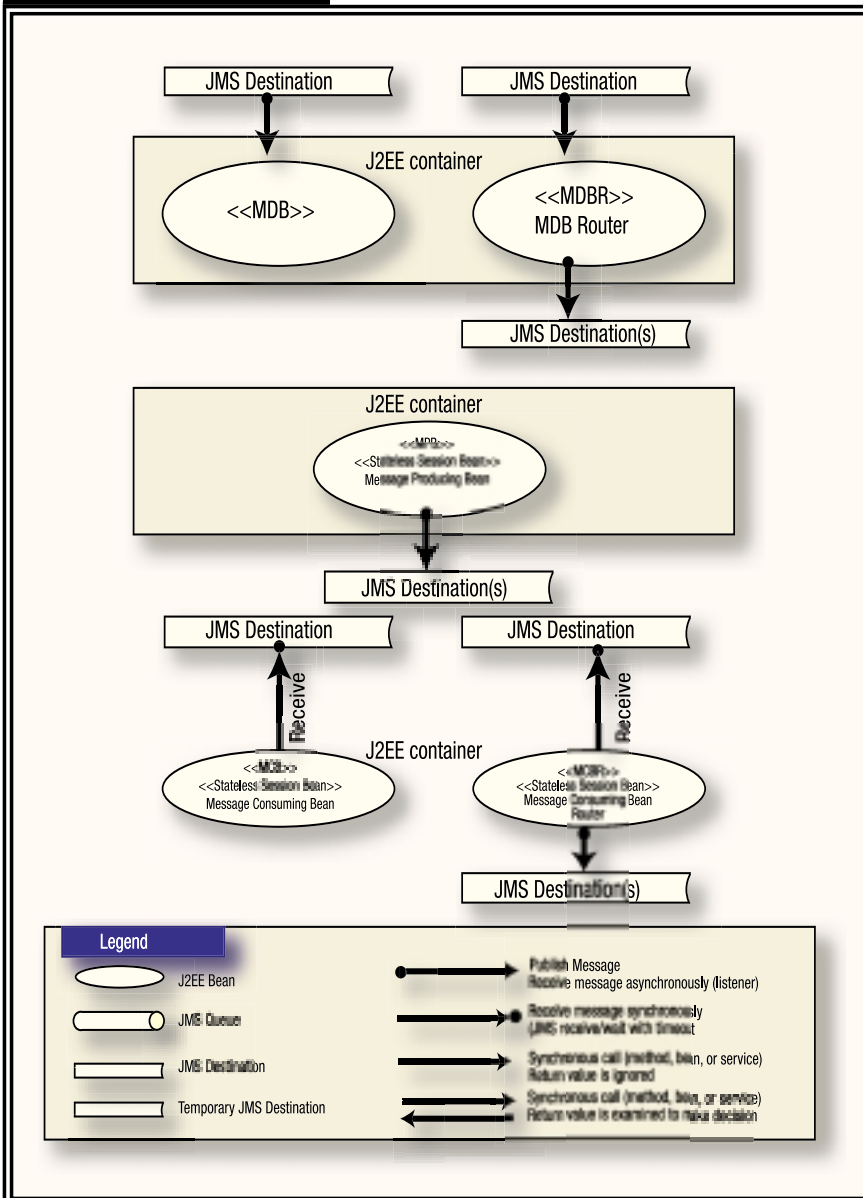
In part 2 of this article we'll describe the implementation of a few of these design patterns inside WLS using SonicMQ and the WLSAdapter as the JMS solution. The specific implementations covered will include:

- Message-driven bean (with a queue listener)
- Message-driven bean (with a topic listener and reconnect)
- Message-consumer bean
- Message-producing bean (with cross-JMS domain XA and reconnect)

We'll summarize with a description of a composite design pattern employing a multibean interaction along the lines of the RFI/RFP example described above.

Given these issues, one might ask: How does the WebLogic embedded JMS work and how does it pass Sun's J2EE 1.3 CTS? The answer comes in two parts: First, Sun CTS only certifies API-level compliance. While that implies a lot about internal functionality, there's no specification for implementation. Which leads to the second part of the answer: the embedded JMS shares the same transaction manager (TM) as the rest of the WLS container. Having only one TM to deal with, WLS is able to perform all the XA integration under its own hood (and in a nonstandard way) while still meeting CTS API-level conformity.

FIGURE 3



Core design patterns included with the SonicMQ WebLogic Server Adapter (WLSAdapter)



For several months now I've waxed lyrical about transactions and how they hide the complexities of distributed updates in applications, and indeed the concept of the two-phase commit transaction is a very powerful one, allowing you to make the assumption that all the transactional data stores that hold your business data will roll forward or back in lockstep.

Show Me Some Commitment – Connecting with Transactions

GETTING “UNDER THE COVERS” FOR WHAT YOU REALLY NEED

BY PETER HOLDITCH



However, as with all “under the covers” behavior (if you’ll excuse a mixed metaphor), sometimes you need to get inside the kimono to get what you really need.

For example, what if you’re using a bean-managed transaction from a stateful session bean to update state represented by a number of entity beans (a common case)? Although the transaction will ensure the consistency of the entity beans, since the state in the session bean is not held in a transactional resource, it is not impacted whatsoever by the transaction. Now consider the case where the transaction rolls back for some reason. Although all the databases will look as though the transaction never happened, the session bean may have state set in it that makes sense only if the transaction committed.

This sounds a bit desperate! What should we do? Throw out the stateful session bean and keep all the state in entities? Well, we could, but if we have no requirements for long-term persistence of the state, there’s no reason to weigh a database down with it, and to split the state from the function would break the encapsulation that the session bean provided. You might think that there must be a more elegant solution, and there is...

JTA provides a session synchronization inter-

face, `javax.ejb.SessionSynchronization`. This is how you, as an application developer, can request that JTA open its kimono to you. Through this interface you provide three callback functions that the JTA subsystem can use to notify your code of what it’s doing behind the scenes. The three methods provided are `afterBegin()`, `beforeCompletion()`, and `afterCompletion()`.

To digress a moment for the pedantic, this interface is actually provided by the EJB container. Only the latter two methods map directly onto JTA interfaces as such. In the context of a raw user of JTA, an `afterBegin` method would make no sense, since the transaction would begin under the control of the application code. In the EJB world, a container-managed transaction could be begun on behalf of the logic by the container, and the `afterBegin()` notification is the EJB container’s way of letting your code know that this has happened. The “raw” JTA equivalent of this interface is in fact called `javax.transaction.Synchronization`.

Anyway, back to the plot... through the synchronization interface, your stateful session beans can register for notification of when the “under the covers” transactional behaviors take place. If you’ll excuse another brief digression, I have a colleague at BEA who has a pathological hatred of stateful session beans. He recommends using them in only two cases: if you need to keep session state in a J2EE application and can’t rely on an HTTP front end (and hence, can’t use an HTTP session object), or if you need to respond to transactional events – while an HTTP session object can’t easily register for these events, the EJB container provides the machinery to do this easily. The fact that this particular individual can be persuaded to use a stateful session bean at all indicates the potential power of linking to JTA in this way.

So, as you by now suspect, the typical use case of the `SessionSynchronization` interface is a stateful session bean with its `beforeCompletion` method containing an implementation that flushes any state that may have been cached before the commit processing begins (of course, if state is cached when the transaction is committed, then it will never be written to the back-end storage). The `afterCompletion` implementation, when the final result is a JTA rollback, will likely return the session state to some values saved prior to its commencement.

As a final note, the `afterBegin` and `beforeCompletion` methods are called in the scope of the transaction to which they refer, so

any updates they make to transactional stores will be included in the transaction, along with whatever updates are caused by the business logic. Clearly, the `afterCompletion` method cannot be called in the context of the transaction since it has by that stage, by definition, completed.

“GROUNDHOG DAY” FOR JTA

That’s all well and good – using the session synchronization interface we have a way to make our stateful session beans’ state pseudo-transactional, and we have a placeholder to allow for session state to be cached at application level should this be necessary. Before the story ends and we live happily ever after, however, a few moments would be well spent thinking through the implications of that innocuous-looking previous paragraph...

Consider again that the `beforeCompletion` method must be called before the two-phase commit process can start. Also remember that the whole point of JTA is that an arbitrary number of objects can be bound together in a single transaction. Given this, it will quickly become apparent that the job of the JTA subsystem at commit time is to call all the `beforeCompletion` methods registered with the transaction before commencing the conventional two-phase commit processing with the underlying data stores also registered with the transaction. It is this characteristic that has sometimes led object-based transaction monitors to be described as performing a “two- and-a-half-phase commit.”

This sounds simple until you realize that not only is the list of `beforeCompletion` methods dynamic, but since all of them are called in the context of the transaction themselves, any of them could cause the list itself to grow. Imagine the scenario where stateful session beans S1 and S2 both implement `beforeCompletion` methods. A business method on S1 is called, causing a transaction to be started and resulting in two entity beans, E1 and E2, being involved in the transaction.

Now it’s commit time. The JTA subsystem goes to commit the transaction. For the first “half phase” it sees that the `beforeCompletion` method on S1 needs to be called. Imagine that this method calls the second session bean, S2. During the first half phase another `beforeCompletion` method has been added to the list. S1’s `beforeCompletion` method returns, and the JTA subsystem realizes that S2’s `beforeCompletion` method is now in the list and needs to be called, so JTA makes the call. If S2’s `beforeCompletion` method has no side effects, then on its return JTA can start the two-phase commit proper. However, if S2’s `beforeCompletion` method has invoked a method on S1, then suddenly S1’s method is back in the list. Despite the fact that it has already been called, the state of S1 may have been changed by

the latest call, so it will need to be called again.

You can see from this that we have created *Groundhog Day* for the JTA subsystem. As fast as it tries to cross `beforeCompletion` methods off its list, new ones are added. This will cause an infinite loop, which may provide entertainment for anyone who has sadistic inclinations toward application servers – for most normal people, this will just be a bad case of bad news. In fact, I have bad news for the sadists too... the WebLogic developers thought of this, which provides an explanation of one of the more obscure JTA configuration parameters, the “`beforeCompletionIterationLimit`”. This parameter is set on a JTA-wide basis by the administrator and limits the number of times that JTA will loop around trying to cross `beforeCompletion` methods off a transaction’s list before it will just give up and abort the transaction. Clearly, identifying one of these infinite loop, *Groundhog Day*-type situations by code inspection as a developer requires a view of the implementations of all the `beforeCompletion` methods in a transaction and is likely to be pretty difficult to spot and fix, particularly in situations where old beans are being reused in new transactional scenarios.

“ If we have no requirements for long-term persistence of the state, there’s no reason to weigh a database down with it”

Based on this, I would propose a best practice for the implementation of `beforeCompletion` methods.

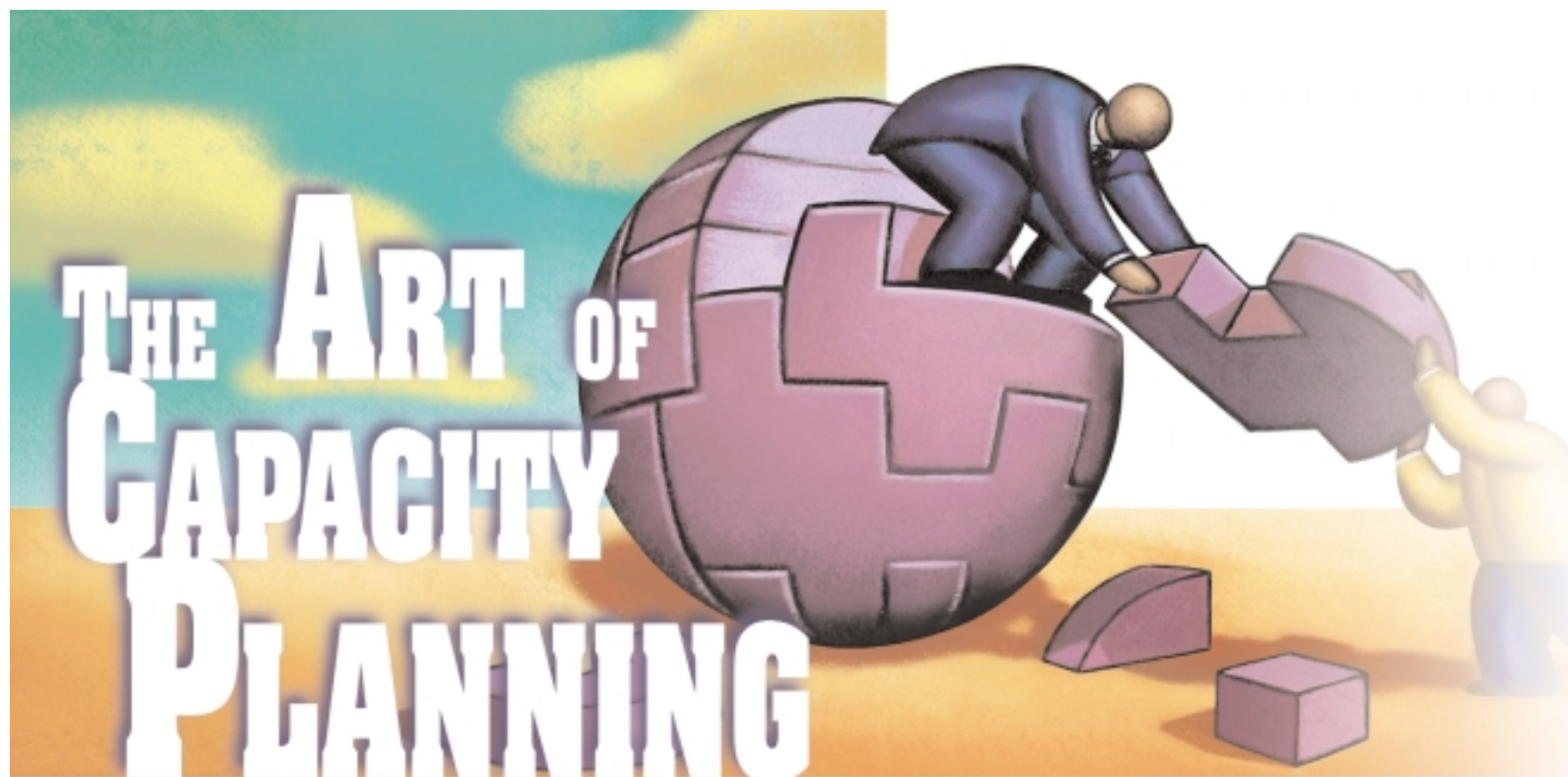
Safest of all will be those that do nothing but touch the in-memory state of the session bean of which they form a part. Next safest will be those that directly access XA resources (for example, by direct JDBC access through a `TxDataSource` to a database). The most dangerous will be those that call external J2EE components. As soon as you invoke external components, you lose control over exactly what resources will be touched, and hence potentially unwittingly send WebLogic into *Groundhog Day* until the `beforeCompletionIterationLimit` is reached. Consequently, I recommend that `beforeCompletion` methods do *not* invoke external components.

So that’s all for this month, more transaction-related chat real soon ...

So that’s all for this month, more transaction-related chat real soon ...

So that’s all for this month, more transaction-related chat real soon ...

Oops, is there a configurable limit for this kind of thing?



GUIDELINES FOR WEBLOGIC SERVER

BY SRIKANT SUBRAMANIAM
& ARUNABH HAZARIKA

WebLogic Server runs on hardware ranging from PCs to high-end mainframes. Therefore, it's essential to carefully choose the level of hardware necessary to provide optimal performance for your WebLogic Server deployment.

This article focuses on the various steps involved in analyzing the constraints on your system components and enumerates several measures that can be taken to ensure that sufficient computing resources are available to support current and future usage levels.

What Is Capacity Planning?

Capacity planning is the process of determining what hardware and software configuration is required to adequately meet application needs. It helps define the number of concurrent users the system can support, the acceptable response times for the system, and the hardware and network infrastructure needed to handle those numbers.

It's important to realize that we cannot generalize this process because each application is different; instead we offer general guidelines that help estimate system capacity.

An iterative process, capacity planning is

achieved by measuring the number of requests the server currently processes and how much demand each request places on the server resources, then using this data to calculate the computing resources (CPU, RAM, disk space, and network bandwidth) necessary to support current and future usage levels.

Why Is Capacity Planning Important?

The first and foremost reason is the user experience factor. Consider a system that can support only a given number of concurrent users while guaranteeing a reasonable response time. As many of us have experienced, when traffic increases on a major Web site that isn't adequately equipped to handle the surge, response time deteriorates significantly. Studies have shown that if a site's response time is more than 10 seconds, users tend to leave. This is generally a bad thing and should be avoided, as it's no secret that a Web site's downtime can result in a significant amount of lost business.

A second reason is that capacity planning helps you decide how to allocate resources for a system in terms of the CPUs, RAM, Internet connection bandwidth, and LAN infrastructure needed to support required performance levels and plan for future growth as well. Once we understand the

limitations of the existing hardware configuration, we can estimate the amount of additional hardware needed to support any increased demands in performance.

Finally, capacity planning is important because it helps answer the question of what hardware and software infrastructure is needed to enable the current system deployment to achieve specified performance objectives.

Factors Affecting Capacity Planning

There are various factors to consider when conducting a capacity-planning exercise. Each of the following factors has a significant impact on system performance (and on system capacity as well). Before embarking on a capacity-planning exercise, it's essential to first tune the WebLogic Server for optimal performance. The WebLogic Performance and Tuning Guide (<http://edocs.bea.com/wls/docs70/perform/index.html>) covers this topic extensively.

PROGRAMMATIC AND WEB-BASED CLIENTS

There are two types of clients that can connect to a WebLogic Server:

1. Web-based clients, such as Web browsers and HTTP proxies, use the HTTP or HTTPS (secure) protocol to communicate with the server. Such a client can be treated as a Web

2. browser client generating HTTP requests.
2. Programmatic clients rely on the T3 or the IIOP protocol and use RMI to connect to the server.

The stateless nature of HTTP requires that the server handle more in terms of overhead. However, the benefits of HTTP clients, such as the availability of browsers and firewall compatibility, are numerous and are usually worth the performance costs.

On the other hand, programmatic clients are generally more efficient than HTTP clients because the T3 protocol does more of the presentation work on the client side. Programmatic clients typically call directly into EJB while Web clients usually go through servlets. The T3 protocol operates using sockets and has a long-standing connection to the server. Consequently, the WebLogic Server can support a larger number of programmatic client threads, which needs to be factored in when calculating the client connectivity bandwidth.

PROTOCOL USED WITH CLIENTS

The protocol used for communication between the WebLogic Server and the clients is another factor in determining the capacity of the deployments. A commonly used protocol for secure transactions is the Secure Sockets Layer (SSL) protocol. SSL is a very computing-intensive technology and the overhead of cryptography can significantly decrease the number of simultaneous connections that a system can support. There is a direct correlation between the capacity of the WebLogic Server and the number of SSL client connections. SSL can significantly reduce the capacity of the server, depending on the strength of encryption used in the SSL connections. Typically, for every SSL connection the server can support, it can handle up to three non-SSL connections.

DATABASE SERVER CAPACITY AND USER STORAGE REQUIREMENTS

Most WebLogic Server deployments rely upon back-end systems such as databases. The more reliance on such back-end systems, the more resources are consumed to meet these requests. The key issues to consider are the size of the data being transferred and the processing capacity of the database server.

Oftentimes installations find that their database server runs out of capacity much sooner than the WebLogic Server does. You must plan for a database server that is sufficiently robust to handle the application. Typically, a good application will require a database three to four times more powerful than the application server hardware. Additionally, it's good practice to place the WebLogic Server and the database on separate machines.

AUTHOR BIO...

Arunabh Hazarika and Srikant Subramaniam are engineers on the Performance Team at BEA's WebLogic Division.

CONTACT...

srikant@bea.com
arunabh@bea.com

The inability to increase the CPU utilization on the WebLogic Server by increasing the number of users is a common problem and generally a sign of bottlenecks in the system. A good place to start investigating would be the database. It's quite possible that the WebLogic Server is spending much of its time waiting for database operations to complete. Increasing the load by adding more users can only aggravate the situation.

An application might also require user storage for operations that don't interact with a database, for instance, a WebLogic-based security realm to store security information for each user. In such cases, you should calculate the size required to store each user's information and multiply this by the total number of expected users to come up with the total user storage requirements.

CONCURRENT SESSIONS AND PROCESSES

One of the main goals of capacity planning is to set quantifiable goals for the deployment infrastructure. This requires determining the maximum number of concurrent sessions the WebLogic Server will be called upon to handle. This affects capacity, as the WebLogic Server has to track session objects (HTTP session objects or stateful session beans) in memory for each session. Use the size of the session data to calculate

“Studies have shown that if response time is more than 10 seconds, users tend to leave a site. This is generally a bad thing and should be avoided”

the amount of RAM needed for each additional user. Next, research the maximum number of clients that will make requests at the same time, and the frequency of each client request. The number of user interactions with WebLogic Server per second represents the total number of interactions per second that a given WebLogic Server deployment should be able to handle.

It's also essential to identify up front frequently accessed components and to allocate adequate resources to them. Typically, for Web deployments users access JSP pages (or servlets), while users in application deployments access EJB.

Additional processes running on the same machine can significantly affect the capacity (and performance) of the WebLogic deployment. The database and Web servers are two popular choices for hosting on a separate machine.

The random and unpredictable nature of user service requests often exacerbates the performance problems of Internet applications. When estimating the peak load, it's therefore advisable to plan

for demand spikes and focus on the worst-case scenario (for instance, the spike in the number of visitors to a site advertised during the Olympics telecast, for instance). Another example would be the spike in traffic experienced by many online retailers during the holiday shopping season. These usage spikes can often result in significant server overload unless properly anticipated.

WEBLOGIC SERVER CONFIGURATION (SINGLE SERVER OR CLUSTERED)

There are many advantages to using WebLogic Server clusters. For instance, they're much more efficient and they offer failover capabilities. The key issues to consider when using a cluster are:

1. Clusters rely on LAN for communication between the nodes. Large clusters performing in-memory replication of session data for EJB or servlet sessions require more bandwidth than smaller clusters. Consider the size of session data, the size of the cluster, and the processing power of the individual machines in computing the LAN bandwidth and network connectivity requirements. The combination of server capacity and network bandwidth determines the total capacity of a given system.
2. If you're using a Web server to forward requests to a WebLogic Server cluster, sometimes the Web server can be the bottleneck. This can happen when using the supplied `HttpClusterServlet` and a proxy server, or one of the supported plug-ins. If the response time doesn't improve after adding servers to the cluster, and the Web server machine shows a CPU usage of over 95%, consider clustering the Web server or running it on more powerful hardware.

WEBLOGIC SERVER CONFIGURATION

It's possible to have many WebLogic Server instances clustered together on a single multi-processor machine. An alternative would be to have a cluster of fewer WebLogic Server instances distributed across many single (or dual) processor machines. There can be advantages in using the second approach:

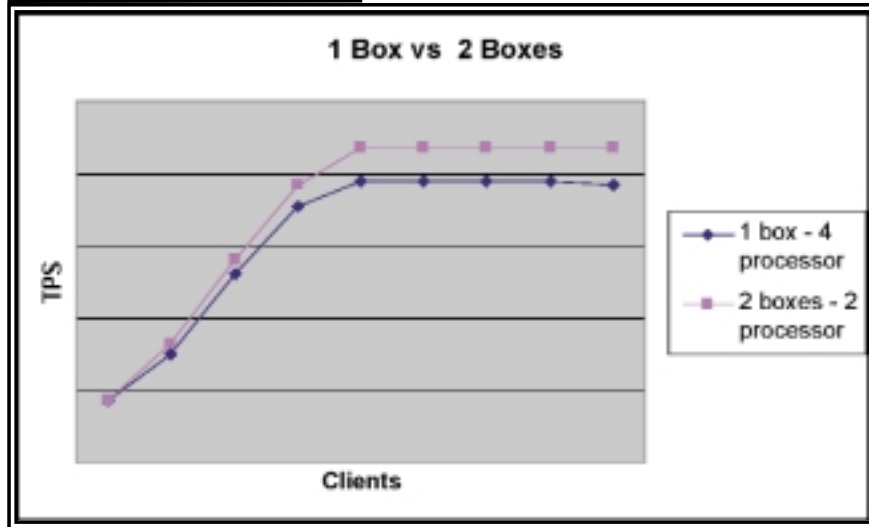
1. Increased protection from failover, since it's unlikely that all the individual machines would fail at the same time.
2. JVM scalability has some practical limitations, and garbage collection is often a bottleneck on systems with many processors. Configuring a cluster of many smaller machines will ensure good JVM scalability. Additionally, the impact of garbage collection on the system's response time can be somewhat mitigated, because garbage collection can be staggered across the different JVMs.

Figure 1 shows the results from an internal benchmark indicating that having multiple

Sitraka JClass ServerViews

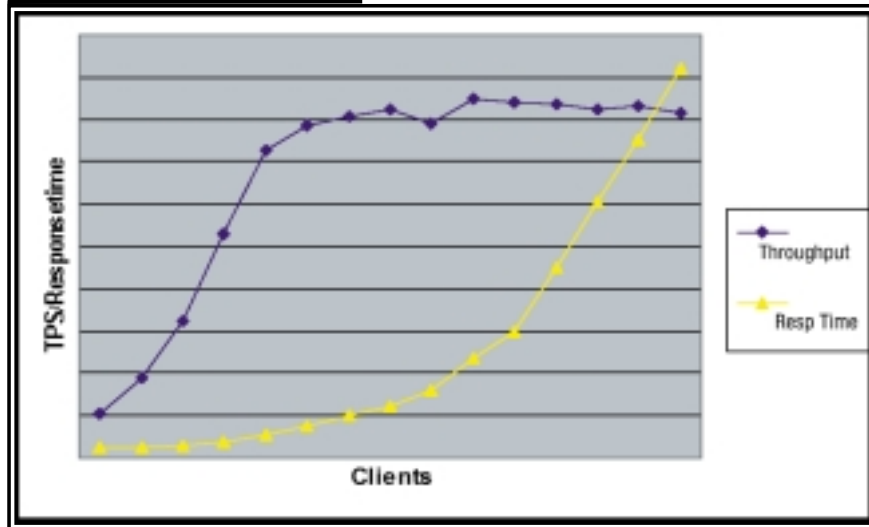
www.sitraka.com/jclass/wldj

FIGURE 1



Comparing performance

FIGURE 2



Effects of increasing the number of clients

smaller boxes offers better performance. We hasten to add that it's quite likely some applications won't conform to this behavior.

APPLICATION DESIGN ISSUES

At the end of the day, WebLogic Server is basically a platform for user applications. Badly designed or unoptimized user applications can drastically slow down the performance of a given configuration. Therefore it's also essential to optimize the application by eliminating or reducing the hot spots and considering the working set/concurrency issues. An end-to-end perspective of the application characteristics is essential in order to diagnose and fix any performance problems. Application optimization and performance tuning WebLogic Server for your specific deployments always go hand-in-hand.

Capacity Planning Guidelines

Once you've developed your application, the next step is to determine the hardware requirements (on the chosen hardware platform). It's essential to choose a transaction scenario that represents the most frequent user flow transactions. In the case of an online bookseller, for instance, a typical transaction mix could be as follows. A user enters the site via the home page, uses the search options to scan through existing inventory, browses through certain titles, selects a title and adds it to the shopping cart, proceeds to checkout, enters credit card information, confirms the order, and finally exits.

There are several tools available to simulate clients (LoadRunner, WebLOAD, etc.). Use the transaction mix you designed in the previous step to generate the client load. Gradually increase the client load by adding more concurrent users. This is an iterative process, and the goal is to achieve as high a CPU utilization as possible. If the CPU utilization doesn't increase (and hasn't yet peaked out) with the addition of more users, stop and look for bottlenecks (in the database or the application). There are several commercially available profilers (IntroScope, OptimizeIt, and JProbe) that can be used to identify these hot spots.

In a finely tuned system, the CPU utilization (at steady state) is usually in the 90-95% range. While throughput won't increase with the addition of more load, response times, on the other hand, will increase as more clients are added. The throughput at this point determines the capacity of the hardware.

Figure 2 shows that increasing the number of clients beyond a certain point doesn't increase the throughput but has a significant effect on the response time. From the graph, decide on an acceptable response time. This, then, indicates the capacity of the hardware needed.

Conclusion

The first step in capacity planning is to set measurable and quantifiable goals for the deployment. As is evident from the iterative process, capacity planning isn't an exact science, hence the need to conservatively estimate your capacity requirements. This is further compounded by the fact that the system load can often be quite unpredictable and can vary randomly (hence the need to focus on the worst-case scenario). Finally, there can be no substitute for load testing. During load testing it's essential to use a transaction scenario that closely resembles the real-world conditions the application deployment will be subjected to.

Acknowledgements

We would like to acknowledge Joginder Minocha for his work on the WebLogic Capacity Planning benchmarks.

Intel
www.intel.com/ad/bea

One of the toughest challenges of any software development architecture is reconciling the object-oriented paradigm with that of the relational database.

Implementing WebLogic QL

ELIMINATE CODE, SIMPLIFY DEVELOPMENT

BY MICHAEL GENDELMAN

If this isn't done properly, the object layer will become too closely tied to the database schema and any change to the database schema will cause a large amount of rework in the object layer. CMP (Container-Managed Persistence), along with CMR (Container-Managed Relationships), EJB QL, and WebLogic QL, provides a mechanism for effectively managing this problem.

For most of us, the power of CMP is not the attempt to seamlessly adapt the code to entirely new database schemes, but a way to manage independent and inevitable evolution of the database and object layer. In an enterprise architecture, it is unlikely that a single database will serve only one client application. As the enterprise requirements evolve, the database scheme will need to change. If we effectively utilize the database abstractions provided by CMP, we can minimize the costs associated with these changes and increase flexibility throughout the enterprise.

CMP requires the programmer to write much less code. Time that was spent managing persistence can now be spent implementing business logic. Although persistence management tools have been around for a while, CMP allows the container to manage the persistence, which opens the door for optimizations not available to the application programmer or other automated persistence mechanisms.

With EJB 1.1, CMP was only suitable for the simplest persistence problems and most advanced systems could not effectively utilize it. Projects built entirely with CMP used an enormous number of fine-grained entity beans that communicated with each other through costly remote calls. Querying data that spanned multiple tables required writing standard SQL statements. We ended up with very slow systems that were still tightly coupled to the database. With EJB 2.0,

new features like local interfaces, CMR, ejbSelect, and EJB QL went a long way toward solving most of these problems. Yet EJB QL is still very new, and unable to entirely fulfill the requirements of most systems. BEA filled this gap by extending EJB QL with WebLogic QL. In this article, I'll show you how to implement ejbSelects and Finder methods utilizing EJB QL and WebLogic QL.

What Are EJB QL and WebLogic QL?

EJB QL is a SQL-like query language used to write queries for ejbSelect and Finder methods. SQL key words like SELECT, FROM, and WHERE are used, but the queries are executed against an object graph of CMP entity beans, not the database. Prior to EJB QL, application server vendors had to provide their own query language. WebLogic had WLQL. EJB QL is new, and will take time to evolve. It is missing key features like subselects, aggregate functions, and group by and order by functionality. As EJB QL evolves, it will surely add most of these features, but for now they're available by using WebLogic QL, BEA's extension to EJB QL.

Setting Up the Example

The example application consists of invoices that can have one or more line items. We will create two entity beans, the Invoice Bean and the Line Item Bean, and one stateless session bean, the Manager Bean. CMR is used to link the Invoice Bean to the Line Item Bean. The entity beans use local interfaces to minimize overhead. This is extremely important when using fine-grained entity beans, which were considered an anti-pattern in EJB 1.1. That is no longer the case if local interfaces are utilized. CMP and CMR are defined in the deployment descriptors. (For information on setting CMP and CMR see <http://edocs.bea.com/wls/docs70/ejb/cmp.html>.)

In constructing the example I tried to keep it simple, but still follow best practices. That's why I included the Manager Bean. Its primary purpose was to control the transactions, and provide a remote interface for my test client. The entity bean method's transaction attributes were all set to "Mandatory", while the session bean method's transaction attributes were all set to "Required". This ensured that each logical unit of work would execute within one transaction, allowing for more efficient execution, but even more importantly it ensured that operations that have to be executed within the same transaction are.

We will use two database tables – the Invoice Table and the LineItem Table – to persist our beans (see Table 1). One final note: all of the examples were developed and tested using WebLogic 7.0.

SIMPLE FINDER METHOD

Let's start by adding a simple finder method to the InvoiceLocalHome, which will return a Collection of all the Invoice Beans. First we'll add the method to the InvoiceLocalHome interface. Notice the return type is Collection. This designates that all of the Invoice Beans matching the criteria of the query will be passed back. If the return type was InvoiceBean, only the first matching InvoiceBean would be returned.

```
Collection findAll() throws FinderException;
```

Listing 1 shows the XML to be inserted into the ejb-jar.xml deployment descriptor. The method name must match the method in the InvoiceLocalHome interface "findAll". Now we'll define the parameters. Even if your query does not require any parameters you must include the <method-params> tag.

Since this is a finder method we would like our return type to be a Collection of Invoice Beans. The key word OBJECT signifies that an entity bean will be returned. In the FROM clause we alias InvoiceBean with "i". This alias not only saves time typing, it can also be used to manage multiple instances of the same bean in one query. When the findAll method of the InvoiceLocalHome interface is called, a Collection containing local interfaces to all of the invoices will be returned. In reality, this would be impractical, but it is a good first example.

FINDER METHOD WITH WEBLOGIC QL

Now let's add a more advanced finder method. This finder will return invoices that do not contain a specified product. For example, show me all the invoices that do not contain a line item with the product "Popcorn." First, we'll add the method to the InvoiceLocalHome interface:

```
Collection findByDidNotOrder(java.lang.String item)
throws FinderException;
```

Always fully qualify the parameter types of the query parameter in code and in the deployment descriptor. This query cannot be expressed in EJB QL, but it can be in WebLogic QL:

```
SELECT OBJECT(i) FROM InvoiceBean AS i WHERE
i.invoiceNumber NOT IN (SELECT i2.invoiceNumber FROM
InvoiceBean AS i2 WHERE i2.lineItems.name = ?1
```

Let's take a closer look at the WebLogic QL query. The first part is the same as the findAll query. The WHERE clause is where it gets interesting. The query utilizes a subselect, which is not available in EJB QL. The subselect utilizes another instance of the Invoice Bean. Notice that the relationship between the line items and the invoices does not have to be defined, as it has

already been defined in the EJB-JAR.XML file utilizing CMR. Now the query is defined in terms of our objects and their relationships, not by the structure of the relational database.

The "name" is an attribute of the Line Item Bean, not a field in the database. The "?1" represents the first parameter being passed into the query, in this case the "item". If two parameters were passed, the second parameter would be referenced by "?2". Be careful when writing EJB QL, as a missing space after the "=" can cause errors during compilation. WebLogic QL cannot be placed directly into the EJB-JAR.XML deployment descriptor, but a placeholder must be created in the EJB-JAR.XML (see Listing 2). Notice that the <ejb-ql> tag is empty.

The real query needs to be placed in the weblogic-cmp-rdbms.xml deployment descriptor, as shown in Listing 3. This will override the query of the same name "findByDidNotOrder" inside the EJB-JAR.XML file.

Simple ejbSelect

ejbSelect methods execute an EJB QL or WebLogic QL query. They can return a single CMP field, an EntityBean or a Collection or Set of either one. BEA has extended the capability to include SQL ResultSet, a powerful feature that allows us to use an ejbSelect in the same way we would use a standard SQL call through JDBC.

There is a special type of ejbSelect statements, in entity. In entity ejbSelects use the bean instance to filter the return data. I prefer to use "non-in" entity ejbSelect methods when implementing home methods, and "in" entity ejbSelect methods when implementing functionality defined in the bean's interface. Although this is not technically required, conceptually it makes a great deal of sense.

EJBSELECT SUPPORTING A HOME METHOD

Let's write a simple ejbSelect that returns a list of all the invoices and their totals. First we should define the method in the InvoiceBean class:

```
public abstract java.sql.ResultSet
ejbSelectAllInvoices()
throws javax.ejb.FinderException;
```

This query will be executed against all of the invoices, as the method name does not end with "inEntity". Notice the return type of ResultSet. This method will return a ResultSet, just as if we had executed a direct JDBC query. Again we have the placeholder in the EJB-JAR.XML deployment descriptor. Now let's take a look at the weblogic-cmp-rdbms.xml deployment descriptor, which is shown in Listing 4. Notice the query returns two values, one a CMP column invoice number, and the other an aggregate of the line items' prices, or

INVOICE TABLE	
InvoiceNumber	Integer
Date	Date
SpecialInstructions	String
LINEITEM TABLE	
LineItemNumber	Integer
InvoiceNumber	Integer
Price	Double
ProductName	String

Database Tables



AUTHOR BIO...

Michael Gendelman is a senior analyst with Viatch, Inc., a New Jersey consulting firm where he develops enterprise systems as a contractor for the U.S. Army. He has been developing distributive systems over the past five years utilizing DCOM, CORBA, and EJB, and is a Java Certified Programmer.

CONTACT...

mgendelman@yahoo.com



the total amount of the invoice. The query also uses the "GROUP BY" clause to perform aggregate functions.

Now let's expose `ejbSelectAllInvoices` through the `InvoiceLocalHome` interface. Remember, this is different from a finder because we are returning select data elements, not bean references. In the `InvoiceLocalHome` we will add the `getInvoiceList` method.

```
Collection getInvoiceList();
```

Next, in Listing 5, we'll add a method to the `InvoiceBean` to handle the `getInvoiceList` call.

In order to handle a custom home method we must add `ejbHome` to the beginning of the method name. Next, we call the `ejbSelectAllInvoices` method. It's not a good idea to return the `ResultSet` to the client.

EJBSELECT WITH WEBLOGIC QL

Let's use `ejbSelect` to total an invoice. First we'll declare the `ejbSelect` method inside the `InvoiceBean` class:

```
public abstract Double.ejbSelectTotalInEntity();
```

The `InEntity` part of the method name tells the container to only consider data associated with this invoice instance. Otherwise this method would total all of the invoices in the database.

Next, declare the query in the `weblogic-rdbms-jar.xml` deployment descriptor (see Listing 6). Remember to always declare the placeholder query in the `ejb-jar.xml` file.

There are a few interesting things about this query. First, it returns an aggregate of a field. Second, the functionality could easily have been attained by iterating through the `listItems` Collection and summing up all of the prices. But that would cause the `listItem` beans to be loaded, and we would still have the overhead of iterating through the Collection. Although we're using local interfaces, there is still some cost involved. Instead, we had the database do the work and return only the answer. This way we reduce the load of the app server, and eliminate unneeded code.

Conclusion

This is just an introduction to EJB QL and WebLogic QL. There's a lot more out there, including WebLogic's ability to execute dynamic queries. I know a lot of developers won't use CMP, and I was one of them with EJB 1.1, but with EJB 2.0 it's definitely worth a second look. It eliminates an extraordinary amount of code and simplifies the entire development process. Once you define your entity beans, and their relationships with CMR, the rest is easy. It's definitely worth trying. The only way to tell if CMP will work for your application is by load testing the application to prove it can handle the performance requirements.

Listing 1

```
<query>
  <query-method>
    <method-name>findAll</method-name>
    <method-params/>
  </query-method>
  <ejb-ql><![CDATA[SELECT OBJECT(i) FROM InvoiceBean AS i]]></ejb-ql>
</query>
```

Listing 2

```
<query>
  <query-method>
    <method-name>findByDidNotOrder</method-name>
    <method-params>
      <method-param>java.lang.String</method-param>
    </method-params>
  </query-method>
  <ejb-ql/>
</query>
```

Listing 3

```
<weblogic-query>
  <query-method>
    <method-name>findByDidNotOrder</method-name>
    <method-params>
      <method-param>java.lang.String</method-param>
    </method-params>0
  </query-method>
  <weblogic-ql>
    <![CDATA[SELECT OBJECT(i) FROM InvoiceBean AS i WHERE i.invoiceNumber NOT IN (SELECT i2 FROM InvoiceBean AS i2 WHERE i2.lineItems.name = ?)]]>
  </weblogic-ql>
</weblogic-query>
```

Listing 4

```
<weblogic-query>
  <query-method>
    <method-name>ejbSelectAllInvoices</method-name>
    <method-params>
      </method-params>
    </query-method>
  <weblogic-ql><![CDATA[ SELECT i.invoiceNumber, SUM(i.lineItems.price) FROM InvoiceBean AS i GROUP BY i.invoiceNumber]]></weblogic-ql>
</weblogic-query>
```

Listing 5

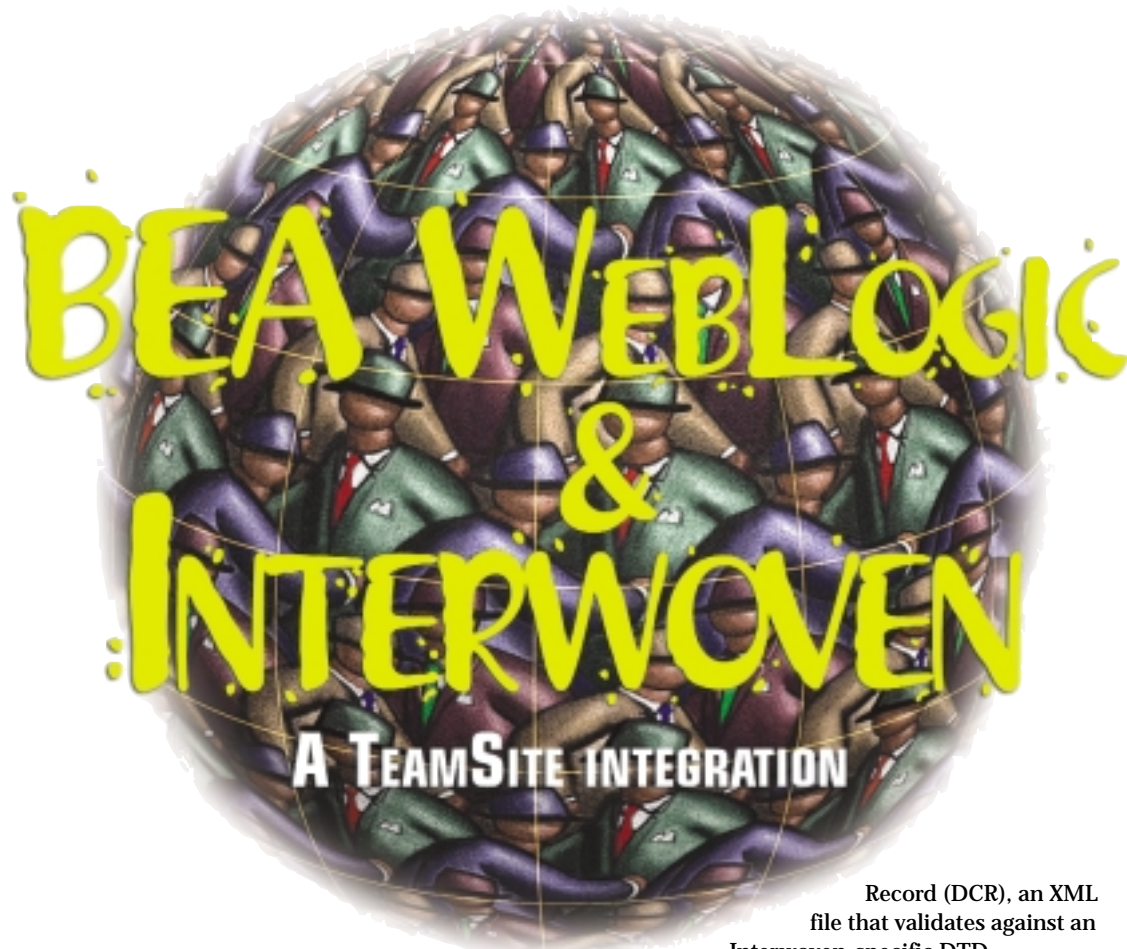
```
public Collection.ejbHomeGetInvoiceList(){
  ArrayList resultList = new ArrayList();
  try {
    java.sql.ResultSet rs =.ejbSelectAllInvoices();
    while (rs.next() ) {
      String row = Integer.toString( rs.getInt(1) ) + "\t" +
        Double.toString( rs.getDouble(2));
      resultList.add( row );
    }
    return resultList;
  } catch (Exception fe){
    fe.printStackTrace();
    return null;
  }
}
```

Listing 6

```
<weblogic-query>
  <query-method>
    <method-name>ejbSelectTotalInEntity</method-name>
    <method-params></method-params>
  </query-method>
  <weblogic-ql><![CDATA[SELECT SUM(i.lineItems.price) FROM InvoiceBean AS i]]></weblogic-ql>
</weblogic-query>
```

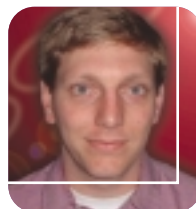
Borland

<http://info.borland.com/new/jb7/5125.html>



Record (DCR), an XML file that validates against an Interwoven-specific DTD.

BY TRAVIS WISSINK



AUTHOR BIO...

Travis Wissink is an independent consultant in the Washington, DC, metro area. He specializes in WebLogic development and content management, as well as Interwoven implementations. Currently, Travis is a lead WebLogic Portal consultant and is integrating BEA Portal with Documentum.

CONTACT...

travis@wissinks.com

Interwoven TeamSite is a best-of-breed Web-based content management system; BEA WebLogic is a certified J2EE application server that (among other features) performs content delivery duties. The two packages perform very different functions but have a relatively simple integration path, allowing them to operate as one. There are many ways to configure these products and various ways to perform this integration – what follows is, based on my experience, the simplest approach with the most functionality. Your mileage may differ.

TeamSite

Interwoven has a (primarily) Perl-based API; three Interwoven products are needed to implement this architecture:

1. **TeamSite Server:** The core package within Interwoven's suite of products, TeamSite Server has been around for more than five years and is reasonably well known within the industry.
2. **TeamSite Templating:** A Web-based interface to capture content, TeamSite Templating stores the captured content in a Data Content

3. **OpenDeploy:** This is a file synchronization application with additional features, such as transactional deployments, so it saves network and processing bandwidth. All OpenDeploy administration can be done through a Web-based (albeit sometimes fickle) deployment interface that can promote any file-based content among registered servers.

WebLogic Server

BEA WebLogic Server is a certified implementation of the J2EE specification. BEA provides a wide range of tightly integrated products – in this architecture we'll use the WebLogic Personalization Server (WLPS) and WebLogic Commerce Server (WLCS). Together they provide APIs and services for some important content management and user management services.

Hardware and Software Architecture

TeamSite must be installed on its own physical server, and TeamSite Templating is then installed on the same server. OpenDeploy must be installed on both the TeamSite Server and the WebLogic Server because it handles the synchronization of the document roots on the two servers. The TeamSite Server uses the OpenDeploy *Base* installation and WebLogic server uses the OpenDeploy

Receiver installation. These two installations should be configured so that the Base install can send files to the Receiver install. The Receiver should be configured to put those files into the BEA WebLogic document manager's designated document root.

The BEA WebLogic Server will be installed on its own physical server, along with the two necessary add-on modules, WLPS and WLCS. Of course, for content storage under this architecture you'll need a relational database (RDBMS). The RDBMS should reside on a different server than the Interwoven and BEA WebLogic servers (see Figure 1).

As you might expect, the delivery solution is hardly plug-and-play. For example, WLCS and WLPS need some WebLogic server resources to be configured. The first resource, a *DataSource* object, connects to a WebLogic *Connection Pool* object. The *Connection Pool* object contains the information necessary to connect to an RDBMS through a JDBC driver. Two other parts of the WLPS Content Manager that need to be configured are a special *Connection Pool* object called the *Document Connection Pool Service* and the *Document Manager Service*. The main purpose of the *Document Connection Pool Service* is to direct the Content Manager to the various Document Manager file system resources. The *Document Manager Service* is a WebLogic object used to cache metadata queries and documents for faster serving. There are many ways to tune the settings of all the various Content Manager objects to improve performance.

Creating and Managing Content

TeamSite Templating allows business users to create content in a browser-based form and produce marked-up content, in our case, static HTML. Some of the required fields in the form should be metadata associated with that content. Ideally, you'll standardize metadata throughout your Web site to mimic the site's taxonomy. Metadata is especially important here for "gluing together" the management and delivery functions taking place within two different software systems. It's metadata that enables BEA to know which static pages or content snippets that come out of Interwoven are to be delivered (and under what context) to end users. Where possible, metadata fields should be drop-down menus or picklists, which allow the WebLogic developers to perform specific queries against specific metadata values. These queries are central to generating standard index pages and automated, dynamic navigation of the live Web site.

On a larger site, developers will code changes to templates and configuration files. The general schematic for the reconciliation and promotion of this content and code is illustrated Figure 2.

When the author saves a file, TeamSite can initiate a workflow that deploys a static snippet of HTML code. TeamSite Templating will generate the HTML code using a transformation file called a *Presentation*

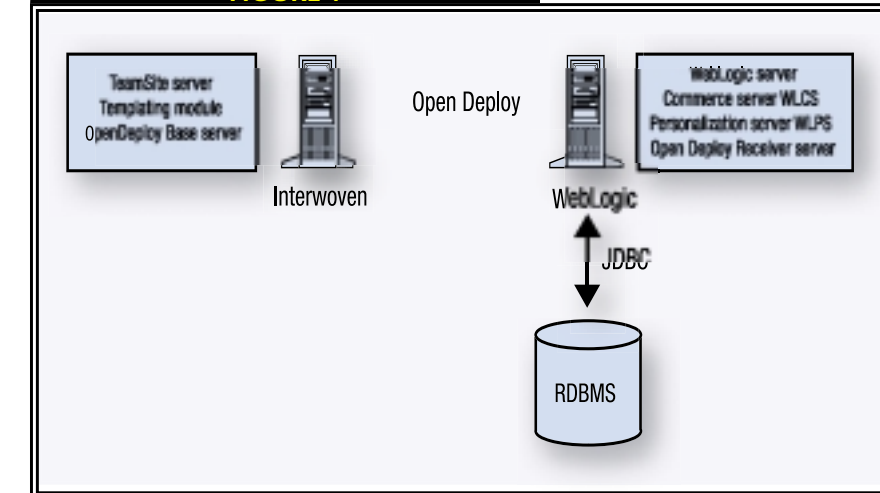
Template. The *Presentation Template* will apply a style to the aggregated content. Note that the *Presentation Template* is not XSL, but an XML file that validates to an Interwoven-specific DTD. For our purposes, the *Presentation Template* should create an HTML snippet that comprises only the content well of a Web page, not the entire page. Other Web page sections, such as the header, footer, and navigational elements, will be retrieved and glued together by the JSP code in the BEA WebLogic Server.

Inside a Deployment

To deploy the HTML snippet, OpenDeploy needs a deployment script to allow the OpenDeploy Base server with TeamSite to send files to the OpenDeploy Receiver on the WebLogic Server. This deployment sends files from the TeamSite Server to the WebLogic Document Manager directory as configured in the Document Manager service. OpenDeploy needs a deploy-and-run script option so that after a completed deployment, it can trigger a WLPS command within WebLogic, the *BulkLoader* method:

```
<deployNRun>
<dnrDeployment location="source" when="after"
state="success">
    <script cmd="/path/to-the/script/that-starts-
the bulkloader.sh" as="weblogic" where="/tmp"
async="yes" />
</dnrDeployment >
</deployNRun>
```

FIGURE 1



General schematic

OpenDeploy deployment is an XML file that validates to an Interwoven-specific DTD. The *deployNRun* option of the deployment file describes how to execute a file inside an OpenDeploy transfer. Inside the *deployNRun* tag is the *dnrDeployment* child, which has three descriptive attributes:

1. **Location:** On which server to execute the command, the Base install, or the Receiver install
2. **When:** Before or after the deployment
3. **State:** Status of deployment – success or failure

Note: State is important because if the deployment fails, you'll want to roll all the changes back and start over.

Inside the `dnrDeployment`, use the `script` tag and its attributes to tell OpenDeploy what to execute and how:

- **Cmd:** The full path to the file that needs to be executed
- **as:** The user to execute the cmd, in this case "weblogic"
- **where:** The directory in which OpenDeploy should execute the command
- **async:** Whether OpenDeploy should execute the command and wait or continue to the next option

retrieve the URLs associated with the relevant deployed content.

Below is the BulkLoader command in WebLogic, along with several important options. Since it is a Java class file, it's started with the "java" command.

```
java
com.beasys.commerce.axiom.document.loader.BulkLoader
[-/+verbose] [-/+recurse] [-/+delete] [-/+cleanup][-
schemaName <name>]
[-properties <name>] -conPool <name> [-match <pattern>]
[-ignore <pattern>] [-d <dir>][--][files... directo-
ries...]
```

You can "verbose" the command so you can watch what it is performing; use this during debugging only. The "recurse" option allows you to drill down and process any subdirectories. Two important options are "delete" and "cleanup" – the only way to remove deleted files from the database. Your script to execute the BulkLoader should look something like Listing 1.

Inside the Delivery

After the BulkLoader has processed the metadata we can present it to our Web site's end users. We'll do this using JavaServer Pages (JSP) and the WLPS content management tags. From the WebLogic front end, the JSPs would use the following code to access the WLPS Content Manager. These examples use the "cm" and "es" tags from the WLPS custom tag library.

```
String contentQuery = " keyword = '4q2000commentary' &&
contentlevel like '*A*' && ((expirationDate >= now) ||
expirationDate='0')";

<cm:select contentHome="com.beasys.commerce.axiom.docu-
ment.Document" query="<%=contentQuery%" sortby = "pri-
ority ASC, postedDate DESC" id="contentList" />
```

The above code queries the metadata to obtain a specific set of records, then sorts the results, ascending by priority and descending by the age of the post.

The query resembles standard SQL and the select tag is well documented. Basically, you tell the select tag your query, which we've abstracted a bit into a separate String object. Also, tell the select tag to sort your result set in ascending and/or descending order. Tell the select tag the content "Home" of the ContentManager EJB; this is the standard for our implementation, and the EJB conveniently comes deployed on installation of WLPS. Because "priority" and "posteddate" represent metadata in the database associated with specific pages or content snippets, the select tag will automatically retrieve that content also. The last attribute to configure for the select tag is the "id," which will be used later in the code to reference the result set.

Now that we have a result set we can display

the results as a dynamic index page of hyperlinks to static content as shown in Listing 2.

We took the result set that was created by our metadata-driven query and parsed out the page titles and links and published them as a list of hyperlinks on an index page. Pretty easy for a Java programmer. It also assumed that metatagging requirements were rigidly enforced in TeamSite Templating. If the metatags aren't there, the content won't be found, at least not through this type of query.

Some Advice

What follows is a list of five things to watch for in any BEA/Interwoven integration.

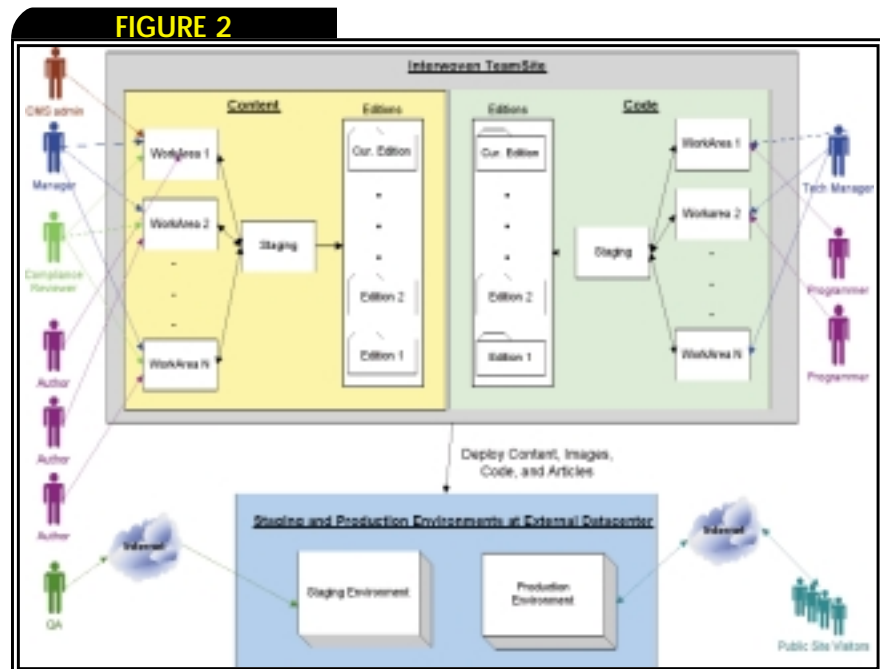
1. **Keep data capture templates as separate as possible from content presentation** – that is, keep content style away from the authoring process. This is essential but sometimes difficult at a time when editors want specific WYSIWYG HTML authoring tools. If you allow the editors to sully the content entry system with presentation logic via HTML tags, then when you want to change the presentation, your content will have to change also. That will take many hours of content editing. Put that time and effort into developing granular data-capture templates instead, and take care of end-user presentation within your JSPs on the delivery side.
2. **If your delivery servers are clustered, strategize about where to put things like images.** Are your BEA WebLogic Servers going to share a file system so that all the content can reside on the same drive? If not, content replication software may be needed. You could use OpenDeploy for this, although at approximately \$8,000 a server this can become rather expensive. I recommend rsync (free from samba.org) on Solaris, and Opalis RendezVous (a more robust solution for a few thousand dollars) on Windows. Like OpenDeploy, these products don't guarantee content replication across your enterprise or clustered Web and application servers. File replication software guarantees replication.

3. **A lot of requirements-gathering and analysis are needed to carefully negotiate the metatagging scheme.** Metadata could make or break your long-term content goals. If you don't gather enough, your application could appear useless within months, because you can't create a new query. If you attempt to gather too much metadata, your content-ingestion methods may be too cumbersome, and your authors will rebel in the face of what they perceive as meaningless entry fields. Find a delicate balance of rich metadata to satisfy both short- and long-term objectives.

4. **All the metadata from the WCM will be deployed and loaded into a database,** more specifically into one table. Many new rows in the databases will start to degrade your Web site performance. To increase database speed you could set up mirrored databases and use the BEA multi-

connection pool option. This allows the databases to contain the same data and allows the BEA WebLogic cluster to query from a clustered database.

5. **If you have a site that will have many pages (about 5,000) of content, the BulkLoad process could take a long time to finish.** With 5–10 metadata items for each page, 5,000 pages requires approximately 2.5 hours to synchronize to the database. One option is to shift the OpenDeploy deployments to a special type of deployment called a *file-based deployment*, which is useful if you have many files on the file system. With this approach, OpenDeploy performs the deployment to include only files that you list in a separate configuration file. That file list can be included with the deployment. After deployment, OpenDeploy can initiate a `deployNRun` script that takes the file list config file and initiates the BulkLoader against the recently deployed files. This increases the accuracy of your delivery engine and conserves processing power. The downside to this approach is that the file list must be created, edited, maintained, and deleted by you, though TeamSite workflow can help here. Another disadvantage is that nothing is deleted from the delivery servers; deleting files isn't a native capability of the file-based deployment. The level of complexity greatly increases when you move to this type of deployment, but the performance rewards may justify the added development work.



Code/content life cycle interaction

The BulkLoader initially looks through a directory that contains files of HTML content and scans for its associated metadata. The BulkLoader looks into the HTML files to retrieve every metatag's name-value pair and loads it into the WLCS database, making a new entry in a database table for every metatag. The relevant database table, `document_metadata`, has four columns, but three in particular are important to us: ID, Name, and Value. These contain the URI (from the document root), name (from a metatag), and value (from the same metatag), respectively.

The BulkLoader sends all the metacontent in the HTML pages into a database so other WLCS libraries can use this descriptive content to

Listing 1

```
. /opt/bea/wlportal4.0/bin/unix/set-environment.bat

java -classpath %CLASSPATH% -Dcommerce.properties=%WL_COMMERCE_HOME%/weblogiccommerce.properties
com.bea.pl3n.content.document.ref.loader.BulkLoader -ignore "doc-schemas" -properties
%WL_COMMERCE_HOME%/bin/unix/loaddocs.properties -conPool commercePool -d
%WL_COMMERCE_HOME%/dmsBase

java -classpath %CLASSPATH% -Dcommerce.properties=%WL_COMMERCE_HOME%/weblogiccommerce.properties
com.bea.pl3n.content.document.ref.loader.BulkLoader -ignore "doc-schemas" -properties
%WL_COMMERCE_HOME%/bin/unix/loaddocs.properties -conPool commercePool -d
%WL_COMMERCE_HOME%/dmsBase -cleanup -delete
```

Listing 2

```
<es:foreachinarray id="content" array="contentList"
type="com.beasys.commerce.axiom.content.Content" counterId="i">
<%
myStr=content.getIdentifier();
%>
<a href="<% =new String("manager_commentary_display.jsp?id="+myStr) %%"><cm:printproperty
id="content" name="title" encode="html" /></a><BR>
</es:foreachinarray>
```




We're nearly done covering the topics for the test. This month I'll discuss JDBC, transactions, and clustering. Be sure to give careful attention to these topics, as they can slip by during studying. As in my previous articles, I've included a sample test to help you study for the real test. Let's get started with JDBC.

Making the Grade

JDBC, TRANSACTIONS, AND CLUSTERING ON THE WLS 6.0 CERTIFICATION TEST

BY DAVE COOKE



AUTHOR BIO

David Cooke is an experienced software developer currently working for Ness Technologies, Inc. (www.ness-usa.com), a consulting firm located in Dulles, VA. In his current position, he utilizes Java and BEA WebLogic Server 6.0 to build J2EE-compliant e-commerce systems for a variety of clients. Dave maintains Microsoft, Java, and BEA developer certifications.

CONTACT...

dave.cooke@ness-usa.com

JDBC

Now, I'm sure most *BEA WebLogic Developer's Journal* readers have a good amount of JDBC experience, so I'll only touch on the WLS-specific aspects of JDBC you'll need to know to pass the test.

First, a word about transaction isolation levels. The isolation level of a connection determines how database transactions are protected from one another. Transaction isolation is a database concept enforced at the database software level. WebLogic supports a series of common transaction isolation levels. These are passed to the proper resource manager, where the database software will handle enforcing them. Note that not all databases support transaction isolation.

A connection pool is a group of JDBC connections to a database. Using connection pools provides a significant performance advantage to your application. Connection pools can be configured either statically at application startup or dynamically within a Java application. You can obtain a pool object from "weblogic.jdbc.JdbcServices," using a JNDI lookup. Be sure to review Table 1, which lists some connection pool methods of note.

WebLogic has a way to access multiple connection pool objects, a feature called MultiPools. A

MultiPool is a useful mechanism to provide load balancing and high-availability services. To gain the load-balancing and high-availability functionality of MultiPools, your databases must support real-time replication. MultiPools are configured statically, using the Administrative Console. In WLS 6.0, there is no way to dynamically configure MultiPools.

MultiPools are load balanced by using the Round-Robin algorithm, which chooses in order from the list of connection pools. MultiPools provide high-availability services if the database connectivity has been lost. A new connection is attempted with the next pool in the connection pool list. Please note that the high-availability services aren't a fail-safe mechanism - MultiPools will only fail over for a new connection. Any existing connection that fails, for any reason, will return an exception to the application - the MultiPool won't attempt to reconnect on behalf of an existing client. Another important note: MultiPools only fail over when database connectivity is lost, or when the database is dead. Capacity or other exceptions will not cause failover because these could be set on a per-connection basis.

Transactions

In order to coordinate distributed transactions, WLS 6.0 supports Extended Architecture (XA), which defines the protocol used for different resource managers to communicate with one another. By using XA, the server implements the two-phase commit protocol to support distributed transactions. Transactions may span EJB accesses, JDBC connections, and JMS connections. In order to use this feature, the resource manager must be XA-compliant.

Let me point out that the two-phase commit protocol guarantees data integrity by ensuring that a transaction is either updated by all of the databases in a distributed transaction, or by none of them. The first phase of a two-phase commit is the prepare phase, in which all the resources vote on whether or not to commit the transaction. If all the resources participating in the transaction vote to commit, the updates are made to all the databases during the second phase. If any one of the databases votes to roll back, then the transaction is rolled back for all of the participating databases.

In handling transactions for EJBs, WLS 6.0 offers two different types of transaction schemes.

Mongoose Technology

www.portalstudio.com

You may choose to define your EJBs as using either container-managed or bean-managed transactions. Using container-managed transactions indicates the container will manage the transactional demarcation of the bean. In contrast, bean-managed transactions leave it up to the bean developer to indicate the transactional demarcation.

TABLE 1

METHOD NAME	DESCRIPTION
ShutdownHard	Shut down and kill all connections immediately.
ShutdownSoft	Shut down pool and wait for connections to be returned before closing.
Reset	Close and reopen all connections.
Shrink	Drop some connections when peak usage period has ended.
DisableFreezingUsers	Disable pool. Clients with current connections are suspended.
DisableDroppingUsers	Disable pool. Destroy client connections to pool.

Connection pool methods

TABLE 2

ATTRIBUTE NAME	DESCRIPTION
Never	The EJB isn't part of a transaction. An exception is thrown in the client if the client calls the EJB in the context of a transaction.
NotSupported	The EJB isn't part of a transaction. Any client transactions will be suspended while the bean is executing.
Supports	The EJB will operate within a transaction, if the client has started one.
Required	The EJB will operate within a transaction and the container will start one if the client has not started one.
RequiresNew	The EJB will start a new transaction, which will be started by the container.
Mandatory	The EJB must operate in a transaction, which the client must have started.

Transactional attributes

TABLE 3

METHOD	DESCRIPTION
Begin	Marks the beginning of a new transaction.
Commit	Completes the current transaction.
Rollback	Rolls back the current transaction immediately.
setRollbackOnly	Modifies the transaction so the only possible outcome is that the transaction is rolled back.

UserTransaction methods

TABLE 4

ALGORITHM	ORDER	DISADVANTAGES
Round Robin	In order	Since some servers may perform more slowly or get larger requests than others, "convoying" may occur, meaning one server may become a bottleneck.
Weight-based	By weight	Small performance hit.
Random	Random	Small performance hit and no guarantee that the load will be evenly balanced.
Parameter-based	By CallRouter interface	Performance hit to determine server in order to determine which object gets called next. (depends on the complexity of the call router).

Load-balancing algorithms

For container-managed transactions, the transaction attribute is supplied to the container by the ejb-jar.xml file. The "trans-attribute" deployment descriptor defines how the container will deal with transactions. The list of transactional attributes is shown in Table 2.

Container-managed transactions may be specified for session beans and message-driven beans, but *must* be used for entity beans. Because a message-driven bean has no client, message-driven beans that use container-managed transactions must be deployed using either the Required or NotSupported transaction attribute.

In message-driven beans, a new transaction is started for every message. The transaction is committed automatically at the end of the onMessage method, unless setRollbackOnly() is called.

For bean-managed transactions, however, since the bean is in charge of managing its own transactions, it must be able to indicate when a transaction has started or ended. The container provides the UserTransaction object to allow the bean to demarcate the transaction boundaries.

The UserTransaction object is obtained through the bean context's getUserTransaction() method. In addition, the container must expose the UserTransaction through JNDI and can be accessed using the java:comp/UserTransaction name. Table 3 lists the UserTransaction methods you should know for the test.

As stated before, a client may start a transaction by obtaining the UserTransaction object through JNDI. After the transaction is started, any EJB methods or JDBC/JMS connections will be included in the transaction, provided the resource managers are all XA-compliant.

Clustering

Believe it or not, WLS 6.0 clustering functionality gets a good amount of coverage on the test. For review, a cluster is a group of WebLogic servers acting as a single server for the purposes of scalability and high availability. Although the cluster appears to be a single server from the perspective of the application, it's necessary for the developer and administrator to understand the behavior of WLS clusters. In order to provide adequate scalability, WLS supports four algorithms for load balancing objects in a cluster. The algorithms determine how the server will achieve scalability by distributing the load across the cluster. Each algorithm has its advantages and disadvantages. The algorithms are listed in Table 4. Be sure to note that the default algorithm for object clustering is Round Robin.

High Availability

As for providing high availability, WLS 6.0 provides some services to ensure that your EJBs are available. WLS employs something known as a replica-aware stub. When an EJB is called through the replica-aware stub, if a failure occurs, the stub intercepts the exception and retries the call on another replica.

Given that, WLS 6.0 handles failover scenarios differently for differing types of services. Before we get into the details, you should know the term *idempotent*. If a bean is marked as idempotent, it means that it can be restarted without any side effects – the result will be the same. In this case, the server will actually retry a method that fails during the request, because it can just retry the method call on another server and assume the result will be the same.

Now, in the case of nonidempotent objects, WLS 6.0 will still attempt to keep your objects available by failing over if the method hasn't actually been called (to be specific, if the ConnectException or MarshalException is thrown).

For stateless session beans, if a bean isn't available on one server, WLS will try to use the same bean on all other servers in the

cluster. As for a stateless session bean, if the bean isn't available, WLS will try the one backup server. Calls to stateful session beans are "pinned" to the server in the cluster, since state is being maintained. The stateful session bean is replicated to the backup server after a transaction is committed or at the end of a method call.

The good news is that there isn't much left to discuss about the test. Next month I'll cover the one major topic left: administration. This will include discussions on deployment, security, and server usage. In addition, I'll touch on a few things that don't really fit into any category I've discussed before. I'll also include a slightly larger sample test that will encompass all of the topics we've covered so far.

That's it for this month! Have fun taking the sample test below. Good luck!

Believe it or not, WLS 6.0 clustering functionality gets a good amount of coverage on the test

Sample Test

- Which class would be the most efficient to use to perform multiple SQL INSERT statements into a database?
 - Statement
 - CompiledStatement
 - CallableStatement
 - PreparedStatement
- Which statement about transaction isolation is true?
 - WLS 6.0 implements transaction isolation support.
 - The database software is responsible for implementing transaction isolation.
 - Changing the transaction isolation levels will segment the database.
 - None of the above.
- Which method of the Pool interface allows you to reduce the number of database connections used after a peak usage period?
 - ShutdownSoft

- Shrink
- Reset
- DisableDroppingUsers

- At which point does a MultiPool implement the failover algorithm?
 - When database capacity is reached
 - When a database-related exception is thrown
 - When the database is dead
 - All of the above

- Which statement about MultiPools is true?
 - MultiPools may be dynamically configured.
 - MultiPools provide real-time replication services to any database system.
 - MultiPools must be configured using the same JDBC driver for all connection pools.
 - None of the above.

- What interface do you use when demarcating a bean-managed transaction?
 - UserTransaction
 - Transaction
 - TxHelper
 - StartTransaction

- How can you prevent a transaction from committing when using container-managed transaction demarcation?
 - Use the rollback() method.
 - Use the getRollbackReason() method.
 - Use the setRollbackOnly() method.
 - You can't prevent a transaction from committing when using container-managed transactions.

- If a transaction is rolled back, what interface must you implement so a session bean receives notification of the rollback?
 - SessionSynchronization
 - UserTransaction
 - Transaction
 - All of the above

- What load-balancing algorithm has the problem of convoying?
 - Round Robin
 - Random
 - Weighted Round Robin
 - Parameter-Based

- Which load-balancing algorithm uses the CallRouter interface?
 - Round Robin
 - Random
 - Weighted Round Robin
 - Parameter-Based

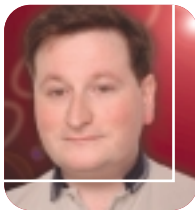
1. d, 2. b, 3. b, 4. c, 5. d, 6. a, 7. c, 8. a, 9. a, 10. d

continued from page 6

These tools discard much of the detailed information required for root-cause diagnosis because they sample method calls and aggregate results in order to reduce overhead. While such monitoring highlights application bottlenecks in production, it does not go the proverbial "last mile." Developers still invest significant time and effort to reproduce the bottleneck in the test lab to diagnose and fix the underlying problem.

Detailed diagnostic tools are used in test environments to capture the details of each significant method call. Such tools fall into two categories: tools that aggregate and summarize the captured information online for presentation, and tools that save detailed information to disk for offline analysis. While both types of tools help find bottlenecks and provide a breakdown of aggregate transaction time, the second category also supports correlation of individual transactions through different software layers and examination of the execution profiles of individual threads. This analysis of individual transactions and threads is critical to diagnosing complex performance problems.

In summary, diagnosing tough performance problems in J2EE applications requires a reliable way of reproducing the problem and a low-overhead technique to examine the internal details of the application under load. The capture and playback of real workload from production provides the best option for reproducing difficult performance problems. Performance diagnostic tools based on byte code instrumentation can capture and correlate individual method calls and thread invocations with low overhead and provide the best option for drill-down analysis.



BY
TAD STEPHENS
& ERIC GUDGION

AUTHOR BIOS...

Tad Stephens is a system engineer based in Atlanta, GA, for BEA Systems. Tad came to BEA from WebLogic and has more than 10 years of distributed computing experience covering a broad range of technologies, including J2EE, Tuxedo, CORBA, DCE, and the Encina transaction system.

Eric Gudgion is a principal systems engineer with the Technical Solutions Group. His background on mainframe systems comes from years of work within the field as a system programmer and system engineer for various Mainframe vendors.

CONTACT...

tad@bea.com
eric.gudgion@bea.com

So, you're going to deploy WebLogic Server on the mainframe. Pretty scary, huh? There are all those "glass house" terms: sysgens, operating systems with a "z," parallel sysplex, Workload Manager, and on and on. Without a little education, the mainframe world can be as foreign to the Java developer and architect as the distributed J2EE world is to a COBOL programmer on the host.

But these two environments aren't as different as you might believe. This article covers how to install, configure, and deploy WebLogic and WebLogic-based J2EE applications on the mainframe, specifically z/OS and z/VM deployments.

The first article in this series (*WLDJ*, Vol. 1, issue 7) described why customers have decided to combine the benefits of the industry's most stable, reliable, and scalable application server with the operational advantages of the mainframe.

- Rewriting existing mainframe applications in Java allows higher programmer productivity and flexibility and eliminates dependence on a single vendor.
- Consolidating UNIX and Windows servers with z/Linux lowers total cost of ownership.
- Deploying new applications on existing mainframes (z/Linux and z/OS) offers better resource utilization.
- Leveraging business contingency benefits through the mainframe qualities of service

and operational properties ensures that J2EE applications are always available.

- Extending existing systems and applications when rewriting isn't practical or feasible lowers cost.

This article details the installation, including the steps required; what is needed on the mainframe; and how it's different from WebLogic installations on other platforms.

Mainframe Configuration and Installation

There are three key ways to install and configure WebLogic for deployment on the mainframe:

1. Install and run in z/OS
2. Install and run in z/VM under Linux
3. Create a Logical Partition (LPAR) and install WebLogic directly to Linux on the mainframe

One advantage of WebLogic on the mainframe is that, regardless of your deployment model, the

same J2EE application developed on Windows or another UNIX platform will run without any changes on the mainframe. You're free to use powerful developer productivity aids without consideration of the required deployment platform. In addition, WebLogic has the unique ability to cluster a J2EE application across various hardware platforms, grouping the mainframe and UNIX or Windows NT/2000 servers in a single cluster.

WebLogic Under z/OS

WebLogic can be installed on z/OS, IBM's flagship operating system used on the zSeries line of hardware. When deployed on z/OS, WebLogic executes as a UNIX System Services (USS) task. USS can be thought of as a mode of execution on z/OS – all the POSIX APIs are implemented directly into z/OS, and provide a standard API set for a multithreaded environment. This execution model enables programs written using UNIX libraries to execute unchanged (or with minor changes) on the mainframe.

Naturally, WebLogic on this platform uses a Java Virtual Machine (JVM); on the mainframe we use IBM's JVM. The current version of IBM's JVM is based on JDK 1.3.1. and can be downloaded free of charge from www.ibm.com/java.

Once installation is complete, use the "java -version" command to verify successful installation.

WebLogic can now be installed – download a copy of WebLogic Server from the BEA Web site (www.bea.com/download) onto a local file system. Once the download is complete you're ready to start the installation process. This is where we need to note a few procedural differences between installing WebLogic on a Windows or UNIX platform.

- The installation process is run from a Telnet session to the mainframe, not via an OMVS shell.
- Only the shell scripts are in EBCDIC; all other files are ASCII. It's always a best practice to review the readme.txt file for the very latest installation steps; the documentation will walk you through the remaining steps.

At this point you have a WebLogic Server installed on the mainframe under USS. To someone familiar with WebLogic, this looks like a WebLogic installation on any other platform: the sample applications, graphical aids and deployment tools, steps to deploy applications, and administration console are the same.

The snippet of the WebLogic console shown in Figure 1 should look familiar. It's actually WebLogic on the mainframe. The only way to see this is by looking at the version information on the console for the platform information.

WebLogic can now be started, shut down, and administered as on any other WebLogic platform.

How can WebLogic make use of the mainframe environment attributes? You could follow the normal method of starting WebLogic, i.e., running the startweblogic script, but that requires the administrator to be logged on to the mainframe operating system via a Telnet session. A different and decidedly more mainframe approach is to create some JCL (Job Control Language) programs, which are like scripts for the mainframe operating system, to start and stop WebLogic. Using a simple JCL procedure, we can control WebLogic.

Listing 1 is an example of some JCL that could be used, although the mainframe system programmer would customize this for a specific site's use.

When using JCL to control the execution of WebLogic, we don't have to use a Telnet session to start WebLogic – this allows the operations staff to automate startup and shutdown, making WebLogic's operation on the mainframe more natural for the main-

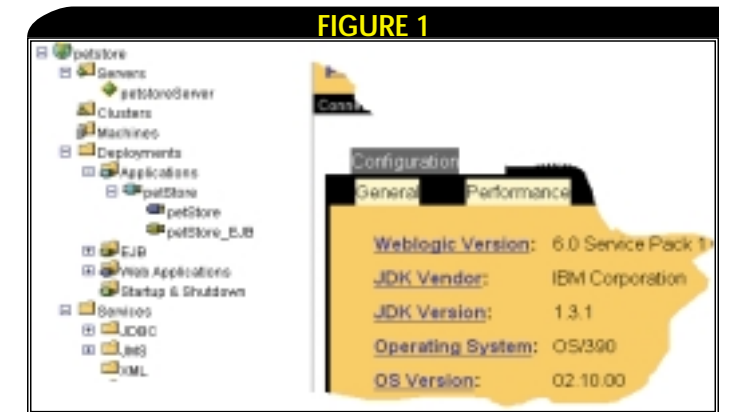
frame staff because they can use the tools they're accustomed to.

WebLogic and the WorkLoad Manager

Workload management is a very powerful function of z/OS. The WorkLoad Manager enables machine resources to be utilized efficiently to achieve the best performance for a given workload. For example, it might define that a workload must have its transactions completed within a given number of seconds – a *Response Goal*.

Like any workload on the mainframe, WebLogic can participate in workload management. For example, the administrator would define a Response Goal for WebLogic where a percent of transactions must be completed within a time period (specified in seconds). z/OS then prioritizes all the work on the machine to achieve this goal. As you might imagine, this is a very powerful, results-orientated scheduling mechanism.

In order to implement a Response Goal for WebLogic, the mainframe system programmer defines a group with the required goal. The goal is then defined to the WebLogic task, which associates the goal with the corresponding task to be completed. The goal and corresponding task are defined using the standard WorkLoad Manager screens in ISPF. Figure 2 shows a page cut from a WorkLoad Manager configuration session.



WebLogic on the mainframe

WebLogic Under z/VM and Linux

Also consider the z/VM deployment model, which has many benefits as well. The z/VM operating system enables multiple virtual machines to be created that represent a physical mainframe, thus allowing a high level of resource-sharing and reuse to be achieved. For example, using multiple z/VM virtual machines as z/VM Guests may allow for multiple UNIX or NT deployments to be consolidated on to a single mainframe server (see Figure 3).

When using z/VM, install Linux into the z/VM Guest machine. WebLogic currently supports SuSE, but other Linux brands will be certified over time. Once Linux is installed on the virtual machine, WebLogic can be installed. Start the installation by accessing Linux running in the z/VM Guest from a Telnet session and run the install command shown below; you'll then be prompted for the remaining installation steps.

```
"java -classpath weblogic600sp2_generic.zip install -i console"
```

The only differences in the steps for installing on z/VM are:

- WebLogic on z/VM uses the IBM JVM (as noted above).
- A JAAS file is created for the associated Linux user that will start WebLogic, such as /home/weblogic.

Because z/VM has a high degree of resource-sharing, the configuration of the z/VM Guest machine that executes both Linux and WebLogic is important. The memory allocated to WebLogic is defined by the parameter *Guest size*. This sets the amount of memory the virtual machine will have. The actual amount of memory required will depend greatly on your application; however, an average value is 512MB. It's important to note that this is virtual memory.

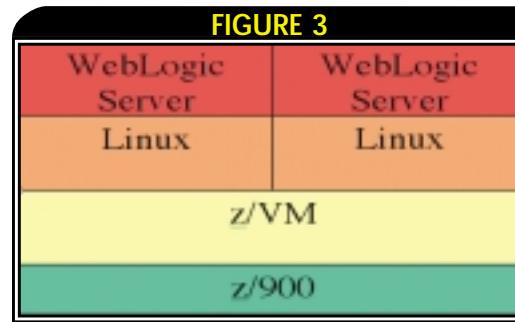
Another important setting is the execution class. This value will differ from site to site, but WebLogic should be defined as a high-priority Guest machine to z/VM, as the transaction-response-end users experience is a factor of the execution class.

z/VM provides a number of key performance optimizations. For example, all network functions can be implemented as if they existed in a single virtual machine. This concept is called virtualization. z/VM provides an option, Guest LAN Support, where a LAN segment is defined in memory and all the Guest machines connect to it via a high-speed, in-memory network. WebLogic can use this option, and the z/VM virtualization will hide the actual implementation.

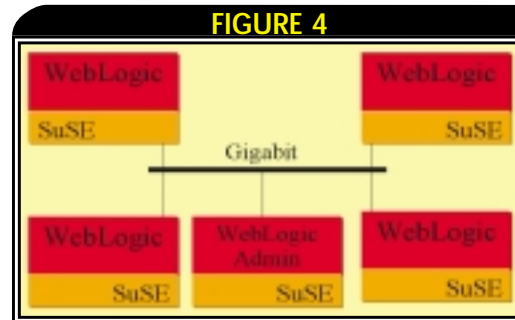
Other network options are available as well, such as the virtual channel-to-channel adapter (VCTCA) to establish a connection between two Guest machines via a virtual point-to-point network.

WebLogic Running Linux Within an LPAR

The last deployment model is the installation of Linux directly on the mainframe hardware. This is known as a Logical Partition (LPAR) installation. This type of installation reserves a portion of the mainframe hardware for a logically separate operating system environment. Although hardware reuse is limited and may not be allocated in the most efficient manner, this is an easy way to try Linux on the mainframe, as z/VM and



Multiple deployments on a single server



With Linux on the mainframe

z/OS at the respective levels are not required (see Figure 4).

Once Linux is installed in the LPAR, the installation of WebLogic follows the same steps as a z/VM Guest machine installation.

Because the WebLogic platform has so many third-party vendors building on or interfacing with WebLogic Server, a whole new breed of applications is becoming available. A number of traditional mainframe vendors are committed to supporting WebLogic deployments on the mainframe. These new offerings will allow WebLogic to be managed and administered using the tools and utilities with which mainframe staff are already familiar.

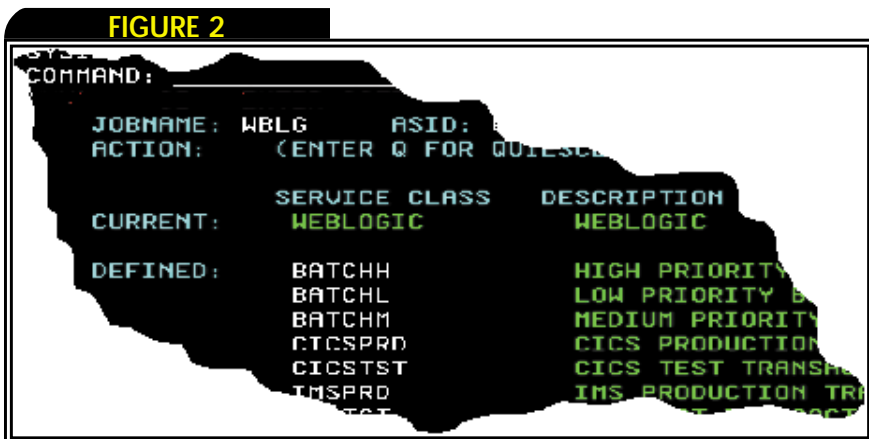
Which Operating System?

The question of which deployment model to use for the mainframe can be answered by identifying what hardware and operating system are available. Although an LPAR provides an easy way to test WebLogic, the ease of deploying WebLogic in z/OS or as a z/VM Guest enables customers to select the environment they're comfortable with.

Because WebLogic and the application are configured and deployed in the same way on all platforms, even if the production deployment is on different hardware or a combination of operating systems/hardware, picking the operating system is strictly a matter of selecting the one that provides the services (like WLM) you need.

What's Next?

With the execution and operational questions answered, we need to build an application that



WorkLoad Manager configuration session

Once you're in it...
reprint it...

- Wireless Business & Technology
- Java Developer's Journal
- XML Journal
- ColdFusion Developer's Journal
- PowerBuilder Developer's Journal

Contact Carrie Gebert
201 802-3026
carrieg@sys-con.com

RePrints

WLDJ ADVERTISER INDEX			
ADVERTISER	URL	PHONE	PAGE
Altaworks	www.altaworks.com	603-598-2582	17
BEA	http://developer.bea.com	408-570-8000	2,3
Borland	http://info.borland.com/new/jb7/5125.html	800-252-5547	35
Covasoft	www.covasoft.com/today's_tech/	888-963-2686	52
Intel	www.intel.com/ad/bea	408-765-8080	31
Mongoose Technology	www.portalstudio.com	281-461-0099	41
Panacya	www.panacya.com	877-726-2292 x3344	7
Performant	www.performant.com/weblogic1	866-773-6268 x2	51
Precise	www.precise.com/wldj	800-310-4777	15
Sitraka	www.sitraka.com/performance/wldj	800-663-4723	11
Sitraka JClass ServerViews	www.sitraka.com/jclass/wldj	800-663-4723	29
Sonic Software	www.sonicsoftware.com/websj	866-438-7664	21
SYS-CON Media	www.sys-con.com	888-303-5282	48
Web Services Journal	www.wsj2.com	888-303-5282	49
Wily Technology	www.wilytech.com	888-GET-WILY	4

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

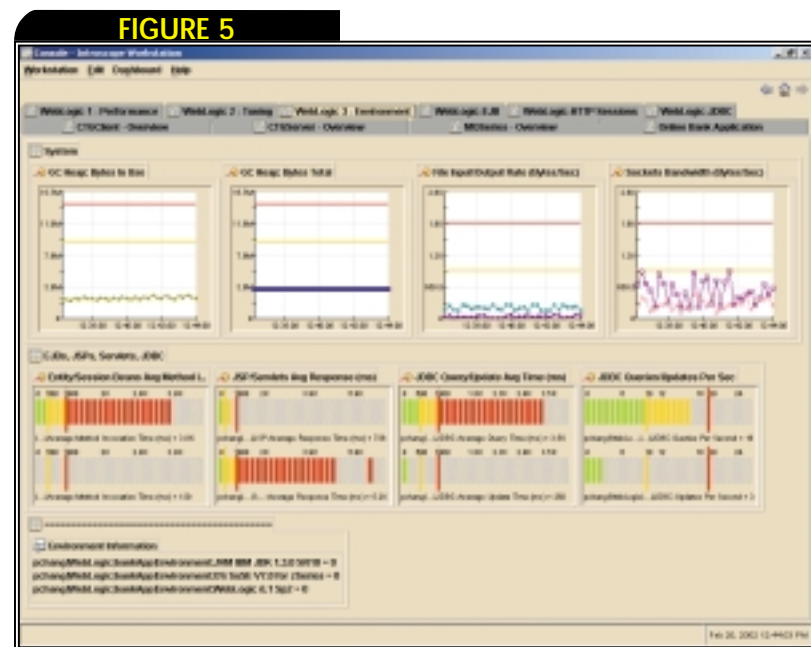
LOOK WHAT'S COMING NEXT MONTH

- Recovering From An Invalid System Password**
Steve Mueller and Scot Weber are in the Admin Corner with a look at recovering from an invalid system password. You can avoid production disasters and downtime with this simple, but hidden, utility.
- WebLogic Server on the Mainframe, part 3**
Tad Stephens and Eric Gudgion offer tips and tricks they have learned in deploying J2EE applications on the mainframe. They include a number of configuration settings, tuning suggestions, and descriptions of existing production applications.
- Using an Implementation Model to Identify WLS Packaging Issues**
Andy Winskill returns to WLDJ with a look at Rational's Unified Process and how it can be tailored to meet your clients' needs. In particular, he uses the implementation model to structure the physical artifacts (or files) within the project.
- Understanding the Advisor Framework**
Dwight Mamanteo begins a series that demystifies the frameworks embedded in WebLogic Portal. This adaptable engine offers plug-in points for extending WebLogic Portal's personalization functionality.
- JMS Performance Notes**
Peter Zadrozny, founding editor of WLDJ, discusses the importance of testing your application and why you can't extrapolate results from similar applications.
- WLS Certification**
Dave Cooke wraps up his guide to WLS 6.1 certification with a look at server administration, including deployment, security, and using the server console.



will access legacy applications and data. The mainframe has many applications, databases, and files that can be accessed in a variety of ways, including Web services, the J2EE Connector Architecture, native API calls exposed to Java, or JDBC APIs. A future article will explore each of these options in more detail, and will look at how using tools such as BEA WebLogic Workshop can make it easy to design and assemble applications with access to mainframe-based applications and data. We'll also detail a production application using WebLogic on the mainframe and some of the development, deployment, tuning, and management tips and tricks learned from production deployments.

A number of options are available to customers who need to deploy J2EE applications on the mainframe, including z/OS, Linux under z/VM, and running Linux natively on the mainframe operating system. The steps to install and configure WebLogic on the mainframe are similar to the steps required on other platforms, but there are some key differences. The actual deployment model the customer selects will depend on a number of factors (skills, availability, costs, etc.). Each option has benefits and drawbacks; however, WebLogic deployment on the mainframe provides a high level of performance and integration.



Willy Introscope workstation

The next article in this series will address development and testing strategies, and will describe lessons learned by current BEA customers running production WebLogic-based applications on the mainframe. We'll cover details of the integration strategies for access to mainframe systems and data, including how to use Web services to gain access to and from mainframe systems.

Listing 1.

```
//WEBLOGIC EXEC PGM=BPXBATCH,REGION=512M,
//      PARM='sh /usr/BEA/weblogic/config/petstore/startPetStore.sh'
//STDOUT DD PATH='/usr/BEA/weblogic.out',
//      PATHOPTS=(O_WRONLY,O_CREAT,O_APPEND),
//      PATHMODE=(SIRWXU,SIRGRP,SIROTH)
//STDERR DD PATH='/usr/BEA/weblogic.err',
//      PATHOPTS=(O_WRONLY,O_CREAT,O_APPEND),
//      PATHMODE=(SIRWXU,SIRGRP,SIROTH)
//SYSPRINT DD SYSOUT=*
//SYSERR DD SYSOUT=*
//SYSOUT DD SYSOUT=*
```

Your Career Depends On It!

Subscribe NOW to the Finest Technical Journals in the Industry

www.sys-con.com/java/

www.sys-con.com/webservices/

www.wbt2.com/

www.sys-con.com/xml

www.sys-con.com/coldfusion/

www.sys-con.com/weblogic/

www.sys-con.com/pbdj/

subscribe online at

www.sys-con.com or call 888 303-5282 / 800 513-7111



WebServices JOURNAL

.NET J2EE XML

Only \$69.99 for 1 year (12 issues)*
*Newsstand price \$83.88 for 1 year
Subscribe online at www.wsj2.com or call 888 303-5282

LEARN WEB SERVICES. GET A NEW JOB!

Subscribe today to the world's leading Web Services resource

The Best .NET Coverage Guaranteed!

Get Up to Speed with the Fourth Wave in Software Development

- Real-World Web Services: XML's Killer App!
- How to Use SOAP in the Enterprise
- Demystifying ebXML for success
- Authentication, Authorization, and Auditing
- BPM - Business Process Management
- Latest Information on Evolving Standards
- Vital technology insights from the nation's leading Technologists
- Industry Case Studies and Success Stories
- Making the Most of .NET
- Web Services Security
- How to Develop and Market Your Web Services
- EAI and Application Integration Tips
- The Marketplace: Tools, Engines, and Servers
- Integrating XML in a Web Services Environment
- Wireless: Enable Your WAP Projects and Build Wireless Applications with Web Services!
- Real-World UDDI
- Swing-Compliant Web Services
- and much, much more!



SYS-CON Media, the world's leading Technology publisher of developer magazines and journals, brings you the most comprehensive coverage of Web services.



*Offer subject to change without notice



BY ANDY WINSKILL



Turning Technology into Profit

THE NEXT BIG THING: INTEGRATION

BEA's eWorld Europe rolled into Paris June 25–26 with more than 2,200 attendees. The subtitle of this year's conference was "Turning Technology into Profit." Alfred Chuang, BEA's founder, CEO, and president, in his opening keynote emphasized that for technology to be effective it had to maximize the value that it returned to the business and the business's clients. His view was that for corporations to realize profit from technology they must move to become a distributed, integrated enterprise. BEA has a vision to help this, "... do more with less." BEA wants to simplify the production of applications and help customers cope with the changes that technology produces. BEA's strategy is USE: companies should have a Unified architecture, which allows companies to Simplify, develop, and deploy applications within an Extensible infrastructure. This strategy is the basis for the new BEA WebLogic 7.0 Platform. It integrates all of BEA's Java products into one product. Some of the aims for this product include common administration and a common, single service pack for the platform (which will be tested across all the products in the platform).

The conference keynotes had a recurring message – integration

through Web services. This wasn't just a jump on the hype bandwagon, presenters acknowledged there are and would be problems. It was mentioned on more than one occasion that there is now a "network effect," where a critical mass is being achieved behind Web services. Part of BEA's strategy for Web services is to simplify the production of Web services with the general availability of WebLogic Workshop, announced at the conference.

Exhibitors

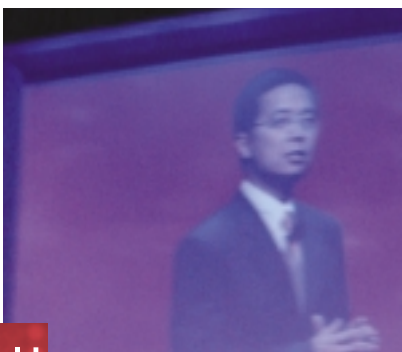
A number of exhibitors was at the conference – large booths from major vendors such as BEA, HP, Intel, and Accenture and smaller exhibits from the likes of Rational, TogetherSoft, Sitrika, ILOG, and others. The stands generated a lot of interest, though not as much as the World Cup semifinals, which were also taking place. The audience was entertained with a number of television screens showing the matches and the excited atmosphere was enhanced by the French cuisine and, more importantly, the French wine!

Sessions

There was a large number of ses-

sions, both business and technical. The technical sessions ranged from basic product overviews to excellent technical presentations, such as Phil Aston's talk on using the Grinder and Rashi Nigam's in-depth look at WLI performance tuning.

The technical keynote presentations were from Scott Dietzen, CTO of BEA; and Adam Bosworth, BEA's vice president of engineering of the Frameworks Division. What was interesting from both presentations was BEA's emphasis on integration. Scott outlined the view that integration could have as much impact on computing as the Internet. Adam argued not only



that application-to-application integration could only be consistently achieved through the use of Web services, but also that the programming model currently used with Web services would have to change from synchronous

rpc-like semantics to asynchronous message-processing semantics. Adam highlighted this point with an effective demonstration of WebLogic Workshop and a BlackBerry device.

Another highlight of the technical keynotes was a demonstration from LevelSeas (a B2B hub – www.levelseas.com) of a real, live B2B project. It effectively showed how mobile devices and multiple servers could be integrated using simple Web services. I found this demonstration exciting as it was a real application currently processing business-to-business transactions. It made maximum impact as LevelSeas have developed a graphical monitor showing the processing of the Web-service calls between machines. As someone who thinks that all computers should have lots of flashing lights, I found this demonstration really appealing!

There was also an interesting presentation from Tyler Jewell and Michael Smith on the "Top 10 Architectural Mistakes." This humorous discussion drew on their experience in the field – most mistakes are a result of poor planning, poor design, or unnecessary overengineering. Comments I would certainly agree with!



AUTHOR BIO...

Andy Winskill is principal consultant at Rosewood Software Services Ltd. in the UK. Andy specializes in BEA and Rational software, and has more than 10 years of experience designing and constructing EAI and B2B applications. Rosewood Software Services is a BEA and Rational partner.

CONTACT: andy.winskill@rosewoodsoftware.com

Performant
www.performant.com/weblogic1

Covasoft

www.covasoft.com/today's_tech/