

BEA WebLogic

DEVELOPER'S JOURNAL

Volume:1 Issue:2

weblogicdevelopersjournal.com

web services **EDGE**
world tour 2002

WASHINGTON, DC FEBRUARY 26
NEW YORK, NY MARCH 19
SAN FRANCISCO, CA APRIL 22

Page 32 Register for Web Services
Edge 2002 East Gold Passport,
Attend the World Tour FREE!

FROM THE EDITOR
Building Skyscrapers
by Jason Westra
page 5

GUEST EDITORIAL
Peeling the Onion...
by Sean Rhody
page 6

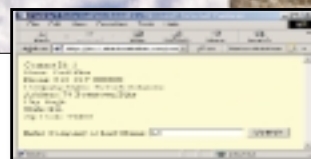
TECH SUPPORT
Comm 101
by John Greene
page 20

NEWS &
DEVELOPMENTS
page 58



Paul Elisii & Larry Zappaterrini 22

SAM'S SOAPBOX: **Web Services in the World of Java** Achieving interoperability and integration, appropriately



8

Sam Pullara

TRANSACTION MANAGEMENT: **When They're Hot and When They're Not** Where the transaction specification fits into the J2EE jigsaw



10

Peter Holditch

CLUSTERABLE EJBS: **An Introduction to WebLogic Server Clustering PART 2** Load balancing, fault tolerance, and configuration of EJBS

12

Tyler Jewell & Lawrence Kaye

TIPS AND TRICKS: **Shared Session** Using EJBs to access persistent data



34

Mika Rinne

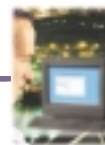
JAVA: **Building and Deploying Apps with Ant** Avoid collaboration nightmares and save hours of debugging headaches



40

Michael Gendelman

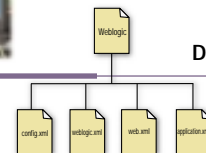
WLDJ FEATURE: **Using EJBGen** Creating one file that does the work of three



46

Dion Almaer

UPGRADES: **J2EE App Migration from WebLogic Server 5.1 to 6.1** Speeding the process of redeploying EJBS, servlets, and JSP



54

Dmitriy Yakubovich

BEA

www.developer.bea.com

BEA

www.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 EDITOR**
JAMIE MATUSOW**ASSOCIATE EDITOR**
JEAN CASSIDY**WRITERS IN THIS ISSUE**DION ALMAER, PAUL ELISII,
MICHAEL GENDELMAN, JOHN GREENE,
PETER HOLDITCH, TYLER JEWELL, LAWRENCE KAYE,
SAM PULLARA, SEAN RHODY, MIKA RINNE,
DMITRY YAKUBOVICH, LARRY ZAPPATERINI**SUBSCRIPTIONS**For subscriptions and requests for Bulk Orders, please send your letters to Subscription Department.
SUBSCRIPTION HOTLINE:
SUBSCRIBE@SYS-CON.COM
Cover Price: \$15/Issue
Domestic: \$149/YR (12 Issues)
Canada/Mexico: \$169/YR
Overseas: \$179/YR
(U.S. Banks or Money Orders)**PUBLISHER, PRESIDENT AND CEO**FUAT A. KIRCAALI
VP, PRODUCTION
JIM MORGAN**SENIOR VP, SALES & MARKETING**
CARMEN GONZALEZ**VP, SALES & MARKETING**
MILES SILVERMAN**VP, EVENTS**
CATHY WALTERS**VP, BUSINESS DEVELOPMENT**
GRISHA DAVIDA**CHIEF FINANCIAL OFFICER**
BRUCE KANNER**ASSISTANT CONTROLLER**
JUDITH CALNAN**ACCOUNTS PAYABLE**
JOAN LAROSE**ACCOUNTS RECEIVABLE**
JAN BRAIDECH**ACCOUNTING CLERK**
BETTY WHITE**ADVERTISING ACCOUNT MANAGERS**
MEGAN RING • ROBYN FORMA**ASSOCIATE SALES MANAGERS**
CARRIE GEBERT • ALISA CATALANO
KRISTIN KUHNLE**CONFERENCE MANAGER**
MICHAEL LYNCH**EXECUTIVES, EXHIBITS**
MICHAEL PESICK • RICHARD ANDERSON**SHOW ASSISTANTS**
NIKI PANAGOPOULOS • JACLYN REDMOND**ART DIRECTOR**
ALEX BOTERO**ASSOCIATE ART DIRECTORS**
AARATHI VENKATARAMAN • LOUIS CUFFARI
CATHRYN BURAK • RICHARD SILVERBERG**WEBMASTER**
ROBERT DIAMOND**WEB DESIGNERS**
STEPHEN KILMURRAY • CHRISTOPHER CROCE**CONTENT EDITOR**
LIN GOETZ**JDSTORE.COM**
ANTHONY D. SPITZER**CUSTOMER SERVICE**
ANTHONY D. SPITZER**CUSTOMER SERVICE LIASION**
PATTI DEL VECCHIO**EDITORIAL OFFICES**SYS-CON Publications, Inc.
135 Chestnut Ridge Road, Montvale, NJ 07645
Telephone: 201 502-3000, 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

Building Skyscrapers with WebLogic

**BY JASON WESTRA**
EDITOR-IN-CHIEF

In the mid 1990s, I worked with an application development environment (ADE) called Forte – essentially, PowerBuilder on steroids. It allowed for scalable, distributed applications to be developed, debugged, and deployed easily within a single environment. The technology was really cool. Sun Microsystems thought so too...and duly purchased the company, Forte Software, Inc., a few years ago. Anyway, Forte's marketing group wanted to express the ADE's power through its packaging – a tough message to relay to developers, who tend to be more interested in the actual installation CD than the box it comes in. However, the design on the cover of the Forté ADE box hit a home run, especially with those who had architected a complex, n-tiered system before. The cover had a large building on it. The building was constructed following an architect's blueprint, and assembled from components such as pre-built support beams, windows, and so on.

To me, the message was clear, and following in Forte's footsteps, whenever I talk about distributed architectures, I often use the analogy of a building as well. To help people understand the importance of architecture in a large system, I like to make the comparison between building a doghouse and building a skyscraper.

Alan Kay, one of the founders of the famed Xerox Palo Alto Research Center, once said, "Architecture dominates materials when addressing highly complex structures." In other words, materials should dominate when building a simple doghouse. "Should I use plywood or pressboard? Should I use a hammer or a nail gun?" There's no real need to draft a blueprint of a simple doghouse, since it would add little value when deciding what materials and tools to use for the job.

On the other hand, when you are designing large, complex buildings or enterprise systems, architecture dominates. Architecture provides a blueprint of the solution you're building. A contractor building a skyscraper without a blueprint might be left wondering if the plumbing pipes in the middle of the conference room could have been avoided whereas a blueprint of the building would've flagged the mistake immediately.

Similarly, software architecture isolates complexity into manageable components, allowing you to see problems with your design more easily. Architecture separates interface from implementation, to allow a skeleton structure to remain even when you are replacing the guts. Architecture also allows you to plan for portability and flexibility if change is necessary.

As Java became a contender for enterprise development, homegrown architectures arose around a disparate group of Java APIs like RMI, and servlets for handling HTTP requests. On their own, these APIs were used to solve problems like distributed communications; but when, where, and how to use them together was left up to you and your architectural guidelines.

Today, the Enterprise Java APIs have been bundled into a logical package called the Java 2 Enterprise Edition (J2EE) platform. The J2EE platform and the Sun Blueprints Design Guidelines for J2EE (J2EE blueprints) address difficult architectural problems such as these. With J2EE, Sun Microsystems – aided by contributions from BEA Systems and others – has laid a foundation for developing complex systems in Java.

I agree that architecture must dominate a complex system. However, I believe that the right materials and tools are also necessary to build a large building or system properly. Can a skyscraper builder be successful without a forklift or crane? How high can the forklift lift? To what heights can the crane reach?

The J2EE platform is merely a foundation upon which vendors provide tools necessary to build scalable solutions. BEA, an early adopter of J2EE, provides the most scalable enterprise development platform in the industry. This month in *WLDJ*, we'll investigate why WebLogic is the platform of choice when developing J2EE applications.

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

CONTACT:jason@sys-con.com

Peeling the Onion...

IT'S WHAT EVERY ENTERPRISE ARCHITECT LIVES FOR!

BY SEAN RHODY



Probably one of the most interesting tasks that can be given to an architect (system architect, program architect, lead architect, application architect – pick the title that resonates most in your environment) is the task of evaluating or determining the direction of an enterprise architecture.

It's a prestigious task – the opportunity to define the software basis for an entire corporation. It usually represents the Holy Grail to an architect. But it's also a task fraught with dangers – some real, some imagined.

From the perspective of a *BEA WebLogic Developer's Journal* reader, we can assume that the direction of the corporation from an IT standpoint is Java – in particular, Enterprise Java. So, at first blush, the task seems trivial: select a platform (WebLogic obviously), select a hardware vendor and a database vendor, and voilà – instant enterprise architecture.

Sweat and Tears

It's about this time that the cold sweat sets in. Because it's an extremely rare company that doesn't already have some previous investment in technology that it wants to milk for all it's worth. After all, if mainframes are so obsolete, why haven't they gone away? The answer is "ROI": the need to deliver Return On Investment.

So our intrepid architect now begins to realize that defining the enterprise architecture is a bit like defining an onion. No, not because it smells and lots of people don't like it...but because it has layers. And it's at this point that he realizes that the easy layers, the ones that

everyone can see and relate to, have already been chosen for him.

As an example, we can more or less just assume that it's an Oracle database. Everyone needs a database and Oracle is the king, so it's easy to see the need to interface with Oracle. Databases aren't truly commodities, but it's been a long time since database selection worked as a selling point of an enterprise architecture. Like plumbing, almost the only time you give it a thought is when it isn't working. Still, when you choose a database, you also restrict some of your choices.

Likewise, the hardware choice is important because it determines your operating model and support structure. Often this is a murky subject, because the omnipresent and ever annoying mainframe lurks at the heart of most organizations, along with the idea of a separate transaction monitor, like IBM's CICS.

Which is fine you say; after all, Java will run on a mainframe. And yes, it will. But now we've peeled back another onion layer and started to understand about connecting the components to each other. And we start thinking – well, CICS is really about business logic, and don't I want that in my J2EE Server instead? Gotcha.

Deep Inside the Onion


So now we come to the more painful, hidden parts of the onion. Because up to now we've talked about the CIO level of architecture, namely the infrastructure and software platform. But once we get past that layer we realize

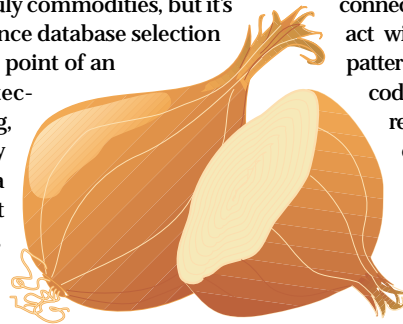
that there are design patterns, methodologies, and best practices that need to be applied as well. The concept of local EJBs in the latest specification is a direct result of the realization that although you could have a client application (be it JSP/Servlets or Java applications) create direct connections to every EJB it needs to interact with, it's not an appropriate design pattern in terms of network efficiency and code execution, not to mention that it really negates the benefits of container-managed transactions.

In fact, the whole J2EE blueprint is an attempt to define best practices around a technology that's powerful, but easily misused. And that's just staying within the relatively safe confines of Java

and J2EE. Now throw a monkey wrench into it with CICS, or add a CRM package that uses only JSP and Servlets, or a business rules engine that doesn't really understand EJBs and you can see where the architect begins to earn his pay: not by selecting the outside layers of the onion, but by ensuring that the inside layers aren't rotten.

Powerful Combination

Fortunately, with WebLogic, architects who are working in the Java space have pretty decent answers to these questions. CICS can be encapsulated in an EJB; WebLogic Integrate provides a business process engine that integrates well with EJB; and WebLogic Portal provides the plumbing around personalization. Add on the capability to treat WebLogic as a set of Web services, and that's a pretty powerful combination. Getting it to work together with the other systems and designing a flexible, scalable architecture, that's the real meat of the architecture. And that's what every architect lives for – peeling the onion. 



AUTHOR BIO...

Sean Rhody is the editor-in-chief of *Web Services Journal*. He is a respected industry expert and a consultant with a leading Internet service company.

CONTACT: sean@sys-con.com

Mongoose Technologies

www.mongoose.tech.com

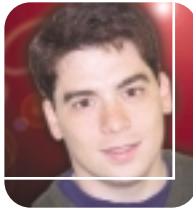
Web Services in the World of Java

ACHIEVING INTEROPERABILITY AND INTEGRATION, APPROPRIATELY

BY SAM PULLARA

THIS MONTH I'M GOING TO TALK about Web Services (capitals intended - I'll explain in a minute). Specifically, as one of the architects of BEA's SOAP/WSDL stack, I'd like to talk to you about Web Services and where they fit in with J2EE and Java in general.

I capitalize the term "Web Services" because when I use it I'm referring to a very specific set of specifications and protocols. Many companies have retargeted their marketing machines at this space and they're using the term "web services" (lowercase s intended) to mean a wide variety of things, some even stretching it out to mean regular Web sites. Web Services are, by my definition, services whose shape (or interface) is determined by a corresponding WSDL document and that are accessed through some binding to the SOAP protocol. Currently, services falling under this umbrella should follow the SOAP 1.x and WSDL 1.x specifications published by W3C. Commonly, the transport protocol is either HTTP or SMTP, but there are probably others out there. There's another type of Web Service that I'll refer to as Business Web Services. These are built using conversation-based protocols on top of something akin to SOAP (or even SOAP itself). Some examples include ebXML, BizTalk, and RosettaNet.



ality to ensure that while Web Service usage patterns are developing, you're not left waiting for the next release.


On the performance side, we developed an extremely fast XML parser and layered a very programmer- and execution-efficient API on top of it. Known as an XMLEventStream, this API allows you to access your XML data efficiently like SAX, while at the same time retaining a lot of the convenience of DOM. We also provided tools to deploy Web Services straight from your SSBs and your JMS destinations, so that you can get started right out of the box.

The more difficult decision was to support only very simple types. Our implementation supports the base types (like int and String) and structs (mapped to JavaBeans or Maps) and arrays of these types. We left the type codec very extensible so that you can map more complicated constructs to your own Java class implementations, but we don't encourage using it if you don't have to. It should be used only when interoperability would suffer without it. We also left out some of the higher-level J2EE constructs such as entity beans.

J2EE Should Never Be Stretched Too Far

Ensuring that your system is interoperable with many different clients has become very important. When you're architecting your applications, you need to make careful decisions when choosing what functionality you'll expose and how you'll expose it. For instance, many people are building Web service gateways to entity beans. Since I don't even believe that a Java client should access entity beans over the network (access should always be through a session bean), how could I support the access of entity beans through what should be a stateless interface with no support for remote references?

Another alarming feature that's becoming prevalent with the more "dynamic" implementations (by which I mean unmanageable) is the passing of serializable Java datatypes directly in a SOAP message. Some are even BASE64-encoding the serialized byte array! This is exactly the sort of thing developers should not do when building interoperable systems. All access points to your application should be designed with cleanliness, compatibility, and above all, maintainability firmly in mind.

If there's one thing the browser has taught us, it's that it is possible to have cross-platform applications hosted over the network. The arrival of Web Services promises that we can host cross-platform application services as well, if only a little care is taken. 

SSB and JMS

When you look at the J2EE programming model for ideal points of integration with Web Services, there are two that jump out at you. Stateless Session Beans (SSB) look almost exactly like a typical Web Service. They have an interface that should map to a WSDL document; they're stateless like RPC-style Web Services; and they have the right granularity for loosely coupled systems. On the other hand, JMS destinations map very well to document-style Web Services. You can define an end-point; routing can be done; and the systems are nicely decoupled from one another. There may be other points of integration between the two systems, but because we know that these integration points are more appropriately decided by the Java Community Process (JCP, see JSR 101 & 109), at BEA we felt that we should stick to the obvious overlaps.

Performant, Extensible, Configurable...and Fast

When using our Web Services stack versus others, the main difference is that it's been built to be highly performant, extensible, and configurable. There are hooks for building extensions to the base function-

AUTHOR BIO...

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

CONTACT: sam@bea.com

Altaworks

www.altaworks.com



Driving home from a meeting discussing the usual mixture of business problems and technological solutions, my eye was caught by a callow youth helping an old lady, laden with shopping, across the main street.

Transactions – When They're Hot and When They're Not

THE TRANSACTION PROCESSING MONITOR IS DEAD, LONG LIVE THE TRANSACTION PROCESSING MONITOR

BY PETER HOLDITCH



AUTHOR BIO...

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

CONTACT...

peter.holditch@bea.com

Sad, I mused – you don't often see that any more. My mind then wandered to hoping that, as technologists, we aren't somehow tacitly colluding in the erosion of the fabric that holds society together. Hmm, I seem to have come over all melancholy. Excuse me whilst I visit The Hunger Site...

Understanding JTA

That's better. Anyhow, I digress. I promised to look this month at when to use transactions and when not to. There's no better place to start than by examining where the transaction specification (JTA) fits into the whole J2EE jigsaw.

At a high level, all J2EE specifications fall into two broad categories: those providing a point technology and those providing a layer that binds a discrete set of specifications into the unified platform that J2EE defines. By way of illustration, messaging between message producers and consumers is a point problem addressed by the JMS specification. Likewise, how to manage server-side business logic accessed from a relatively large and potentially diverse client population is a point problem dealt with by EJB. In the other camp, providing a general purpose naming system for advertising and looking up "stuff" is the job of JNDI, and demarcating logical groups of

data updates done using various "point solution" specifications is the job of JTA.

To cement these two classes of specifications together into a unified platform, it's important to understand how they relate to one another.

In the case of JTA, transactions are typically propagated through the XAResource extension to various interfaces. They're propagated to databases through the JDBC interface, and into messaging systems through the JMS interface; the J2EE Connector architecture provides for them to be propagated to external applications and – in the specific case of WebLogic Server – into Tuxedo applications and potentially from there to mainframe systems via the WebLogic Tuxedo Connector. Finally, with the session synchronization interface, any business logic you dream up can be informed before and after a transaction goes through the commit phase.

On the face of it, anything likely to affect the durable state of an application, can be involved in a JTA transaction. This gives application architects the possibility of dividing the work of their application into however many atomic lumps of functionality they desire, at whatever level of granularity they see.

Of course, J2EE is merely a toolkit; just because you can do something with it, doesn't mean it's a good idea. I always get enraged when I hear things like "Don't use Entity Beans, they don't work," since that seems to me to be akin to saying "Don't use boxing gloves, they don't work." While this might be seen as words of wisdom to someone starting guitar lessons, it comes across as somewhat less perceptive if said to someone on their way to a boxing ring.

Enough hyperbole. The rest of this month's column will explore how transactions might be used to best effect in an application.

Transactions and JDBC Databases

The first thing that usually springs to mind after mentioning transactions is databases. And indeed most of the time transactions are used in conjunction with one or more databases. (It's worth reflecting here on the phrase "one or more.")

The next thought, after transaction and database, is often "performance hit" and people immediately run screaming back to database local transactions. In reality, the perceived "hit" is not big. WebLogic Server optimizes commit processing to a single commit-only phase if a given transaction turns out to only be addressing a single RM instance. This means any overhead caused by a JTA transaction is largely confined to the mapping between the WebLogic Server trans-

action ID and the database's own notion of a transaction, which should be a fast lookup inside the database.

The benefit of using JTA transactions from the start is that, as you come to reuse your components in the future, they'll be able to participate in transactions with larger scope without requiring code changes. This is a big win in terms of maintenance cost and increased ROI through component re-use.

Whilst talking about single or multiple database instances, it's also worth considering the impact of JDBC connection pools. The transaction manager will start a new transaction branch for every connection pool it touches. This means that even if you're only accessing one physical database instance, you'll get a two-phase commit if you do so through two different connection pools. Not only does this defeat the one-phase commit optimization just mentioned, but typically data locked by one transaction branch isn't visible in another. This means you can access the same data twice, in the context of the transaction, and get a deadlock situation between branches of the transaction. On the plus side, however, the two branches can't cause one another unforeseen side effects by manipulating data both rely on in unexpected ways. These issues must be considered when deploying when you decide which components should be accessing which TxDataSources.

On locking, there's another thing: any data you modify as part of a transaction will be locked by the database (subject to the JDBC transaction isolation level you set) until the transaction completes. In this context, "completes" means when the second phase of the commit happens, not when your logic finishes executing. Clearly, if your customer database gets locked in one gigantic transaction, your system will come to a standstill until the transaction completes.

Non-JDBC transactions

Everything I've said about transactions through JDBC to databases also applies to transactions through the J2EE Connector architecture (CA), or through the WebLogic Tuxedo Connector to external applications. The main difference is that it's likely that additional business logic will be executed as well as persistent state being updated in the context of the transaction. There's an additional caveat in the case of J2EE Connector architecture, however: not only is the specification new, and the market for adapters immature as a result, but many applications simply don't have the infrastructure necessary to expose this kind of two-phase commit capability (shame on you, software vendors, for not building on BEA earlier!). So the idea of a completely homogeneous transaction propagated around everything in an IT infrastructure is a little far-fetched at this point. My advice: use it if it's appropriate and it's there – but don't bank on it.

Remember, now, the youth helping the old lady across the road. This situation is equivalent to doing a two-phase commit between an old lady and a youth, the goal of which is to get old ladies safty across the road. While that's a laudable objective, it's not ideal if you want to get a large number of youths across the street. In this case, you'll let the youths dodge traffic as nimbly as they can and leave the old ladies to effect their traversals as traffic permits. Even having off-loaded the old ladies, the throughput of road-crossing youths will be greatest if you don't force them to cross as a group. This is analogous to lots of small transactions, as opposed to one large one.

We're now moving into the realm of JMS and transactions.

Introduce Some Asynchronicity

J2EE CA, JDBC, and the WebLogic Tuxedo Connector are all designed to send requests and get responses in real time. In contrast, JMS is

designed to send requests reliably, but at some unspecified point in the future. It would be the exception rather than the rule that logic would ever be coded to send a JMS message and then do a blocking wait for the reply – that case is better handled by RMI. What you can do with JMS is send a message as part of a transaction – so it's guaranteed to be dispatched if and only if the transaction commits successfully. You can rest safely in the knowledge that the sending transaction won't be held up by potentially large amounts of downstream logic and updates, which will be done in the context of a different transaction controlled by the message receiver. For example, if you imagine the transaction to be "open new account" and the downstream logic to be "send welcome pack by post," it's clear that the customer database updates must be committed before the mail service delivers the welcome pack. These actions, although logically both part of the same business transaction, should be decoupled. Think of your customer database as a youth, and the U.S. Postal Service as an old lady, or in short use JMS where appropriate to decouple large monolithic transactions into smaller, nimbler ones.

Another reason to use JMS and transactions combined is to protect applications from resources that don't support transactions, such as the J2EE CA cases discussed earlier. If you front-end a transaction-unaware application with a JMS queue, then you are guaranteed that it will update at least once at some future point, if and only if the update transaction commits. This avoids the potentially far-from-trivial task of backing out changes made to application state through "compensating transaction" logic if a transaction rolls back unexpectedly after a real-time update to the non-transactional system. Of course, the downside is that the transaction-unaware application must be capable of detecting multiple repeat updates, since updates could be resubmitted to it if it fails while consuming messages. This is easy if the application's behavior is idempotent (that is, calling it twice with the same input data will leave it in the same state as only calling it once) but harder in other cases. For example, a SetCustomerAddress method can be called repeatedly with the same data to no ill effect, whereas an IncreaseAccountBalance method, if called twice, will try and create money from nowhere. While this would be a pleasant surprise if the account in question is your own, it probably isn't allowed in the business domain.

Summary

To summarize, using transactions to group operations on individual components together into higher-level business transactions is a powerful technique that can promote component re-use without reducing the reliability of the overall system. (For example, allowing the transfer method to be implemented as a transaction surrounding a call to credit and a call to debit.) All powerful techniques need to be used carefully, however, and this one's no exception. Misused transactions can result in applications grinding to a halt with all their data locked. To avoid this, the architect must understand the expected scope and duration of transactions, and use techniques like JMS messaging to introduce some asynchronicity into the system.

That concludes this month's column. Next month, Transactions: the Saga Continues, on business transactions, two-phase commit, and other stories. 





BY
TYLER JEWELL
AND
LAWRENCE KAYE

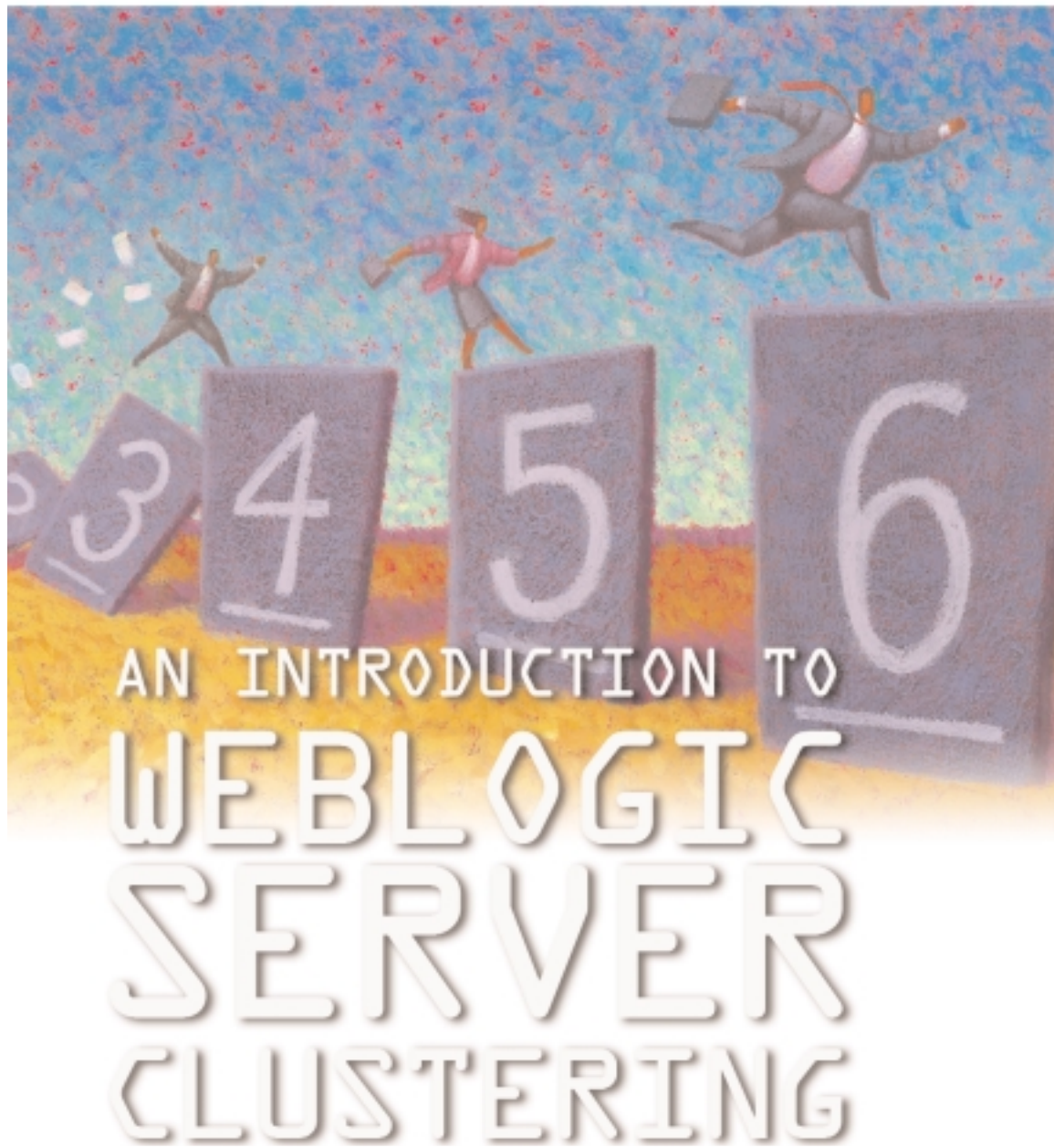
AUTHOR BIOS...

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

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

CONTACT...

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



This is the second in a series of three articles discussing the clustering capabilities of BEA WebLogic Server 6.1 (WLS). This month we discuss replica-aware stubs, their impact on a clustered system, and how they're used with EJBs.

How WebLogic Can Instrument EJBs

WebLogic can provide clustering logic for an EJB in four possible locations (see Figure 1):

1. The JNDI naming server, where the home stub is bound
2. The container
3. The home stub
4. The remote stub

Load balancing, fault tolerance,



distributed systems will still require some form of initial access logic that load-balances JNDI InitialContext requests to the different servers in the cluster. WebLogic recommends using DNS round-robinning, a proxy server, or some hardware local director. All of these technologies give you the ability to represent an entire cluster through a single IP address or DNS name. Each one internally maintains a table of IP addresses and servers that are participating in the cluster and then routes "new" requests using a load-balancing algorithm. Subsequent requests may be re-routed to the server that handled the initial request, depending upon the type of request and the technology used.

CONTAINER

WebLogic can provide some load balancing and failover logic directly within the container. WebLogic does stateful session bean replication to provide minimal failover capability. When a stateful session bean is created, a backup copy can be placed on another server in the same cluster. The backup copy isn't used unless the primary fails, at which point the backup becomes the primary and nominates another backup. Every time a transaction on a stateful session bean commits, the bean is synchronized with its backup to ensure that both locations are current. If the container ever has a system failure and loses the primary, the remote stub of the bean fails over to the secondary server and uses the instance that's located there. Stateful session bean replication is a highly desirable feature since it provides noninterruptible access to some business logic. It isn't designed, however, to provide persistent access to data, since simultaneous failure of the primary and backup servers cannot be recovered. Entity beans should always manage persistent data. Node C in Figure 1 represents this.

HOME STUB

This object is the first object accessed by remote clients and runs locally on a client's virtual machine (VM). Since EJB compilers generate stubs and skeletons at deployment time, the underlying logic in a stub can be WebLogic-specific. WebLogic can instrument some method-level load balancing and failover schemes directly in the stub. Since the primary action of the home stub is to create or load beans on a remote server, the server in which a bean is ultimately created isn't important.

The home stub and the remote stub are the most interesting aspects, since both of these objects are downloaded by a Java client and run within the JVM affiliated with the client, not the application server.

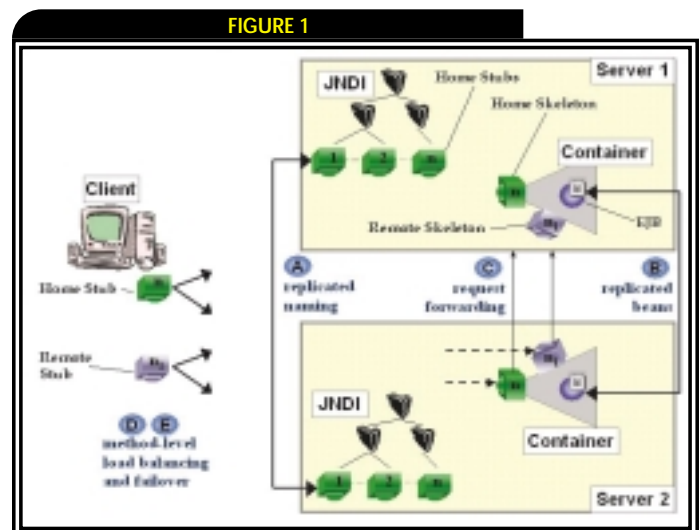
WebLogic can develop load balancing and failover algorithms for EJBs not even resident in their own server! Given all of the locations where WebLogic can instrument one form of load-balancing or failover logic, the permutation of options is enormous.

Let's examine one-by-one what a WebLogic server instruments:

JNDI NAMING SERVER

The JNDI naming server is the initial access point for all clients to retrieve a home stub of a session or entity bean. This doesn't apply for EJB 2.0 message-driven beans, however - they take a message-oriented middleware approach. To make EJBs highly available, WebLogic can easily replicate naming servers across nodes in the cluster. Since the home stubs placed into the naming tree are lightweight and serializable, it's easy for WebLogic to synchronize the trees in a cluster so that all of the servers provide a single, joint naming tree to their clients. Developers may choose to make their EJB container available on one server in the cluster or on all servers in the cluster. In the former scenario, the home stub replicated throughout the naming server has a hard coded reference to the home skeleton on the host server. In the latter scenario, the replicated home stub can reference the home skeleton on a different node. In Figure 1, this is represented by Node A.

Note: This doesn't encapsulate initial access logic. Replicated naming servers make the home stubs of EJBs highly available to all clients, but



Options for Clusterable EJBs

and configuration of EJBs

Effectively, every create(...) and findXXX(...) method could load-balance requests to a different home skeleton in the cluster. Node D in Figure 1 represents this.

REMOTE STUB

The remote stub is instantiated by the home skeleton and returned back to the client. It can perform the same types of load balancing and failover that a home stub can, but WebLogic has to be careful about when it chooses to do so. For instance, if a client has created an entity bean, it's likely that the entity bean will only exist in one server in the cluster. It's too expensive to have the contents of the same primary key active on multiple servers in the cluster if there are not multiple clients all requesting the same entity bean. To increase performance, WebLogic will likely only want to load entity beans in memory lazily when clients request their services. A remote stub that accesses an entity bean cannot freely load balance its requests to other servers in the cluster since the entity bean will only be active on a single server. Essentially, the remote stub is "pinned" to the server that it came from and isn't

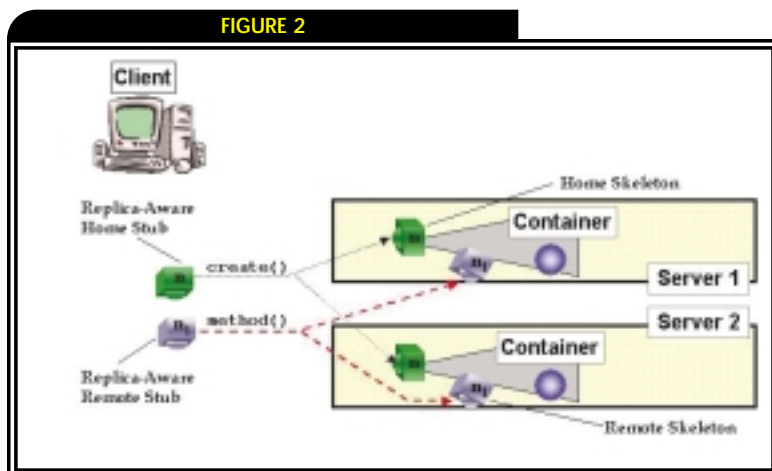
ancing or failover of requests for local clients. Only remote clients can use the load balancing and failover features of WebLogic Server EJBs.

REPLICA-AWARE STUBS

This RMI stub load balancing technology occurs in other places besides EJBs. It uses it to implement JDBC DataSource, JMS Connection Factory, JMS Topic, and JMS Queue objects. All of these objects are implemented as RMI stubs that are "replica-aware" – i.e. stubs that can load balance invocations to different servers hosting the service. For example, the JMS ConnectionFactory interface has a getConnection() method. Since the WebLogic ConnectionFactory is implemented as an RMI stub, subsequent calls to getConnection() will be load-balanced across different servers in the cluster as long as the client is a remote client. This behavior applies to all of technologies listed here.

Stateless Session EJBs

The simplest form of load balancing and failover for EJBs occurs with stateless session EJBs. This section discusses how load balancing, failover, and configuration work (see Figure 2).



Big picture of Stateless Session Beans.

free to load balance at will. Node E in Figure 1 represents this.

Note: WebLogic Server 6.1 implements the latest version of the EJB 2.0 specification (EJB 2.0 PFD 2, 4/25/01). This version introduced local interfaces, special interfaces that allow a client collocated in the same VM as the EJB to access the EJB using pass-by-reference semantics instead of pass-by-value semantics. This gives a client a performance boost and a more robust object model, since local interfaces can have non-serializable interfaces. However, load balancing and failover of requests for EJBs can only be done for clients not co-located in the same VM. If an EJB container were to fail, it's likely that the entire VM hosting the server would fail as well. This would imply that any local clients would not be available and they subsequently couldn't failover to a container on another server. As a result of this behavior, WebLogic Server doesn't provide load bal-

LOAD BALANCING

Since all instances of a stateless session bean type are identical, a home or remote stub can have its request serviced by any available object in any server. As long as the request gets handled, the remote stub has no preference as to whether the instance on the first server or the second server handles it.

WebLogic provides a better level of scalability by configuring home stubs and remote stubs to load balance method invocations across servers. Each method call handled by a stub would execute a "ServerChooser" algorithm in the stub to select the server best suited for handling this method invocation. The ServerChooser would return the IP address or connection to the server best meeting the criteria needed for this method invocation, and the stub would then forwards the invocation to that server.

Since all stateless session EJBs are identical, the home and remote stubs have a lot of flexibility in selecting which server should handle particular requests. WebLogic provides some basic load-balancing algorithms:

- **Random:** A seed-based, random selection of a server to handle a request. This model can incorporate statistical irregularities over the short term.
- **Round Robin:** This popular algorithm sends requests to different servers in a statistically regular way.
- **Weighted Round Robin:** A derivative of round robin that favors certain servers, using ratios. Administrators and developers apply relative weights to different servers that determine how much of the load they should handle. One design pattern used in some production systems is to set to zero the weight of a server

Sitraka

www.sitraka.com

that needs to be brought down for maintenance, thus eliminating any EJB traffic to that server.

- **Programmatic:** You code your own stub-based balancing algorithm as a Java class that's compiled into the stub implementation. The class has a single method that's executed whenever a remote or home stub has to select a server. The method you implement will typically have access to reflection method and input arguments that relate to the method the client invoked. From this information, the method you implement needs to programmatically generate a list of servers or IP addresses of servers to try when the stub needs to load-balance. The order of the list you generate dictates the order of the balancing scheme applied by the stub. This particular algorithm is a great way to implement a staging, test, and production environment for your EJBs. Depending upon the values of the input parameters or the current value of a flag specified in a text file, your programmatic filter can alter requests to different servers depending upon whether you're testing an application or running it in production mode.

FAULT TOLERANCE

Load balancing sounds pretty easy to incorporate, doesn't it? Well, for the most part, it is. However, incorporating fault tolerance and failover of your EJB applications is a little bit trickier. There are two main things that an application vendor has to determine to accomplish a successful failover (and to be able to support fault tolerance).

First, how does a home or remote stub determine if a failure situation has occurred? If an exception is generated, how does a stub know that the exception is a failure condition that cannot be recovered? There are three types of exceptions that a stub can receive: application exceptions (those defined by the bean developer), system exceptions (typically in the form of an `EJBException`), and network/communication exceptions. Application and system exceptions are clearly defined in the EJB specification and typically are propagated to the invoking client to be handled. Network/communication exceptions such as `SocketException`, `CommunicationException`, and others, indicate a much more dire, unable-to-communicate-with server scenario. An application server vendor's stub can intercept all system and communication exceptions and perform a failover to another server – if they can determine that it is safe to do so. Why wouldn't it be safe for them to perform this failover action? Read on for the answer!

The second main consideration is: At what point in the invocation did the failure occur? There are three possible situations that the stub must address:

1. The failure occurred before the server-side invocation started. This is a great situation for the stub to be in. If it can determine that a failure occurred after the method request was

sent but before it was invoked on the server, the stub can treat this failure as a load-balance scenario and redirect the method invocation to any other available server.

2. The failure occurred after the server-side invocation completed, but before the response message was properly propagated to the client. This is also a great situation for the stub to be in because it requires no further action.
3. The failure occurred during the server-side invocation. This presents a sticky situation for the stub. The method that was invoked on the server may or may not have altered some server-side resources that impact future behavior of the system. If the stub makes a subsequent request on a method that impacts server-side resources or data, the stub may inadvertently perform the same altering action twice in a row during a failure recovery, which would cause the system to have indeterminate behavior! If the method doesn't perform any of these altering actions then it could safely invoke the method again; if it does alter the system, it can't.

Unfortunately, it's very difficult for a stub to positively determine that the system is in either of those states. As a result, if a failure situation occurs, a stub will likely have to assume that the server was in the worst-case scenario (i.e., 3) – even if in reality it was in one of the happier scenarios. So, how's this resolved? WebLogic supports the concept of “idempotence” for methods. An idempotent method is one that yields the same result on subsequent invocations if the same input arguments are passed in. A non-idempotent method is one that alters the state of the system (yields different results) on subsequent executions.

Types of methods that are idempotent include:

- Any method that acts as a “reader/getter” method
- Any method that performs a nonaltering query
- Any method that accesses a resource manager, but leaves the state of the resource manager unaltered

As a bean developer, you're aware of the behavior that the methods you write will have on a system. You are responsible for specifying which methods in your bean are idempotent and which are not. At runtime, WebLogic can freely perform a failover switch during a failed method invocation if the method that was being executed is idempotent. After all, if the method doesn't alter the state of the system, then the stub is confident that the existing system is intact and that a new invocation won't have any odd side effects. In situations where a failure situation occurs on a method that is idempotent, the stub will be forced to throw the communication exception that it receives to the client. Your client application will then have to decide whether or not it should try another invocation:

```

try {
    MyHome home = (MyHome) ctx.lookup("MyEJB");

    // This should use PortableRemoteObject.narrow().
    My remote = (My) home.create();

    remote.invokeIdempotentMethod();
} catch (javax.naming.CommunicationException exception) {

    // This is a type of Naming Exception
    // This is an acceptable exception for me.  Calling
    // this method again
    // will not adversely impact the system.

    try {
        remote.invokeIdempotentMethod();
    } catch (java.io.IOException exception) {

        // This must be some sort of socket exception.
        // I don't want to call this method again, so now
        // it must be
        // handled in another way that you determine!

    }
}

```

CONFIGURATION

Making a stateless session EJB support load balancing and failover is quite simple. You don't have to change the way you develop the bean. Rather, all configuration is done in the `weblogic-ejb-jar.xml` deployment descriptor (see Listing 1).

Stateful Session EJBs

Now let's examine how load balancing, failover, and configuration work in stateful session EJBs.

LOAD BALANCING

With stateful session EJBs, a client is "pinned" to the server object it creates. As a result of this pinned nature of clients to instances, remote stubs are wired to a particular server that contains the instance the client is referring to. WebLogic provides no load balancing of remote method invocations as a result of this behavior.

However, WebLogic does provide load balancing of remote home method invocations for stateful session EJBs. Since a `home.create(...)` merely creates an instance on a server, it doesn't matter which server the instance gets created on. WebLogic uses a load-balancing algorithm to direct different invocations to different servers as a result.

FAILOVER

In order for failover to be successful with stateful session EJBs, the state of the stateful session EJB must be located on another machine. WebLogic supports primary/secondary replication of stateful session EJBs to support failover of remote method invocations. A primary stateful session EJB instance is stored in the server where the EJB is created. The container will replicate the instance to a secondary

server. If the server that hosts the primary instance becomes unavailable, remote method invocations will be redirected to the server with the secondary instance. The secondary instance will then become the primary and nominate another server to host the secondary instance. The remote stub maintains the IP addresses of the servers that host the primary and secondary instances.

CONFIGURATION

Just as with stateless session EJBs, all configuration of stateful session EJBs is done in the `weblogic-ejb-jar.xml` deployment descriptor (see Listing 2).

Entity Beans

Now let's look at entity beans, starting with cluster-aware home stubs. Then we can consider read-write entity beans, read-only entity beans, and how to get the best of both worlds by implementing the read-mostly pattern.

CLUSTER-AWARE HOME STUBS

Just like stateful and stateless session beans, entity beans can have cluster-aware home stubs. This allows for lifecycle methods to be distributed amongst the server instances in a cluster. For example, load balancing can be attributed to the database record creation, deletion, and query functionality behind the `create()`, `remove()`, and `findXXX()` methods defined in the bean's home interface. The EJB 2.0 specification introduced the concept of home methods, business functionality that can be attributed to groups of database records, rather than a single record. Calls to these home methods can be load balanced through the cluster-aware home stubs. As with session beans, an entity bean's home stub is made cluster-aware by setting to true the appropriate `home-is-clusterable` tag in the `weblogic-ejb-jar.xml` file.

READ-WRITE ENTITY BEANS

The caching strategy selected for a given entity bean determines the possible clustering behavior of the bean's remote stub. Setting the value for the appropriate `cache-strategy` tag in the `weblogic-ejb-jar.xml` file configures this behavior. By default, entity beans are deployed with a read-write caching strategy, allowing clients to use the beans to edit database records. Creating or finding a read-write bean returns a remote stub pinned to a single server, which means that load balancing and failover capabilities are restricted to the home stub. Since, in this situation, multiple beans representing the same underlying database record could simultaneously exist within the cluster, data integrity must be preserved by having each bean instance read from the database before each transaction and write to the database with each commit operation.

READ-ONLY ENTITY BEANS

In situations where only data retrieval is necessary, entity beans can be deployed with a read-only

caching strategy. Accessing a read-only bean returns a cluster-aware remote stub, which can load-balance on every business method call – thus improving scalability. In addition, the contents of read-only beans are cached on every host server to avoid database reads, further improving performance. In order to be classified as read-only, a candidate entity bean's methods must be idempotent.

While EJB clients can't modify read-only entity beans, the data the beans represent can be changed by an external source. With regard to caching, a tradeoff must then take place between quick access to the data and maintaining the most current snapshot of the data. In WebLogic Server 6.1, two data update strategies are available: periodic updates or programmatic invalidation.

The periodic update strategy is governed by the read-timeout-seconds deployment descriptor parameter. By setting this parameter in the weblogic-ejb-jar.xml file, the container will refresh the data stored in the bean by loading from the database at regular intervals. The advantage of this mechanism is that it's simple and requires no additional programming – all necessary configuration takes place in the deployment descriptor. However, this update mechanism does suffer from one efficiency flaw: the data is blindly loaded into the bean at regular intervals, even if a database update doesn't occur during the established time period.

Programmatic invalidation overcomes this obstacle, adding intelligence to the update of read-only beans. A developer can trigger a refresh of a single bean instance or group of instances of a certain bean class by invoking an invalidate() method, avail-

able through the home interface. When the invalidate() method is called, the WebLogic EJB container will invalidate the appropriate beans on the local server and send a multicast message to all other servers in the cluster to do likewise.

The added control over when a bean gets updated comes with a small price, however. In order to invoke the invalidate methods, the EJB client must explicitly cast the bean's home stub to either the weblogic.ejb.CachingHome or weblogic.ejb.LocalCachingHome interface (the latter for entity beans whose clients are other EJBs resident in the same instance of WebLogic). These interfaces are BEA proprietary extensions, filling in a performance hole in the current specification. The potential improvement in data access performance mitigates the minor inconvenience of removing such method invocations in the event of migrating the EJB client to another application server environment.

APPLYING THE READ-MOSTLY PATTERN

What about entity beans that require occasional programmatic updates? Is there a way to cash in on the performance benefits of read-only beans while still providing write access? Yes: the answer lies in the implementation of the read-mostly pattern, by deploying the same bean class twice – once each with read-only and read-write caching strategy tags. Clients of the read-only bean can benefit from load-balanced access to cached data, while clients of the read-write bean can rely on data integrity during transactions.

If you're relying on syncing read-only beans with periodic updates, via the read-timeout-seconds parameter, we'd make the following recommendations:

Listing 1: Configuration of Stateless Session EJBs

```
<!-- Stateless Session EJB description in weblogic-ejb-jar.xml -->
<!-- Other Tags As Appropriate Here... -->
<stateless-session-descriptor>

<!-- Other Tags As Appropriate Here... -->

  <stateless-clustering>
  <!-- Set this to be true to enable the home and remote stubs to be clusterable. -->
    <stateless-bean-is-clusterable>          true
  </stateless-bean-is-clusterable>

  <!-- Select one of the load-balancing algorithms WebLogic supports. -->
  <stateless-bean-load-algorithm>          round-robin
  </stateless-bean-load-algorithm>

  <!-- The name of a class that implements weblogic.rmi.extensions.CallRouter.
  This class performs routing of remote method invocations. -->
  <stateless-bean-call-router-class-name>   common.QARouter
  </stateless-bean-call-router-class-name>

  <!-- Set this value to true if the methods of the remote interface are
  idempotent and should be automatically failed over in a failure situation. -->
  <stateless-bean-methods-are-idempotent>  false
  </stateless-bean-methods-are-idempotent>
</stateless-clustering>
```

Listing 2: Configuration of Stateful Session EJBs

```
<!-- Stateful Session EJB description in weblogic-ejb-jar.xml -->
<!-- Other Tags As Appropriate Here... -->
<stateful-session-descriptor>

<!-- Other Tags As Appropriate Here... -->

  <stateful-session-clustering>
  <!-- Setting this to true enables load balanced home.create(...) methods. -->
    <home-is-clusterable>                  true
  </home-is-clusterable>

  <!-- This is the load balancing algorithm the home stub will use. -->
  <home-load-algorithm>                   random
  </home-load-algorithm>

  <!-- The name of a class that implements weblogic.rmi.extensions.CallRouter.
  This class performs routing of home method invocations. -->
  <home-call-router-class-name>           common.QARouter
  </home-call-router-class-name>

  <!-- Set this value to be InMemory to enable replication of beans. -->
  <replication-type>                      InMemory
  </replication-type>
</stateful-session-clustering>
```

- Set all values of read-timeout-seconds to the minimum acceptable values that provide frequent enough updates without sacrificing performance – appropriate selection may be an iterative process for a given application.
- If the data behind several classes of read-only beans could be updated in a single transaction, make sure to set all corresponding values for read-timeout-seconds identically.
- Minimize the amount of data updated in the read-write beans and test to make sure that the transactions in which they're involved don't extend past the update intervals of their corresponding read-only beans.

Certainly, to avoid this optimization process developers can choose the programmatic invalidation mechanism described above. However, this practice would reduce EJB portability since the bean implementation class would contain calls to proprietary WebLogic interfaces.

A third option, available when using container-managed persistence, is the use of the invalidation-target parameter when configuring the read-write bean in the weblogic-ejb-jar.xml deployment descriptor file. By specifying the name of the corresponding read-only bean within this tag, that read-only bean will be automatically invali-

dated when the read-write bean is updated. In other words, invalidation-target provides developers with an optimal refresh strategy for read-only beans without sacrificing code portability.

CONFIGURATION

The weblogic-ejb-jar.xml deployment descriptor file shown in Listing 3 illustrates the appropriate settings for an entity bean deployed with the read-mostly strategy.

Conclusion

While the concepts behind EJB clustering take some time to digest, the mechanism for implementing load balancing and failover in WebLogic-based enterprise applications is distilled down to setting a few tags within the deployment descriptor files. Certainly, determining the optimal settings for those parameters that are suitable for a given application is an iterative process. However, by isolating clustering behavior to configuration files, WebLogic minimizes the time necessary to reach that level of optimization.

In the next installment of this series, we'll discuss common architectural practices for tying together presentation and business logic in a clustered environment.

Listing 3: Configuration of an Entity Bean Deployed with the Read-Mostly Strategy

```

<!-- Read-Mostly Entity EJB description in weblogic-ejb-jar.xml -->
<!-- Other Tags As Appropriate Here... -->

<!-- First, the Read-Write component -->
<weblogic-enterprise-bean>
  <ejb-name>exampleReadWriteBean</ejb-name>

  <!-- This example uses the automatic invalidation capability -->
  <!-- so the read-write bean must use CMP -->
  <entity-descriptor>
    <persistence>
      <persistence-type>
        <type-identifier>WebLogic_CMP_RDBMS</type-identifier>
        <type-version>6.0</type-version>
        <type-storage>META-INF/weblogic-cmp-rdbms-jar.xml</type-storage>
      </persistence-type>
      <persistence-use>
        <type-identifier>WebLogic_CMP_RDBMS</type-identifier>
        <type-version>6.0</type-version>
      </persistence-use>
    </persistence>

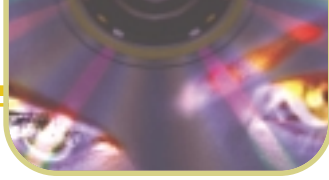
    <!-- This tag links the read-write and read-only beans so that an update
    -->
    <!-- to the former triggers invalidation and refresh of the latter -->
    <invalidation-target>
      <ejb-name>
        exampleReadOnlyBean
      </ejb-name>
    </invalidation-target>
  </entity-descriptor>
  <jndi-name>exampleReadWriteBeanHome</jndi-name>
</weblogic-enterprise-bean>

<!-- Now, the read-only component-->
<weblogic-enterprise-bean>
  <ejb-name>exampleReadOnlyBean</ejb-name>

  <entity-descriptor>
    <entity-cache>
      <!-- Value of 0 means that the bean is never periodically updated -->
      <read-timeout-seconds>0</read-timeout-seconds>
      <!-- Tells the container that the bean cannot manipulate its own data-->
      <concurrency-strategy>ReadOnly</concurrency-strategy>
    </entity-cache>
    <persistence>
      <!-- Tags identical to those for the read-write bean above-->
    </persistence>
  </entity-descriptor>

  <jndi-name>exampleReadOnlyBeanHome</jndi-name>
</weblogic-enterprise-bean>

```



Comm101

EFFECTIVE CUSTOMER/DRE COMMUNICATION

BY JOHN GREENE

AS A WEBLOGIC DEVELOPER, there'll come a time when you'll need to file a support case with BEA; many of you have done this already. Throughout the problem-solving process, you'll be speaking with one of BEA's DREs (developer relations engineers). These folks are very knowledgeable regarding WebLogic, and hopefully the two of you will quickly find a resolution to your problem.

We're dealing with complex systems, however, and for a variety of reasons, things may not go smoothly 100% of the time. Surprisingly, much of this "nonsmoothness" can be attributed more to communication issues than technical issues. While BEA's Support Guidebook provides you with much information about support processes, response times, severity-level definitions, and so on, it doesn't really cover communication-related issues. So how can communication-related nonsmoothnesses be prevented?

As the customer, there are many things you can do to help ensure a positive customer support experience. You can help us help you. In this article, I'll outline some common communication problems that arise between customers and DREs. I'll also present relevant warning signs and recommend corresponding courses of action so you can help keep the process running smoothly.

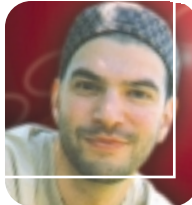
What's Your Problem?

Believe it or not, one of the biggest communication problems is simply problem definition. It's absolutely essential that you communicate what you believe the problem to be and why you believe it to be a problem. If you're on the phone, ask the DRE to explain the problem back to you. Make sure the DRE fully understands every nuance. At the same time, be careful not to let the actual problem be obscured by your impression of the solution. State the problem and make sure the problem is understood by the DRE before suggesting possible solutions.

For example, suppose you are testing failover of your banking application, and to your horror when you bring down one machine, some users can see other users' data. How would you report the problem? Would you conclude that there's a problem with WebLogic clustering and report the problem as such? If so, bzzzt! Try again. Focus on the symptoms. Sure, you should mention the configuration, environment, clustering, etc., but the root problem is users seeing other users' data—focus on that. Don't throw off the DRE by reporting that there's a clustering/failover problem; you don't know that for sure.

One Thing at a Time

In general, it's best to limit each support case to one specific problem. If you have more than one problem, file separate support cases.



Work on one thing at a time; divide and conquer. Remember that your DRE is working on several support cases for different customers concurrently. If you are focused, your DRE will be focused too.

Timeline, Severity, and More

Be certain that your DRE understands the importance of your issue. Is your problem in production or development? Is your entire development effort blocked because of this issue? Are you going into final testing? When is your "go live" date? It's a good idea to communicate this information upfront so that you and your DRE can troubleshoot accordingly.

In the future, the myBEA.com portal will allow you to record/update project/profile information and associate support cases with a project. Until then, it's your responsibility to make sure your DRE knows how this issue is impacting your business.

The People's Court

One of the best ways to keep support cases on track is for both parties to be explicit about whose court the case is in every step of the way. Support management has been training DREs to do this, but this is something you can drive as well.

Here is an example of a breakdown: your DRE sends you an e-mail asking for some configuration files, start scripts, and a test case, and then sets the case status to "Awaiting Customer Information" in the Case Tracking application. Later that day, you discover something new regarding your case and send an e-mail to your DRE stating the new information. Now there may be a problem. You're expecting your DRE to take action based on your recent e-mail, but your DRE thinks that your e-mail was just providing some additional information, and is still waiting for the test case and configuration files.

Avoiding this sort of situation couldn't be easier. Simply conclude your e-mail with something like this: "I know you were waiting for a test case, but in light of this new information, do you still need one? My expectation now is that the ball is in your court regarding this case. Please contact me at your earliest convenience."

Use the Phone

While e-mail and WebSUPPORT (www.bea.com/support/web.shtml) are the preferred means of customer/DRE communication, there are times when using the phone is more appropriate. You can always call the support number in your region (www.bea.com/support/contact_cs.shtml) and ask to speak with your DRE. If you reach your DRE's

—continued on page 25

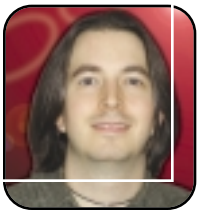
AUTHOR BIO...

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

CONTACT: jg@bea.com

Report Mill Software

www.reportmill.com



BY
PAUL ELISII
 AND
LARRY ZAPPATERRINI

AUTHOR BIO...

Paul J. Elisii is the founder and CTO of e-Tech Solutions, Inc., a Philadelphia-based e-business development company and a BEA Partner. He is active in developing and researching emerging technologies for e-Tech Solutions.

Larry Zappaterrini is an XML and Java developer with e-Tech Solutions, Inc. He was extremely helpful in performing research and in developing the sample applications for this article.

CONTACT...

pelisii@etechsolutions.com
 lzap@etechsolutions.com



BUILDING SOAP
 WITH **WEBLOGIC**

WEB SERVICES HAVE MADE QUITE AN IMPACT SINCE THE CONCEPT AND TECHNOLOGY WERE INTRODUCED. JUST ABOUT EVERY MAJOR VENDOR IS PUTTING CONSIDERABLE EFFORT INTO MAKING THEIR PRODUCTS CAPABLE OF DEVELOPING AND USING WEB SERVICES.

WEB SERVICES

6.1

This article will illustrate how a Web service can be developed and deployed with the Web services capabilities found in WebLogic 6.1. A sample application that illustrates how to use SOAP and Web services to make an application deployable anywhere and accessible from any technology will also be reviewed.

Background

Web services has been getting a tremendous amount of attention over the past year – and in my

opinion, justly so. The concept of a Web service is simple, but opens up tremendous opportunities for companies that use the Internet to provide or use business services. For example, a company that sells heavy machinery needs the services of a shipping company to deliver their products. A Web service allows the heavy machinery company to make an HTTP call over the Internet to the shipping company to send the shipping address and weight. The Web service then determines the cost and allows the heavy machinery company to book the shipping service and pay for it.

Interestingly enough, the heavy machinery company can use this shipping Web service on all their orders, regardless of how the order comes in. The success or failure of this Web services model is not dependent on how consumers make their purchases. Web services are a server-to-server and business-to-business model. Some of the key goals of the Web services business model are to reduce costs, increase customer service, and increase sales by making their services more easily accessible.

Web services are pure back-end business components that are built for computer access – not human access. There is no concern for “look and feel” when you develop Web services. The “usability” and the success of the service revolve around robustness of the methods and properties of the service.

The technologies used in Web services are based on the industry standards of XML and HTTP. This allows collaboration among businesses providing and utilizing Web services across any technology. The heavy machinery company's system can be built completely in Microsoft technology that invokes services on the shipping company's Enterprise Java Beans server.

Fundamentally, as long as Web services are built so that they can transmit and receive data in XML format and are accessible via HTTP, anyone should be able to use those services regardless of their technology. But relying only on those technologies leaves many of the components required for building truly useful Web services to be developed by the service providers and consumers. This can lead to incompatibilities between service providers and consumers and a lot of reinvention of the same core services.

Some additional technologies that are an extension to the basics of XML and HTTP have been developed that define in more detail how Web services can operate. These technologies are SOAP (Simple Object Access Protocol), WSDL (Web Services Description Language), and UDDI (Universal Description, Discovery, and Integration). These further standardize and enhance the ability of Web service providers and consumers to interact.

SOAP is the specification for the actual documents (requests and responses) that contain the

data sent and received in the Web service transactions. WSDL provides information about the Web service, such as the available methods and their arguments. UDDI is essentially a directory mechanism for broadcasting and searching for available Web services on the Internet.

SOAP, WSDL, and UDDI

SOAP is a protocol for exchanging information between Web service providers and consumers. The protocol is based on XML and HTTP, which makes using SOAP very easy when you make calls across distributed heterogeneous computing environments. SOAP enables the use of Web services based on a shared and open Web infrastructure and takes the idea of Web services beyond the initial idea of simply passing XML back and forth between Web systems. It allows developers to perform RPC calls on methods that have been discovered through WSDL.

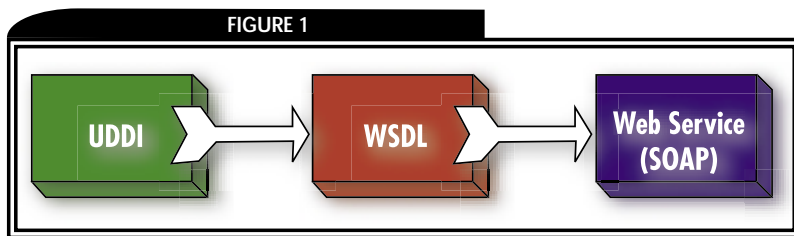


FIGURE 1
Process requestor goes through to identify the service

SOAP is a step beyond other RPC technologies, such as COM or CORBA, because it does not require a particular vendor's product or technology to make it work. CORBA requires a particular ORB that the client and server must adhere to; COM requires specific technologies on both sides to make it work. SOAP is an open standard that can be used with any technology that supports those standards. The SOAP Protocol has the following components:

- The SOAP Message is an XML document that is passed between the Web Service Requestor and the Web Service Provider. This message is made up of a SOAP Envelope, a SOAP Header, a SOAP Body, and a SOAP Fault. The SOAP Envelope is required for all SOAP messages and contains the SOAP Body and SOAP Header. It also can contain encoding styles and, possibly, versioning information. The SOAP Header is optional in a SOAP Message. It is used to provide user authentication or transaction management. The SOAP Body contains the data being sent in the message. The SOAP Fault element is a predefined child of the SOAP Body and contains error status information. It is used only in SOAP Responses.
- A set of SOAP Encoding Rules and binding conventions that process the SOAP Request and Response messages and make them useful for the server and the client. These encoding and decoding processes are implemented by each of the languages that support SOAP,

similar to the way that vendors have developed XML serializers and deserializers. These libraries automatically handle encoding and decoding the SOAP Messages so that from a SOAP Client, the developer just needs to instantiate a SOAP Object, connect to the remote SOAP Web service, and then invoke the methods on that SOAP Server.

The Web Services Description Language is an XML-based specification schema that describes a Web service's methods and interfaces. A Web services client would use WSDL to collect information about the Web service's methods and properties. Most SOAP Client implementations will automatically build bindings to the service's methods and properties automatically so the developer doesn't have to do it explicitly.

In the following example, a Microsoft ASP application is invoking an EJB.

```
set soapclient = CreateObject("MSSOAP.SoapClient")
call soapclient.mssoapinit("http://ets_lnx1:7001/contactlistings/wsdl.jsp")
set Contacts = soapclient.getAllContacts()
```

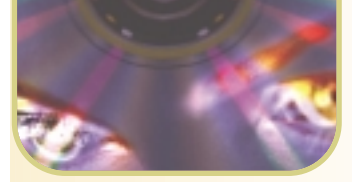
The initial call to "mssoapinit" loads the Web service's WSDL into the client object of "soapclient" so that subsequent method calls, "soapclient.getAllContacts()", can be checked against the supported methods within the Web service. If a call is made to an invalid method name, an error will be generated.

WSDL Documents contain the following elements to define a Web service:

- The **Types element**, shown in Listing 1, describes the data type definitions of the Web service.
- The **Message element** defines the data being communicated via the Web service, including data inputs and outputs, in effect, the messages supported by each of the methods. Listing 2 is an example from our sample Web service. Each method supported in our Web service contains a corresponding Request and Response message in the WSDL.
- The **Operation** defines an action that a Web service can perform.
- The **Port Type** describes the set of operations one or more ports support. In the example below, the port type describes the input and output messages supported by each Operation (method).

```
- <portType name="ContactListingEJBPortType">
- <operation name="getContactsMatching">
  <input message="tns:getContactsMatchingRequest" />
  <output message="tns:getContactsMatchingResponse" />
</operation>
```

- The **Binding** describes the protocol and data format for the port type. As Listing 3 shows, the Binding describes the encodingStyle and other elements for the data format.



- The **Port** is the endpoint location of the service defined by a binding and a network address.
- The **service** describes a group of related ports. The following is an example of the service and port elements of the WSDL used in our sample Web service application. The location element defines the exact location of the actual Web service.

```
- <service name="ContactListingEJB">
  <documentation>todo</documentation>
- <port name="ContactListingEJBPort"
binding="tns:ContactListingEJBBinding">
  <soap:address
location="http://lrx1.etechnologies.com:7001/contact-
service/contacturi" />
  </port>
</service>
```

As you can see, there is a lot involved in the inner details of SOAP and WSDL. WebLogic creates the WSDL and all the SOAP Protocol communications for you automatically so you can focus your efforts on building the actual EJBs and business logic that provide the service functionality.

Another protocol that is a key part of Web services development, UDDI, is used to advertise the existence of a Web service. UDDI services are used to create or find a UDDI registry in either a private or public directory. The UDDI registry itself is an XML packet that contains information about how to access specific Web services and the corresponding information about the organization that provides it. Essentially, it is a way to find WSDLs. Figure 1 illustrates the process that a requestor of a Web service would go through. First it identifies the service that is available via UDDI. Its search will yield a WSDL that contains all the necessary data that allows the requestor to access and use the Web service via SOAP.

WebLogic does not currently have direct support for UDDI, but it is planned for the near future. Our sample Web service application does not include the ability to create a registry for our Web service; however, an overview of UDDI and how it works is described below. The main components of UDDI include:

- A **businessEntity element** that describes information on the business providing the Web service
- A **businessservice element** that contains links to more specific information about the services provided.
- A **bindingTemplate element** that contains information about what the service can do and where to access the service.
- A **tModel element** that describes the data for the service and how to access it.

All of the components of creating and registering into UDDI registries will become automated

processes. From a developer's perspective, UDDI can be viewed simply as a repository containing specific and associated Web services information, like a Java Naming and Directory Interface.

Developers would use these tools to access a UDDI registry to locate services information, prepare systems for Web services compatibility, and to describe their own Web services.

Web Service Support

WebLogic 6.1 supports the ability to create Web services and clients. Specifically, support for all of the following is now available:

SOAP SERVER SUPPORT

- WLS 6.1 provides a servlet-based SOAP implementation that integrates with stateless session beans and JMS.
- SOAP invocations can be mapped to a stateless session bean method or cause a JMS message to be produced for consumption by a JMS listener, such as a message driven bean.
- These components can interact with the rest of the WebLogic Server J2EE application environment.
- The SOAP implementation supports SOAP 1.1 (with attachments) over HTTP; however, attachments are currently ignored (<http://edocs.bea.com/wls/docs61/WebServices/overview.html#1037109>)

SOAP JAVA CLIENT INCLUDED IN 6.1

- WLS 6.1 provides a thin Java client for Web services. No Weblogic.jar is required unless a WSDL is not provided by the service to be consumed, then certain WebLogic classes are needed (<http://e-docs.bea.com/wls/docs61/WebServices/advanced.html#1001373>)
- Thin client downloadable from URL.
- Interface and proxy automatically generated from WSDL.

WSDL SUPPORT

- WLS 6.1 automatically generates WSDL for stateless EJBs and JMS.
- WLS WSDL is accessible and downloadable from user-defined URLs to support publishing and retrieval of Web services.

UDDI SUPPORT

- A UDDI client will be available on the BEA Developer center to support publishing and retrieval of Web services in a UDDI registry.

INTEROPERABILITY

- WebLogic's Web services implementation of SOAP is interoperable with other vendors' implementations. We will demonstrate Microsoft SOAP Client, a Perl SOAP Client, an Apache SOAP Client, and a BEA SOAP Client accessing our WebLogic SOAP Server later in this article.

voice mail, leave a message specifically stating that you'd like a phone call (not an e-mail) in return.

Involve Management

BEA's number one core value is: "Customer issues transcend all others." While we strive to make each customer support experience a positive one (see my previous column in **WLDJ** Vol. 1, issue 1), problems can still occur. As a support manager, I'm often puzzled by how long it takes a dissatisfied customer to contact management. A few months back, I spoke with a customer who hadn't heard from his engineer in two weeks. He wanted to go live in a few days and the problem had now become urgent.

Looking at the case, I saw that its status was "Awaiting Customer Information" and no contact had been made (by either party) in the last 10 days. I informed him that I would speak to the DRE about the lack of communication on BEA's side; DREs should not let cases idle that long, even if they are "Awaiting Customer Information." Out of curiosity, I asked if he had tried to contact his DRE, and he replied that he had not – that he was waiting for his DRE to contact him. Considering the apparent urgency of the problem, I wouldn't recommend following the example set by this customer.

Simply put, at the first sign of trouble, involve management! Call your local support number and ask to speak to your DRE's manager. It's better to call too soon than too late – think of it as preventative maintenance. Convey your expectations to management and agree on a course of action.

BEA

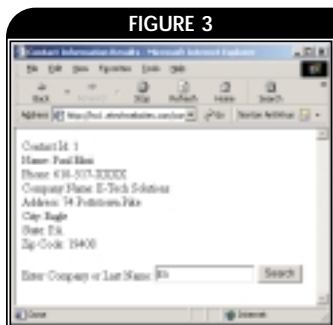
www.developer.bea.com

BEA

www.developer.bea.com



Internet phone invoking same URL as Web browser shown in Figure 3



Web browser invoking same URL as Internet phone shown in Figure 2

SECURITY

- Supports security and transactions via standard J2EE mechanisms.
- HTTP-authenticated identity can be passed to EJBs. WLS 6.1 Web services supports J2EE roles via Web application and EJB deployment descriptors.

CONTACT MANAGEMENT APPLICATION EXAMPLE

To illustrate the process of building and deploying a Web service with WebLogic 6.1, we took an existing J2EE application and exposed it as a SOAP Web service. The application is a Contact Management Application that returns a list of matching contacts via a last name or company name that is entered by the user. The application was built as a multi-channel Model – View – Controller application with the ability to dynamically render both the input form and the results in WML or HTML through XSLT. Accessing the same URL via either the Web or an Internet phone invokes the same back-end EJB that performs the search and returns the results. The WML and HTML interfaces access a servlet that formats the data through an appropriate XSL and returns it.

Figures 2 and 3 are examples of an Internet phone and Web browser both invoking the same URL

We also wanted to add to the existing architecture a way for other servers to access our EJB so that they could use the services in ways that are incorporated into business processes in new and different ways.

By taking our business logic of retrieving and returning Contacts that are contained in an EJB and exposing it as a Web service, we allow any SOAP client to access that service. Figure 4 is an example of a Visual Basic application that invokes the service, passing it a partial last name and having the EJB, through SOAP, return that data!

The architecture of the SOAP component is illustrated by the diagram in Figure 5. The SOAP Client first invokes a session on the SOAP Server by accessing the WSDL. This returns information about the SOAP service back to the client so that it knows how to call the methods contained in the service. The method calls are then made via a Servlet that provides XML Parsing and SOAP encoding/decoding that interfaces with our EJB.

Regardless of the API used to fetch the service proxy and invoke the service operations, the proxy or some delegated object is responsible for marshalling the arguments to XML fragments using the SOAP encoding rules (or some other custom encoding format). The proxy or some delegated object is also responsible for constructing the SOAP document and transmitting the same using a configured transport.

The transport infrastructure delivers the message to the SOAP processor on the J2EE server. The processor maps the request to a stateless EJB, converts the message arguments to Java types using the SOAP decoding rules (or custom decoders provided by the application), and invokes the EJB. The SOAP processor formulates a SOAP message response using the returned result and relays it back to the client as an HTTP response.

WebLogic SOAP Server

When you build a WebLogic Web service several decisions must be made. First, you must decide whether the service will be Message-style or RPC-style. Message-style services are asynchronous in nature and usually data driven. Think of it as posting some data to a service without requiring an immediate response. In WebLogic, these services are implemented as JMS listeners with a publish/subscribe messaging model. RPC-style services use a more interactive model, with clients directly invoking process-oriented methods and getting an immediate response. In WebLogic, these services are implemented as stateless session EJBs. Our example uses an RPC-style service.

The process of implementing and deploying a SOAP service on WebLogic Server is rather simple. First you implement a stateless session EJB that exposes the methods that you want to make accessible via SOAP. Remember to stick to data types that WebLogic's SOAP implementation supports for parameters and return types of the EJB's methods (see <http://e-docs.bea.com/wls/docs61/WebServices/develop.html#1038901> for details).

A key design constraint to consider is to have a single deployment that allows clients to access the EJB directly and as a SOAP Web service. The direct EJB call will be accessed by internal clients and the SOAP service will be accessed by external clients. This creates a best-of-both-worlds scenario for WebLogic development – a single set of EJB source code with a single deployment that gives high performance direct-EJB access and open SOAP XML access at the same time!

To build contracts.ear, the Ant utility (<http://jakarta.apache.org>) must be invoked with the projects build file. After you have created the EJBs and are ready for deployment, you may compile it in several different ways. The simplest method is to utilize the Jakarta project's build tool Ant and the custom tasks provided with WebLogic server for compiling and packaging EJBs in a suitable way for the WebLogic environment. There is also a helpful task provided for creating the J2EE application EAR file to represent the SOAP service.

To build contacts.ear, Ant must be invoked with the project's build file. This will cause the EJBs to be compiled (if necessary), packaged, compiled into a JAR usable by WebLogic, then added into an enterprise application to be invoked as a SOAP service. The ws-gen task handles a number of steps that are required to assemble the EAR file. Among them is the generation of the WSDL for this service based on the EJB's class definition through the use of WebLogic utility classes. It is possible to assemble the application by hand, too. For more information about this, see <http://e-docs.bea.com/wls/docs61/WebServices/package.html#1001373>.

The example application also incorporates a Web application front-end that allows other clients to utilize the EJB. This part of the application can currently respond to HTML and WML clients, and can have other types of clients added fairly easily

because the output from the JSPs is driven by XSL. The requested JSP determines which XSLT file to use based upon what it can determine about the client making the request. The default response is to return HTML, but if the page senses that the client is a WML browser, the output is WML. This is implemented using the WebLogic XSL custom tag that utilizes the Apache Project's Xalan XSL Transformation engine. The instructions found at http://e-docs.bea.com/wls/docs61/xml/xml_apps.html#1079383 were followed to utilize the custom tag. The build process also uses WebLogic's JSP compiler from within Ant to make certain that the JSP files will compile correctly in the WebLogic environment, saving time by allowing for the correction of JSP errors prior to deployment time.

The process of integrating this portion of the application with the Web service contacts.ear file may be automated in the Ant build process. The basic idea is to expand contacts.ear and web-services.war, which is created when contacts.ear is expanded. Copy any Web application resources except for web.xml into the directory where web-services.xml was expanded. Next, patch the web.xml file that is created from expanding web-services.xml with what is required for this portion of the application, namely JNDI information for looking up the EJB. Create a Web application based upon this new directory structure and create a new contacts.ear file that includes this Web application and the EJB jar file.

Creating and using the patch file requires the standard Unix development tools diff and patch. Windows does not have these tools, so see cygwin.com to install them. To create the patch file, make a minimum web.xml file (only has empty web-app tag) and use diff to display the differences between the application's web.xml and the blank one. The command should be similar to `<diff -C 2 blank-web.xml web.xml > web.xml.patch` to create the patch file. This patch file is then used from within build.xml to patch the web.xml file created by wsgen, preserving all of the elements in the wsgen Web.xml and adding the elements that are necessary for the rest of the Web application.

Deploy the application as you would any enterprise application into WebLogic server. If all goes well, the application should be accessible at the URL/contacts. From there, the client.jar file can be downloaded for use in creating a WebLogic SOAP client to access the service. Now a client can be built to access this service. The JSP-based client should be accessible at /contacts/list from HTML and WML clients.

The SOAP Clients

To illustrate how various clients can interface with our SOAP Server, we have provided examples of four different clients and descriptions of how each accesses the SOAP Server.

APACHE SOAP IMPLEMENTATION

There are currently two versions of Apache SOAP in development: the current stable version, Apache

SOAP 2.2, and the next generation of Apache SOAP, known as Axis, now in alpha. Neither of these clients are WSDL aware, so using them to access SOAP services in a WebLogic server requires some finesse.

The first step is to locate the actual URI of the service. This can be found by looking for the service element in the WSDL and getting the value for the location attribute of the nested soap:address element. This value is the endpoint to use in the client. Since the Apache clients do not recognize WSDL, they see the WebLogic SOAP service as an untyped server, meaning no xsi:type attribute is provided in the returned XML. You must tell the client what data type to expect from the method call so that it may return the correct data type in the code. Doing this in the Axis client is slightly easier than in the version 2.2 client. Finally, any JavaBeans or other complex data types need to have serializers and deserializers registered for them, depending upon whether the complex type is a parameter or a return type. In the example application, you must register the JavaBean com.etechsolutions.contact.ContactInfo and java.lang.Array deserializers provided with the client classes. At this point, the request can successfully be made to the SOAP service to execute the desired method. Information about the Apache SOAP implementations can be found at <http://xml.apache.org>.

BEA Java Client

WebLogic includes a client-side SOAP API that can be used to develop SOAP Clients that can access SOAP Servers of any type – Microsoft, Apache, etc. The WebLogic SOAP Client supports the calling of Web services that contain a WSDL as well as those that do not support the use of a WSDL.

Accessing a SOAP service with a BEA SOAP client using WSDL is a lot like accessing an EJB. The client uses a javax.naming.InitialContext object with a particular java.util.Properties object to get an instance of an object that represents the service. In the case of the sample application, the object is an instance of the EJB object. Then you may call whatever method the SOAP service exposes.

When you invoke a Web service via the WebLogic SOAP Client, you can use either a static approach or a dynamic approach. The static method accesses the EJB interface directly and is, therefore, the most type-safe approach. This is the approach we used in our example.

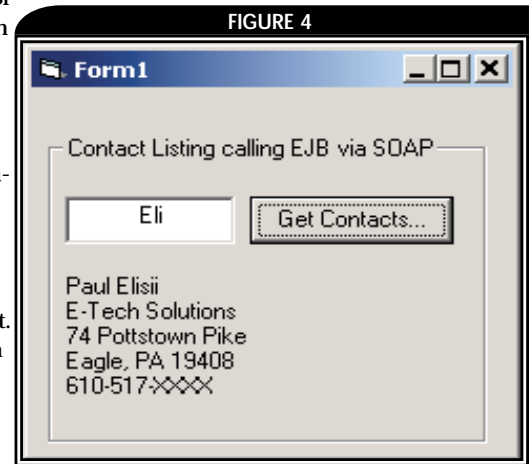


FIGURE 4 Example of a Visual Basic application that invokes the service

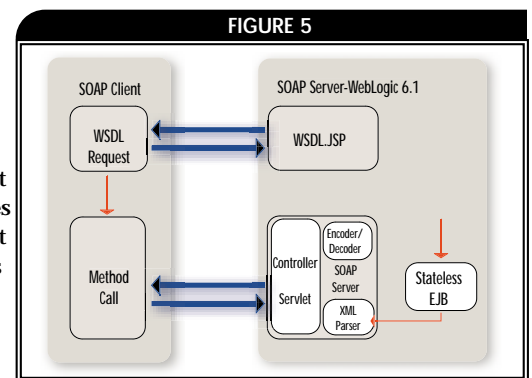


FIGURE 5 Architecture of the SOAP component

First, we set up an INITIAL_CONTEXT_FACTORY that allows us to load the service-specific information into the listing object, which is then used to make the direct method calls against the SOAP service which invokes the EJB (see Listing 4).

The dynamic approach is very similar, except that the client does not explicitly reference the EJB interface in its calls. Instead, it goes through a WebserviceProxy.

WSDL files make Web services easier and better to use; however, there are occasions where Web services that you need to call do not support the WSDL interface. In these cases, you will need to access the service without a WSDL. The process of invoking non-WSDL Web services includes:

- Creating a factory for SOAP encoding
- Connecting to the Web service
- Getting the send method of the service
- Sending the SOAP message
- Processing the SOAP response (if any)

WebLogic provides all of the relevant Java classes to enable SOAP Clients to carry on SOAP Interactions with non-WSDL Web services.

PERL IMPLEMENTATION

The Perl client implemented in the example application uses the SOAP::Lite module found at www.soaplite.com. This implementation treats complex data types such as JavaBeans as associative arrays for ease of use. The array of com.etechsolutions.contact.ContactInfo objects returned in Java code is seen as an array of associative arrays using the SOAP::Lite module. The only information that needs to be extrapolated from the WSDL is the namespace attribute of the soap:body elements nested in the input and output elements of the operation elements used to represent the methods in the service. That value must be supplied to the SOAP::Lite client as the URI to use to access the service.

MICROSOFT IMPLEMENTATION

The Microsoft Implementation uses the Microsoft SOAP Toolkit 2.0, which can be downloaded from <http://msdn.microsoft.com/>

library/default.asp?url=/nhp/Default.asp?contentid=28000523. The SOAP Toolkit is a set of libraries that allow you to create a connection to any SOAP server and invoke methods contained there.

To create a Microsoft VB SOAP Client, you first need to include the "Microsoft SOAP Type Library" and the "Microsoft XML, v3.0" libraries in your project. These are both part of Microsoft SOAP Toolkit 2.0. The actual code required to use SOAP Objects is quite small. The first thing you must do is instantiate a SOAP Client Object.

```
set soapclient = CreateObject("MSSOAP.SoapClient")
```

The next step is to connect that SOAP Client object to a SOAP WSDL. In this case, we are connecting to the BEA WebLogic-generated WSDL.JSP.

```
call  
soapclient.mssoapinit("http://ets_lnx1:7001/contactlistings/wsd1.jsp")
```

The final step is to invoke methods on the SOAP Server. Because we connected to the WSDL, the compiler is able to check for syntax errors if nonexistent methods are called, or are called with the wrong arguments.

```
set Contacts = Soapclient.getMatchingContacts( "Elisii")
```

In our example, the data being returned from the call is a complex data type; it is an array of objects. In order to handle this properly from VB, we must declare the return type as IXMLDOMNodeList. This allows us to manipulate the nodes of the objects returned as follows:

```
Dim Node1 As IXMLDOMNodeList  
Set Node1 = Contacts(0)  
output.Caption = Node1(6).Text 'firstName
```

Summary

Web services can greatly enhance a business's ability to create a more efficient way for suppliers and customers to access their products and

RECEIVE \$150
DISCOUNT OFF FULL CONFERENCE
WEB SERVICES EDGE REGISTRATION



**Learn How to Develop
SOAP Web Services NOW!**
at a One-Day Tutorial... Coming to a City Near You!

services and to extend their reach to a much wider range of potential customers. Web services allow a customer using any type of technology to reach a business's Web services through HTTP and XML. SOAP and its related technologies, WSDL and UDDI, allow for a more robust way to create and invoke Web services. The SOAP protocols allow clients to make structured Remote Procedure Calls against a server;

WSDL allows for the discovery of the SOAP Server details; and UDDI allows for the creation and promotion of the Web service on private or public registries.

Web Logic 6.1 provides a great platform for deploying existing or new EJBs as SOAP Servers. It is compliant with the industry-standard SOAP implementation, which allows it to be called from any client, including Microsoft's SOAP Client.

Listing 1

```
- <types>
- <schema targetNamespace="java:com.
  etechsolutions.contact"
  xmlns="http://www.w3.org/1999/XMLSchema"
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
  xmlns:soap="http://schemas.xmlsoap.org/soap/
  encoding/" xmlns:contact="java:com.
  etechsolutions.contact">
+ <complexType name="ArrayOfContactInfo">
- <complexType name="ContactInfo">
  <attribute name="zipCode" type="string" />
  <attribute name="clientId" type="int" />
  <attribute name="state" type="string" />
  <attribute name="city" type="string" />
  <attribute name="phone" type="string" />
  <attribute name="address" type="string" />
  <attribute name="firstName" type="string" />
  <attribute name="lastName" type="string" />
  <attribute name="companyName" type="string"
  />
/>
</complexType>
</schema>
</types>
```

Listing 2

```
- <message name="getContactRequest">
  <part name="arg0" type="math:BigDecimal" />
</message>
- <message name="getContactResponse">
```

```
<part name="return"
  type="contact:ContactInfo" />
</message>
+ <message name="getContactsMatchingRequest">
+ <message name="getContactsMatchingResponse">
  <message name="getAllContactsRequest" />
+ <message name="getAllContactsResponse">
```

Listing 3

```
- <binding name="ContactListingEJBBinding"
  type="tns:ContactListingEJBPortType">
  <soap:binding style="rpc" transport=
    "http://schemas.xmlsoap.org/soap/http" />
- <operation name="getContact">
  <soap:operation soapAction="urn:getContact" />
- <input>
  <soap:body use="encoded" namespace=
    "urn:ContactListingEJB" encodingStyle=
    "http://schemas.xmlsoap.org/soap/encoding/" />
</input>
- <output>
  <soap:body use="encoded" namespace=
    "urn:ContactListingEJB"
  encodingStyle="http://schemas.xmlsoap.org/soap/
  encoding/" />
</output>
</operation>
```

Listing 4

```
// Set up properties for InitialContext
```

```
Properties props = new Properties();
props.put(Context.INITIAL_CONTEXT_FACTORY,
  "Weblogic.soap.http.SoapInitialContextFactory");
props.put("Weblogic.soap.wsdl.interface",
  ContactListingEJB.class.getName());
if(debug) props.put("Weblogic.soap.verbose",
  "true");
```

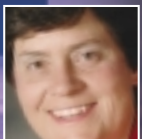
```
// Set up InitialContext and get a handle on
the ContactListingEJB
Context context = new
  InitialContext(props);
ContactListingEJB listing =
  (ContactListingEJB)context.lookup(endpoint);

// Decide which method to call based upon
command line search term
ContactInfo[] contacts = null;
if((contactName != null) &&
  (contactName.length() > 0))
  contacts =
  listing.getContactsMatching(contactName);
else
  contacts = listing.getAllContacts();
```

Jump-start your Web Services knowledge Get ready for Web Services Edge East and West!

AIMED AT THE JAVA DEVELOPER COMMUNITY AND DESIGNED TO EQUIP ATTENDEES WITH ALL THE TOOLS AND INFORMATION TO BEGIN IMMEDIATELY CREATING, DEPLOYING, AND USING WEB SERVICES.

EXPERT PRACTITIONERS TAKING AN APPLIED APPROACH WILL PRESENT TOPICS INCLUDING BASE TECHNOLOGIES SUCH AS SOAP, WSDL, UDDI, AND XML, AND MORE ADVANCED ISSUES SUCH AS SECURITY, EXPOSING LEGACY SYSTEMS, AND REMOTE REFERENCES.



PRESENTERS...

Anne Thomas Manes, Systinet CTO, is a widely recognized industry expert who has published extensively on Web Services and service-based computing. She is a participant on standards development efforts at JCP, W3C, and UDDI, and was recently listed among the Power 100 IT Leaders by Enterprise Systems, which praised her "uncanny ability to apply technology to create new solutions."



Zdenek Svoboda is a Lead Architect for Systinet's WASP Web Services platform and has worked for various companies designing and developing Java and XML-based products.

EXCLUSIVELY SPONSORED BY



BOSTON, MA (Boston Marriott Newton) **JANUARY 29**
WASHINGTON, DC (Tysons Corner Marriott)..... **FEBRUARY 26**
NEW YORK, NY (Doubletree Guest Suites)..... **MARCH 19**
SAN FRANCISCO, CA (Marriott San Francisco)..... **APRIL 23**

REGISTER WITH A COLLEAGUE AND SAVE 15% OFF THE \$495 REGISTRATION FEE.

Register at www.sys-con.com or Call 201 802-3069





INTERNATIONAL WEB SERVICES CONFERENCE & EXPO



INTERNATIONAL JAVA DEVELOPER CONFERENCE & EXPO



INTERNATIONAL XML CONFERENCE & EXPO



FUNDAMENTALLY IMPROVING THE SPEED, COST AND FLEXIBILITY OF BUSINESS INTEGRATION

Sponsored by:



Sponsored by:



Produced by:



Web Services – Skills, Strategy, and Vision

For the developer, the latest tools and techniques...

For the architect, the latest designs....

For the VP/CIO, technology management issues

- Invaluable information in the form of discussions, presentations, tutorials, and case studies
- Unmatched Keynotes and Faculty - gurus in the Java, XML, .NET, and Web Services world
- The largest independent Web Services, Java, and XML Expos
- An unparalleled opportunity to network with over 5,000 *i*-technology professionals

New in 2002!

Each track will feature "Hot Breaking" Sessions to keep attendees up-to-the-minute on Emerging Technologies and Strategies!

Featuring 2 Keynote Panels and Industry Perspectives from the Visionaries shaping Next Generation Technologies and Business Strategies

2002

ONLINE EARLY BIRD REGISTRATION NOW OPEN! SAVE \$400

JUNE 24-27
JACOB JAVITS
CONVENTION
CENTER
NEW YORK, NY

OCTOBER 1-3
SAN JOSE
CONVENTION
CENTER
SAN JOSE, CA

FOR MORE INFORMATION

SYS-CON EVENTS, INC
135 CHESTNUT RIDGE RD.
MONTVALE, NJ 07645

201-802-3069
WWW.SYS-CON.COM

Who Should Attend...

- Developers, Programmers, Engineers
- Senior Business Management
- Senior IT/IS Management
- i-Technology Professionals
- Analysts, Consultants

Who Should Exhibit...

Java, XML, Web Services, and .NET Technology vendors, staking their claim to this fast-evolving marketplace

Web Services Edge Conference Committee



Alan Williamson
Java Track Chair
Editor-in-Chief
Java Developer's Journal

Java Track

- Java 1.4: What's New?
- Building Truly Portable J2EE Applications
- J2ME: MIDlets for the masses
- WebScripting Technologies: JSP/CFML/Velocity
- Where is Swing in this New Web Services World?



Ajit Sagar
XML Track Chair
Editor-in-Chief
XML-Journal

XML Track

- XML Web Services: Standards Update
- Using XML for Rapid Application Development and Deployment with Web Services
- Unlocking the Promise of XML: From Hype to How-To
- The Use of XML Technologies to Enhance Security
- Content Management with XML



Sean Rhody
Conference Tech Chair
Web Services Track Chair
Editor-in-Chief
Web Services Journal

Web Services Track

- Starting Out in Web Services: Fundamentals of Web Services (WSDL, UDDI, SOAP)
- Exploring .NET myServices Initiative
- Standards Watch: Reviews and Discussions of the Interactions of All the Relevant Standards
- Guarding the Castle: Security and Web Services
- The Microsoft Way: .NET, Passport, and other MS Technologies for Web Services
- Laying Down the Rules: Web Services and Business Process Engines
- Practical Experiences with Web Services and J2EE
- The Odd Couple: Making .NET and J2EE Work Together



Andy Astor
IT Strategy Track Chair
International Advisory Board
Web Services Journal

IT Strategy Track

- Key Architectural Issues in a World of Web Services
- .NET vs J2EE – From Religious Wars to Fact-Based Decision Making
- Getting Started with Web Services
- Selecting a Framework: Toolkit, Platform, or Roll Your Own?
- Infrastructure Vendors—A Map of the World
- Extreme Programming and Web Services Projects: Defining ROI
- Standards You Need to Know About
- Best Practices You Need to Insist On
- Introduction to Business Rules and Rules Engines
- Panel Discussion- Web Service Business/ Economic Models



Jim Milbery
Vendor Track Chair
Product Review Editor
Java Developer's Journal

Vendor Track

- .NET, J2EE, JMS and other Messaging
- Case and Development Tools
- The Use of Testing Tools
- How-to Demonstrations



The Problem:

We have a J2EE application where clients with the same userid need to share the session data that is maintained on the server side. In addition, clients aren't just Web clients, but applets and applications as well. How should we implement that?

Shared Sessions USING EJBS

BY MIKA RINNE

The Solution:

This is a typical problem when writing J2EE applications, since the J2EE APIs don't provide an out-of-the-box solution for this.

The Servlet API typically used with Web clients provides nice ways to manipulate the client's session and the data related to it, but doesn't offer a direct means of sharing the session data between clients (for security reasons). It's also somewhat limited to Web clients only.

Enterprise Java Beans (EJBs), on the other hand, could be used to implement this quite easily, since they work with any type of Java clients. It may be a bit heavyweight, but programmatically it's quite straightforward, so let's take a closer look at it.

A stateless Session EJB easily comes to mind, since it enables the client's state on the server-side. Session EJBs (or instances of them) are also sharable between client sessions by passing the reference to them (or actually into their remote interface) within the application, like a servlet. In order to pass references between sessions, the application needs to explicitly maintain a look-up table of the references for each client identified, so it knows to relate the user (along with userid) with the proper instance of a Session EJB when that already exists.

However, that's pretty much what WLS does when Entity EJBs are used – or at least there is a lot of similarity in this scenario. It keeps the reference to each Entity EJB instantiated in the memory and then passes it (or the reference to the EJB's Remote Interface of

it) to the client, which requests it via the bean's `findByPrimaryKey` method of the Home Interface. If the Entity EJB is not found since it does not yet exist, it can be created on the fly to be related with the client and its session the first time the client logs in. Therefore, Entity EJB instances are easily shared, even between clients not running in the same JVM.

Note: an Entity EJB instance does not always refer to a row in the database; it can be totally virtual. But since Entity Beans are especially good at accessing persistent data, it gives us a chance to provide persistency for our sessions (and the data in them) in case of application crashes, etc.

To locate the session references for clients, use of the `findByPrimaryKey` method is logical, since the primary key is actually the client's userid for the application. The only thing that's left for the application to do is the appropriate authentication of clients, before they can access the session data via the bean's `findByPrimaryKey` method.

Creating the Sample Application

Now we're going to design the sample application. First, we divide the application into four parts:

1. **UserSession:** Class containing the user's session data in Hashtable plus some other information like the user's `userid`. It utilizes the `Serializer` class for session data persistency (see below). `UserSession` class can be used locally at first, and then remotely via the Entity EJB "wrapper" to be easily shared with multiple clients of many types (JSP and Java clients, etc.).
2. **UserSessionSerializer:** Class for session data persistency. Java serialization is used instead of JDBC in the first step to keep this as simple as possible. Later on, we can replace this with pool-oriented JDBC-based persistency for distributed larger-scale applications. There's always one file on the disk per session – even if it's shared by multiple clients – and an EJB instance accessing it.
3. **EJB wrapper:** For distributed `UserSession` objects. It's subclassed from the `UserSession`. As an Entity EJB, it consists of three separate classes in the `sharedSessionSample.remote` package (others are stored in the package "local").
4. **Sample JSP client** using `SessionData` objects, first locally and then remotely

Next, we create a package named `sharedSessionSample.local`, and place two classes, the `UserSession` class and the `UserSessionSerializer` class, in it.



AUTHOR BIO...

Mika Rinne is a senior consultant with BEA Systems Inc. He has been programming since he was 13 and built an Internet and BEA TUXEDO-based on-line ticket selling service, the first of its kind in Scandinavia in 1995.

CONTACT...

mika.rinne@kolumbus.fi

The `UserSession` class has three class members (variables) and methods:

Members:

```
protected String path;
protected String userId;
protected Hashtable sessionData;
```

Methods (the throw definitions are not shown):

```
public void create(String userId)
public void load(String userId)
public void save()
public void remove()
public Object getValue(String name)
public void setValue(String name, Object object)
public void removeValue(String name)
public Hashtable getValues()
public void setValues(Hashtable values)
```

The class member `path` defines the directory where the serialized sessions are stored on the disk and loaded from. Typically, the path is the same for all sessions. The `userId` defines the owner of the session and is used to name the file when serialized (that is, a session of the user "John" will be serialized as a file named "John.session" onto the disk into the specified directory). The `Hashtable sessionData` holds the session named values. *Note:* we don't actually store the whole `UserSession` when serialized, just the `sessionData` member object.

Getters and Setters

Methods are constructed so they can easily be extended to work as an Entity EJB. The `create()` method is used to create a new, empty session, which can be saved by `save()` method after one or more values have been set to the session via the setters `setValue()` or `setValues()`. The difference between these setters is obvious: the first one sets one named value at the time, whereas the latter sets all the values for the session (passed as a `Hashtable`). It is the same with getters `getValue()` and `getValues()`. The method `load()` is used to load an existing session and its data from the disk, and throws an exception if not found (in that case it should probably be created), as opposed to the method `create()`, which throws an exception when the session already exists.

The setters do not `save()` automatically; one has to be called explicitly by the client. However, that's implemented to the Entity EJB.

One thing to note is that the `UserSession` is implemented in such a way that `getValues()` and `setValues()` work "by value" instead of "by reference." It's the preferred approach here, to avoid unintentional changes to the session data on JSPs (I've seen it happen many times with objects like a `Hashtable`). The actual code is shown in Listing 1 (the code for this article may be found online at www.sys-con.com/weblogic/sourcecec.cfm).

The methods `create()`, `load()`, and `save()` use the `SessionSerializer` class, which has no member variables, just static methods:

```
public static void serialize(String path, String
userId,
Hashtable sessionData)
public static Hashtable deserialize(String path,
String userId)
public static void remove(String path, String
userId)
```

The `serialize()` method is used to save (using Java serialization) the `Hashtable` containing the named session values onto the disk. `Deserialize()` is the opposite to `serialize()` in order to instantiate the `Hashtable` object from existing session values on the disk. The `remove()` method can delete serialized session values from the disk.

Shown below are examples of the implementation of `SessionSerializer`'s `serialize()` method and the `UserSession`'s `save()` method to show how the use of the `SessionSerializer` has been implemented.

The `SessionSerializer`'s `serialize` method:

```
public static void serialize(String path,
String userId,
Hashtable sessionData) throws Exception
{
    FileOutputStream ostream =
new FileOutputStream(path + "/" + userId + ".session");
ObjectOutputStream p = new ObjectOutputStream(ostream);
p.writeObject(sessionData);
ostream.close();
}
```

The `UserSession`'s `save()` method:

```
// Note that the path needs to be set first,
before calling save
public void save() throws Exception
{
    UserSessionSerializer.serialize(path, userId,
sessionData);
}
```

Next, the files are compiled into Java classes. Once we get the WLS running and these files copied into the classpath, the sample JSP page can be created to test accessing session values locally (that is, within the same JVM) as the first part of the sample. It uses the `setValue()` method for incrementing an integer value "i" for the user; if a user isn't specified in the HTTP request it is considered to be "JohnDoe." There's no authentication, which in real life would be needed (the standard Web app definition can be used; see Listing 2).



Alternatively, the method `setValues()` (instead of `setValue()`) can be used to increment the value for "i" as shown below:

```
int i = 0;
Hashtable values = userSession.getValues();
if(values.get("i") != null)
{
    i = ((Integer) values.get("i")).intValue();
    i++;
}
values.put("i", new Integer(i));
userSession.setValues(values);
```

Conversion

Now that the classes for accessing the session values locally have been implemented and tested, they're prepared for remote use by both local and remote clients by being converted into an Entity EJB. That also enables easy sharing of the session data between clients, since the Entity EJB takes care of synchronization between client method calls. In addition, using bean-managed persistence (BMP) and Java serialization, we're able to serialize any named values in the `Hashtable` without having to worry about JDBC datatypes and so on in our EJB.

However, normal rules apply when writing (updating) the same session values from two or more clients simultaneously; there's no way that the EJB could prevent overwriting of data in those cases. For that reason, make sure to add an identifier (e.g., a transaction id or timestamp from the file system, etc.) that's sent between the client and the server to indicate if the session data has been updated elsewhere (another client) when doing updates from multiple clients to the same session and its data.

In order to enable the `local.UserSession` for remote use, three EJB classes are implemented:

- ***UserSessionHome***: the home interface of the entity EJB
- ***UserSession***: the remote interface of the entity EJB
- ***UserSessionBean***: the bean implementation class that is subclassed from the `UserSession` of the local package

All classes are placed in a package named `sharedSessionSample.remote` to differentiate them from the local classes (partly with same name).

The `UserSessionHome` interface defines two methods to create new and locate existing sessions remotely:

```
public UserSession create(String userId)
public UserSession findByPrimaryKey(String userId)
```

Since the `UserSessionBean` (which we'll see in a minute) is subclassed from the `local.UserSession`,

the `remote.UserSession` remote interface is used to publish the methods of the `local.UserSession` for remote use (the throw definitions are not shown):

```
public Object getValue(String name)
public void setValue(String name, Object object)

public void removeValue(String name)
public Hashtable getValues()
public void setValues(Hashtable values)
```

The EJB implementation class `remote.UserSessionBean` has only one member variable – along with those that are inherited from the `local.UserSession` class – and the following methods:

Members:

```
private transient EntityContext ctx;
```

Methods (some standard EJB methods and throw definitions are not shown):

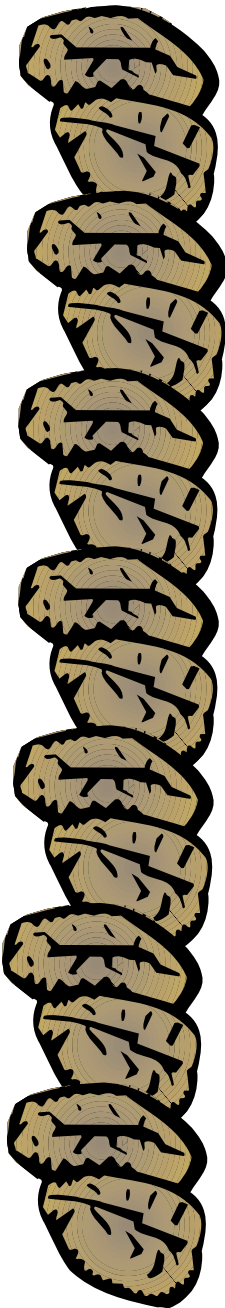
```
public String ejbCreate(String userId)

public String ejbFindByPrimaryKey(String userId)
public void ejbLoad()
public void ejbStore()
public void ejbRemove()
protected void setPath()
```

The `ejbCreate()` method is used to create a new session data object in case it doesn't exist yet. This object is created by calling the `create()` method of itself inherited from the `local.UserSession` class. If the creation succeeds (that is, no session exists already with the given `userId`), the `userId` is returned and the EJB is instantiated by the WLS for shared and remote use. If the session already exists, a standard `DuplicateKeyException` is thrown (see Listing 3).

The `setPath()` method is used to read the serialization directory path from EJBs' properties and is implemented as follows:

```
try
{
    InitialContext ic = new InitialContext();
    setPath((String)
ic.lookup("java:/comp/env/storagePath"));
}
catch (NamingException ne)
{
    // defaults to '/sessions' directory under
WebLogic
// root dir, if not defined. For clustering, use a
shared dir.
    setPath("./sessions");
}
```



Note that in the case of clustering, which is a nice feature of WebLogic and especially typical of high load sites, the directory path should be a shared resource for all application server instances – otherwise, the clustering wouldn't work properly.



Load and Store Methods

The opposite of `ejbCreate()`, the `findByPrimaryKey()` is used to locate an existing session on the server. First, the method tries to find a serialized session on the disk by calling its own `load()` method, inherited from the local `UserSession` class, and throws `ObjectNotFoundException` if it is not found. However, if the session is found (the `load()` succeeds), it returns the `userId`, from which the server knows to instantiate the EJB by calling the `ejbLoad()` method implicitly, resulting in the loading of the actual session data from the disk (see Listing 4).

The same `load()` method is used for two purposes: first, for trying to locate an existing session from the disk when doing the `findByPrimaryKey()`, and then for loading (deserializing) the actual data when the EJB is instantiated by the server (the `ejbLoad()` method). This may not be the best practice, but it works (We could have a separate find method instead, but to limit the size of the code we omit that).

The `ejbStore()` method is called automatically by the server when any setter of the EJB is called. Therefore, there's no need for an explicit `save()` method in the EJB. We could easily add it for optimization reasons later on, but it's hardly needed here. Thus, the `ejbStore()` method is as follows:

```
try
{
    save();
}
catch(Exception e)
{
    throw new
EJBException(e.toString());
}
```

Finally, the `ejbRemove()` method, which can be used to delete existing session data:

```
try
{
    remove();
}
catch(Exception e)
{
    throw new EJBException(e.toString());
}
```

Now that the EJB is ready, it's time to modify the JSP for using the EJB instead of the local class. In Listing 5, the `UserSession` is accessed remotely to do basically the same increment as in Listing 2, but now it enables the client (a standalone Java application, for example) to run in a separate JVM from the server.

In order to set more than one value at a time, the `setValues()` method can be used, as in this example of incrementing:

```
int i = 0;
Hashtable values =
userSession.getValues();
if(values.get("i") != null)
{
    i = ((Integer)
values.get("i")).intValue();
    i++;
    values.put("i", new Integer(i));

    // Put other values as well and
    finally set them
    userSession.setValues(values);
}
```

For optimization, the JNDI lookup for the EJB could be done only once and stored in the WebLogic Server's memory with application scope (see Listing 6).

In addition, we could also cache the `UserSession` object itself to the user's HTTP session once found, in order to avoid subsequent `findByPrimaryKey()` calls in case of JSP. Of course, for Java clients like applets, the caching isn't needed (used as a local object). The code for caching is shown in Listing 7.



SUBSCRIBE AND SAVE

WebServices JOURNAL

Offer subject to change without notice

ANNUAL NEWSSTAND RATE	
\$93.88	
YOU PAY	
\$69.99	
YOU SAVE	
\$13.89	Off the Newsstand Rate

DON'T MISS AN ISSUE!

Receive 12 issues of *Web Services Journal* for only \$69.99! That's a savings of \$13.89 off the annual newsstand rate. Sign up online at www.sys-con.com or call 1 800 513-7111 and subscribe today!

In February *WSJ*:

Getting Greater Business Value from Web Services
Thinking of a wider audience

MS Gets It At Last!
Pleasant surprises from the Beta2 of .NET

The Largest Web Application in the World
Is this what .NET My Services will become?

Security and the .NET Framework
Grave danger awaits if your security isn't robust

Peopleclick and HR.NET
How can Web services and .NET work for developers today?



STRATEGIES & SOLUTIONS FOR UNWIRING THE ENTERPRISE

International Wireless Business & Technology Conference & Expo

Wireless Edge will provide the depth and breadth of education and product resources to allow companies to shape and implement their wireless strategy.

Developers, i-technology professionals and IT/IS management will eagerly attend.

Who Should Attend

Mobile & Wireless Application Professionals

who are driving their enterprises' wireless initiatives:

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

Plan to Exhibit

Provide the Resources To Implement Wireless Strategy

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

C O N F E R E N C E T R A C K S

TRACK 1: DEVELOPMENT

- WAP
- i-Mode
- Bluetooth / 802.11
- Short Messaging
- Interactive Gaming
- GPS / Location-Based
- Wireless Java
- XML & Wireless technologies

TRACK 2: CONNECTIVITY

- Smart Cards
- Wireless LANs incl. Bluetooth
- UMTS/3G Networks
- Satellite Broadband

TRACK 3: WIRELESS APPS

- Education
- Healthcare
- Entertainment
- Transport
- Financial Services
- Supply Chain Management

TRACK 4: HARDWARE

- Cell Phones/WorldPhones
- PDAs
- Headphones / Keyboards / Peripherals
- Transmitters / Base stations
- Tablets

TRACK 5: MANAGEMENT

- Wireless in Vertical Industries
- The WWW
- Unwired Management
- From 3W to 4W: Issues and Trends
- "Always-On" Management
- Exploiting the Bandwidth Edge
- Unplugged Valueware
- Wireless Sales & Marketing

**FOR EXHIBIT & SPONSORSHIP
INFORMATION PLEASE CALL
201 802-3004**



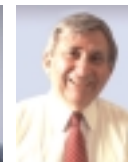
Arie Mazur
President, Chief Executive Officer, and Chairman, Slangsoft



Chris Bennett
Architect, Freedom Technologies



Daniel Elliott
Senior Vice President, Mobile Business, CompuCom



Douglas Lamont
Professor of Marketing, DePaul University



Kenneth Leung
Business Development Manager, Retail and Emerging Technologies, IBM



Keith McIntyre
Vice President and Chief Technologist, Stellcom



Steve Milroy
Wireless Technologist, Immediant



Tony Wasserman
Director, Mobile Middleware Lab, Hewlett-Packard Company

Exclusive Sponsorship Available

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



ONLINE REGISTRATION NOW OPEN

WWW.SYS-CON.COM/WIRELESSEGE2002

Or, To Register By Phone
Call: (201) 802-3069



	Development	Connectivity	Wireless Applications	Hardware	Management
M A Y 8 , 2 0 0 2					
8:30-9:45	Embedded Java (Bill Ray, Network 23 Limited)	A Real Time Model of (3G) UMTS Access Stratum (Niloy Mukherjee, MIT Media Laboratory)	Using Mobility to Streamline Traditional Business Processes (Dan Elliott, CompuCom)	Java Software Performance On Wireless Devices: Myths and Realities (Ron Stein, Nazomi)	LMDS - The Last Mile Enabler of Class 5 Revenues with VoIP (Ed Peters, Ensemble Communications, Inc.)
10:00-11:15	KEYNOTE				
11:30 A.M.	EXPO FLOOR OPEN				
LUNCH BREAK					
1:30-2:30	XML & Wireless Technologies (Karl Best, OASIS)	Securing Wireless Data Via Smart Card (Joseph Smith, New Dominion Software)	Collaboration for Wireless Warriors (Timothy Butler, SiteScape, Inc)	User Interactivity for Information Appliances (Arie Mazur, Slangsoft)	Leveraging Wireless In Customer Acquisition and Retention (Kenneth Leung, IBM)
2:45-3:45	KEYNOTE				
3:45-4:45	Developing Mobile Web Applications (Tony Wasserman, Hewlett-Packard Company)	In-building Wireless, the Next Frontier (Mary Jesse, RadioFrame Networks)	Turbocharge Mobile Applications with J2EE (Dr. Jeff Capone, Aligo, Inc.)	Cell Phones/ WorldPhones (Speaker TBA)	Mobilize Your Enterprise (Chris Bennett, Freedom Technologies)
5:00-6:00	VoiceXML Workshop (Bryan Michael, BeVocal)	Satellite Broadband (Speaker TBA)	Mobile Portals - The First Step Towards the Mobile Enterprise (Tony Wasserman, Hewlett-Packard Company)	PDAs (Speaker TBA)	Wireless Solution Return On Investment (ROI) In the Real World (Steve Milroy, Immedient)
M A Y 9 , 2 0 0 2					
8:30-9:45	Brew, I-Mode, WAP, and J2ME: How the Battlefield Is Shaping Up at the Start of the Mobile War (Reza B'Far, eBuilt, Inc.)	The Future of IP Mobility (Antti Eravaara, NetSeal, Inc.)	Multimodality: Revolutionizing Our Wireless Lifestyles (Arvind Rao, OnMobile Systems)	Tablets (Speaker TBA)	Marketing Value in Automotive Telematics Through Mobile Communications (Douglas Lamont, DePaul University)
10:00-11:15	KEYNOTE				
11:30 A.M.	Guide to Developing Applications with Micro Java (Kurt Baker, Kada Systems)	EXPO FLOOR OPEN			
LUNCH BREAK					
1:45-3:00	Building Secure Mobile Solutions (Keith McIntyre, Stelcom)	Smart Card Communications (Bill Ray, Network 23 Limited)	Developing New Applications Using VoiceXML (Jonathan Taylor, Voxeo)	PDAs (Speaker TBA)	Policies and Profiles: The Key to Mobile Data Services (Doug Somers, Bridgewater Systems)
3:15-4:30	Bluetooth™ Wireless Technology and Java™ Technology (Michael Portwood, Exuberance, LLC)	UMTS/3G Networks (Speaker TBA)	Instant Messaging and Wireless Computing Collide (JP Morgenthal, IKimbo)	Transmitters / Base Stations (Speaker TBA)	Total Business Solutions B2E (Speaker TBA)
4:45-6:00	Creating Carrier Optimized Wireless Internet Applications (David Young & Victor Brilon, Lutris)	Wireless LANs/Bluetooth (Speaker TBA)	Rapid Development of Successful Wireless Applications (Rod Montrose, AVIDWireless)	Headphones / Keyboards / Peripherals (Speaker TBA)	Wireless Sales & Marketing (Speaker TBA)

REGISTER BEFORE

March 1

And

Save

Up To

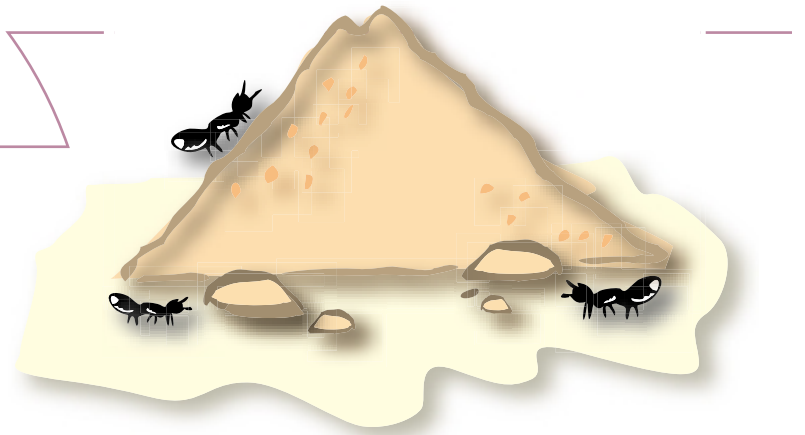
\$400

Plan to Attend the 3-DAY Conference

PRODUCED BY



May 7 will feature full-day tutorials. Consult www.sys-con.com/wirelessege2002 for up to-the-minute conference news.



Building and Deploying Applications With Ant

ANT CAN HELP DEVELOPERS AVOID COLLABORATION NIGHTMARES AND SAVE HOURS OF DEBUGGING

As a software developer I enjoy designing and developing new systems, but I don't enjoy spending time on configuration. Building and deploying WebLogic applications can quickly become complex.



BY
MICHAEL GENDELMAN

AUTHOR BIO...

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

CONTACT...

mgendelman@yahoo.com

Each deployed unit, whether it's an EJB JAR File, WAR File, or EAR File requires deployment descriptors to be placed in specific directories. The correct classes must be included, and in the correct directories.

Changes to a common class may require the redeployment of the entire application. Mistakes in the build process can require hours wasted debugging.

Use of a proprietary IDE, while able to automate the build process for a single developer, can become a nightmare when trying to deploy someone else's work. There are several Java IDEs on the market today that provide automation for building and deploying WLS applications. These tools usually store the build and deployment information in a proprietary format.

While it is possible to save this information and transfer it to other developers, it can present several problems. First, all of the developers need to work with the same tool. Second, some tools save all of the deployment information into a single file, making it difficult to synchronize to a single copy. Third, the tools don't usually allow for a high degree of flexibility, making it difficult to deploy to different servers, deploy multiple components, choose

between multiple deployment options, and deploy the components without using the IDE.

Build Tools

Ant is a build tool made specifically for Java. Build or make tools have been around for years, and they are used to automate the build process of applications. They usually run compilers, provide access to the file system, and handle a variety of other tasks associated with building an application. These tools are usually tied very closely to the operating system. They generally use a proprietary file format that is often hard to edit and easy to make mistakes with. While most make/build tools are capable of building Java applications, they do not provide operating system independence, nor do they closely integrate with Java.

Why Ant?

Ant was created as part of Apache's Jakarta Project. It has integrated Java tasks such as java, javac, JAR, WAR, and many others. It also supports many file system tasks, making them operating-system independent. The scripts you write on Windows will work equally well on Solaris. If Ant doesn't contain what you need, it can be extended using Java. And because Ant is extendable, it's being integrated with almost everything, including WLS.

I will demonstrate how to integrate Ant to automate the build and deployment process of WLS applications. There are many different ways to do this, and no one way is the best. Creating Ant scripts is an iterative process, requiring refactoring, just like good code. I will focus on using Ant to build and

deploy WLS applications. General information about Ant can be found at <http://jakarta.apache.org/ant>.

Ant doesn't replace the Java IDEs. Developers should still use whatever tools they are comfortable with to develop code and write deployment descriptors, but when a component is ready for integration into the project build, an Ant build.xml script should be included. The build.xml file is an XML file that describes the build and deployment process of your application or component.

With Ant, you still have to do all of the configuring, but then Ant automates the process. Ant allows you to define your packaging, build, and deployment processes in XML. The XML becomes a living documentation; everyone on the team can read, modify, and most importantly, execute the XML Ant scripts. The process can also be refined by updating the XML build scripts. Ant makes it possible to quickly perform complex and tedious operations without mistakes.

How to Get Started With Ant

Before we can start, we need to set up the environment. With WLS, this couldn't be easier. As of WLS 6.1, Ant is included in weblogic.jar. WLS now includes Ant build scripts with its example code. BEA includes an Ant bat and shell script in the WLS bin directory. In other words, by installing WLS 6.1, you've already installed Ant. If you're using an older version of WebLogic Server, you'll need to download Ant from <http://jakarta.apache.org/ant/manual/install.html>. Simply download the latest version, and unpack it. It will contain the Ant batch and script files in its bin directory.

Lets start by describing our example application. The application will contain two EJBs: EJBOne and EJBTwo. EJBOne and its helper classes are located in package mypackage.ejbone, EJBTwo and its helper classes are located in package mypackage.ejbtwo. The application will also contain servlets and JSPs. The servlet and JSP helper classes will reside in the package mypackage.client and the JSPs will reside in a directory called content, along with HTML and image resources. Here I detail the layout of the sample application directory structure:

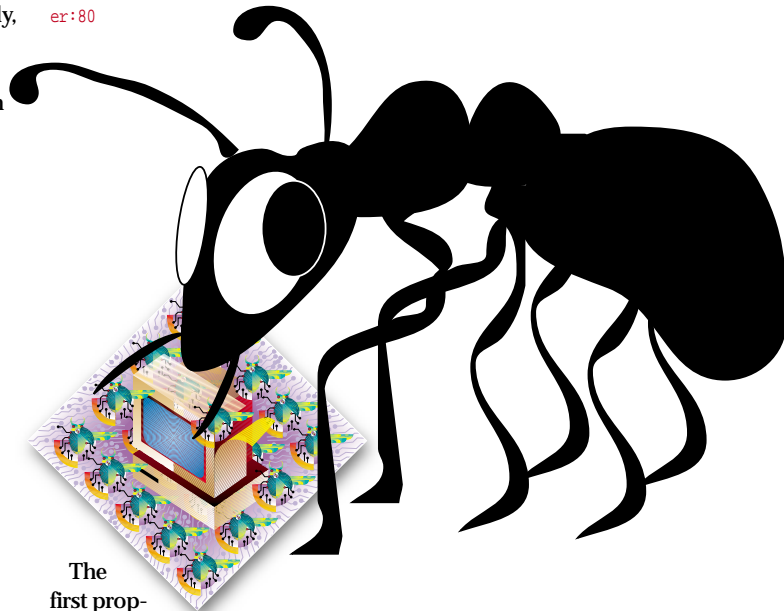
```
project/src/content
project/src/mypackage/client
project/src/mypackage/common
project/src/mypackage/ejbone
project/src/mypackage/ejbone/notneeded
project/src/mypackage/ejbone/META-INF
project/src/mypackage/ejbtwo
project/src/mypackage/ejbtwo/META-INF
project/src/ear/META-INF
project/staging
project/build
```

Using Ant With WLS

PROJECT PROPERTIES

The first thing I do, although it's not required, is define a property file. The property file contains information that may change between different development environments within a given project. The following is an example of a property file. The file could be named anything, but for this example we will call it config.properties.

```
WL_HOME=c:/bea/wlserver6.1
STAGING=c:/project/staging
BUILD=c:/project/build
SOURCE=c:/project/src
DEVELOPMENTSERVER= http://localhost:7001
TESTSERVER=http://testserver:80
```



The first property in the file is WL_HOME. WL_HOME is important for several reasons. First, each development environment may have WLS installed in different locations. Second, as you migrate to newer versions of WLS, you can control which version of WLS is used during the build process. The next property is STAGING. It contains the location used to store all of the components (e.g., JAR, EAR, WAR) before they are deployed. BUILD is used to store all of the compiled files and temporary files.

DEVELOPMENTSERVER and TESTSERVER define the URLs for the different servers. You could place this information in each build.xml file, but making changes could become very tedious. This would also not offer the degree of flexibility of the property file. Each developer can have a custom property file. The developers may use different development servers, or even be using different operating systems. The information could also be placed in the ant.bat or ant.sh file.

Each developer could have their own copy, but they would not have the ability to customize for



each application. The values could also be specified on the command line, but that would be quite a bit of typing. The basic reasoning behind the property file is to allow us to customize based on differences in development environments and differences in application environments.

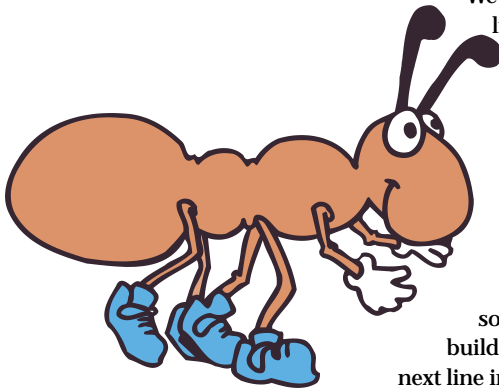
Building Components

The next step is to start building components. Let's start by building a JAR file for EJBOne. The first line of this example defines some basic information about the script.

```
<project name="EJBOne" default="deploy" basedir=".">
<property file="../../config.properties"/>
```

The default target is the target that will be run if one is not specified on the command line.

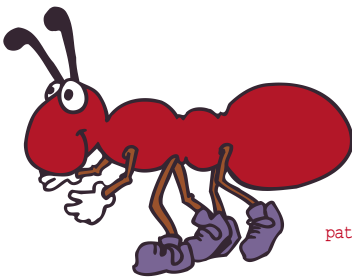
We will discuss targets in greater detail a little later. The basedir defines the directory where all of the commands will execute. In this case, I placed the XML build script in the base directory. I like to do this so each component has its own build script. I check the build script into version control with the component's source code. Any developer can take the component source code, run the Ant script, and build and deploy the component. The next line includes the property file we defined in Listing 1. We will use most of these properties throughout the example.



Compiling

A target defines a unit of work. It can be specified from the command line or defined as a dependency of another target. The following code defines the target compile_ejbone:

```
<target name="compile_ejbone">
  <mkdir dir="${BUILD}/ejbone"/>
  <mkdir dir="${BUILD}/ejbone/META-INF"/>
  <javac srcdir="${SOURCE}" destdir="${BUILD}/ejbone"
    excludes="mypackage/client/**"
    includes="mypackage/common/**,
      mypackage/ejbone">
  <classpath>
    <pathelement
      path="${WL_HOME}/lib/weblogic.jar"/>
    </classpath>
  </javac>
  <copy todir="${BUILD}/ejbone/META-INF">
    <fileset dir="${SOURCE}/mypackage/ejbone/META-INF">
      <include name="*.xml"/>
    </fileset>
  </copy>
</target>
```



The compile target is responsible for compiling the classes for the EJBOne. First, we call mkdir to create the necessary directories. Remember that the commands are operating-system independent, so this script should work in any environment. After creating the directories, we are ready to compile some code.

We invoke the javac compiler by calling the Ant task javac. The javac task allows us to selectively choose which packages and Java classes to compile. If the includes property is not specified, all files in the source directory will be compiled. The excludes property allows the removal of specific Java files and packages.

In this case, we need to compile all of the classes in package mypackage.common and mypackage.ejbone, except the class ExtraClass and the package mypackage/ejbone/notneeded. We define the class path to ensure we compile with the correct WLS version. Finally, we copy the deployment descriptors into the build directory so they can be jarred together.

Building a JAR

The next step is to build the JAR file for the EJB. The following code uses the Ant core task JAR to build the preliminary JAR file containing the compiled classes and deployment descriptors.

```
<target name="jar_ejbone" depends="compile_ejbone">
  <jar jarfile="${BUILD}/_ejbone.jar"
    basedir="${BUILD}/ejbone"/>
</target>
```

The jar_ejbone target depends on the compile_ejbone target. We can't create the JAR file until we compile the classes. When the jar_ejbone target is called, it ensures the compile_ejbone target is called first. If the compile_ejbone target has not been executed, it will automatically execute before executing the jar_ejbone target.

Running EJBC

The XML in the following code demonstrates how to integrate ejbc with Ant. This target has two dependencies, jar_ejbone and compile_ejbone. The order of the dependencies is not important because jar_ejbone can never be executed before compile_ejbone. jar_ejbone will not be executed twice, even though it's listed as a dependency for two targets.

```
<target name="ejbc_ejbone" depends="jar_ejbone,compile_ejbone">
  <java classname="weblogic.ejbc" fork="yes" failonerror="yes">
    <sysproperty key="weblogic.home"
      value="${WL_HOME}"/>
    <arg line="-compiler javac ${BUILD}/_ejbone.jar
      ${STAGING}/ejbone.jar -keepgenerated"/>
```



```
<classpath>
  <pathelement
path="${WL_HOME}/lib/weblogic.jar"/>
</classpath>
</java>
</target>
```

Target `ejbc_ejbone` calls the `java` task. This task, as you can probably guess, executes Java applications. The Java application, in this case, is `weblogic.ejbc`. `Fork` launches a new `vm`, and `failonerror` does exactly that. If the `ejbc` compile process fails, the build will fail and stop. The `failonerror` property can only be set to `yes` if the `fork` property is also set to `yes`. `Sysproperty` allows us to specify properties to `ejbc`. Next we specify the command line parameters for `ejbc`. And finally we define the class path.

Deploying

Now that we have created our EJB JAR file, let's deploy it. The `update_ejbone` target shown below looks almost identical to the `ejbc_ejbone` target, except we run `weblogic.deploy`.

```
<target name="update_ejbone"
depends="ejbc_ejbone,jar_ejbone,compile_ejbone">
  <java classname="weblogic.deploy"
fork="yes" failonerror="yes">
    <sysproperty key="weblogic.home"
value="${WL_HOME}"/>
    <arg line="-url ${DEVELOPMENTSERVER} update mypassword ejbone
${STAGING}/ejbone.jar"/>
    <classpath>
      <pathelement
path="${WL_HOME}/lib/weblogic.jar"/>
    </classpath>
  </java>
</target>
```

The argument line for `weblogic.deploy` is very interesting. First, we use the `DEVELOPMENTSERVER` parameter from the `config.parameter` file; this will set the deployment URL to our predefined development server.

Next, we specify the action for `weblogic.deploy` to perform. The next parameter is the password; since we didn't specify a user name, it defaults to "system." It's not always a great idea to hard code the system password. Later we will examine a better solution.

Finally we specify the component name and the component location. Now if we type `c:\Ant update_ejbone`, `EJBOne` will be compiled, `JARred`, `ejbc` compiled, and deployed to the development server. The component will be deployed to the staging directory, not the application directory. If you have previously installed the same component to the application directory, you should remove it.

Additionally, this example demonstrates how to update an already deployed component. Creating a target that installs a new component is a very similar process. I chose to demonstrate the update because I often deploy the same components, but very rarely create entirely new components.

Building a WAR

Now that we have taken care of our EJB, let's create a WAR file. The copy and compile targets in Listing 1 ensure that everything required to build the WAR file ends up in the `build/war` directory. Ant has a predefined task called `war` that is used in the `war` target. We could use the `jar` task, but the `war` task makes sure everything ends up in the correct directory. The `webxml` property allows us to set the location of the `web.xml` deployment descriptor, but the `war` task was not made specifically for `WebLogic`, so we must manually include the `weblogic.xml` file. The `webinf` property allows us to define additional files that belong in the `WEB-INF` directory of the WAR file. Here we include the `weblogic.xml` deployment descriptor. Next we define where the classes are located by setting the `classes` property. And finally the `fileset` property is used to add the HTML, images and JSP files.

Easing Deployment of Large Applications

What if there is a need to deploy all of our components at one time? It would be tedious to run each `build.xml` script separately. We can create one Ant build script that runs all of the Ant build scripts for our application. Listing 2 shows the master `build.xml` script. This script will be the script that manages all of the scripts in the application. I define the target "all," which has no functionality of its own. Its only purpose is to invoke its dependencies. The `ejbone`, `ejbtwo`, and `war` targets use the `ant` task

SAVE 30% off the annual newsstand rate

JAVA DEVELOPER'S JOURNAL

Offer subject to change without notice

ANNUAL NEWSSTAND RATE
\$71.88
YOU PAY
\$49.99
YOU SAVE
30% Off the Newsstand Rate

DON'T MISS AN ISSUE!

Receive 12 issues of *Java Developer's Journal* for only \$49.99! That's a savings of \$21.89 off the cover price. Sign up online at www.sys-con.com or call 1 800 513-7111 and subscribe today!

In February JDJ:

Getting Started with Java on PDAS

Build and deploy Java apps
by Rob Tiffany

Pattern Foundations: The Open-Closed Principle

Create systems that are easily maintained and flexible
by Kirk Knoernschild

Coming Out of the JDO Closet

The JDO specification is a promising newcomer in the object persistency arena – for J2EE projects large and small
by Yaron Telem and Shay Litvak

Use XML Inside Existing Applications...

...without learning the XML native programming models
by John Goodson





" If Ant doesn't contain what you need, it can be extended using Java. And because Ant is extendable, it's being integrated with almost everything, including WLS "

to call another build script. This allows us to control all of the build scripts within our project from the master script. The master build script will be placed in the root source directory. By typing "ant" in that directory, all three of the components will be deployed. Typing `c:\> ant ejbone` and the WAR file.

The next target in the master build file is "ear." This target builds an EAR file and deploys it to the development server. This time, when we call the other build.xml scripts, we specify the `ejbc` and `war` targets, because we don't want to deploy them as individual components.

Next, we copy the application.xml into the META-INF directory, and JAR everything together into the EAR. The next step in the target is the deployment. The deployment has been changed by adding the `{password}` parameter. Now the password is no longer hard-coded into the script. When running the script, simply type `c:\ant ear -Dpassword=yourpasswordhere`, and the password is passed in from the command line.

Notice, when we call `weblogic.deploy`, we use the "update." This will only update a component that is already deployed. We could add an additional target for new deployments or add targets

that deploy to different servers. We could even add targets that deploy logical groups of components. The master build file adds an incredible amount of flexibility.

Conclusion

Now that you know how to use Ant to build and deploy WAR, EJB, and EAR files to WLS, and how to use a master build script to manage your project, you can see how the master build script can be adapted to meet the unique challenges of your project. This is just a sample of what Ant can do to improve your development process.

When implementing Ant for your project, you should consider creating a clean target for each build.xml file. The clean target would be responsible for removing any files generated by the build.xml file. Common parts of the EJB build.xml file can be moved to the master build script, although I strongly recommend leaving the compilation and building of the JAR file in a separate build.xml for each EJB.

Try integrating Ant with your version control system. Remember, the more repetitive tasks we automate, the more time we can spend developing software.

Listing 1

```
<project name="War File" default="war" basedir=".">
<property file=".../config.properties"/>

<target name="update_war" depends="compile_war,war">
<java classname="weblogic.deploy" fork="yes"
failonerror="yes">
<sysproperty key="weblogic.home"
value="{WL_HOME}"/>
<arg line="-url {DEVELOPMENTSERVER} update
mypassword war {STAGING}/war.war"/>
<classpath>
<pathelement path="{WL_HOME}/lib/
weblogic.jar"/>
</classpath>
</java>
</target>

<target name="compile_war">
<mkdir dir="{BUILD}/war"/>
<javac srcdir="{SOURCE}" deprecation="on"
destdir="{BUILD}/war"
includes="mypackage/client/**,
mypackage/common/**">
</javac>
</target>

<target name="war" depends="compile_war">
<war warfile="{STAGING}/persdemo.war"
webxml="{SOURCE}/mypackage/client/
web-inf/web.xml">
<webinf dir="{SOURCE}/mypackage/client/
web-inf/">
<patternset id="wls">
<include name="weblogic.xml"/>
</patternset>
</webinf>
```

```
<classes dir="{BUILD}/war"/>
<fileset dir=".../content"/>
</war>
</target>
</project>
```

Listing 2

```
<project name="Master" default="all" basedir=".">
<property file="./config.properties"/>

<target name="all" depends="ejbone,ejbtwo,war"/>

<target name="setup">
<mkdir dir="{BUILD}"/>
<mkdir dir="{STAGING}"/>
</target>

<target name="clean">
<delete>
<fileset dir="." includes="{BUILD}"/>
<fileset dir="." includes="{STAGING}"/>
</delete>
</target>

<target name="ejbone" depends="setup">
<ant antfile="build.xml" dir="{SOURCE}/
mypackage/ejbone/" target="update_ejbone"/>
</target>

<target name="ejbtwo" depends="setup">
<ant antfile="build.xml" dir="{SOURCE}/
mypackage/ejbtwo/" target="update_ejbtwo"/>
</target>

<target name="war" depends="setup">
<ant antfile="build.xml" dir="{SOURCE}/
mypackage/client/" target="update_war"/>
</target>
```

```
<target name="ear" depends="setup">
<ant antfile="build.xml" dir="{SOURCE}/
mypackage/ejbone/" target="ejbc_ejbone"/>
<ant antfile="build.xml" dir="{SOURCE}/
mypackage/ejbtwo/" target="ejbc_ejbtwo"/>
<ant antfile="build.xml" dir="{SOURCE}/
mypackage/client/" target="war"/>

<mkdir dir="{BUILD}/ear"/>
<mkdir dir="{BUILD}/ear/META-INF"/>

<copy todir="{BUILD}/ear">
<fileset dir="{STAGING}">
<exclude name="**/*.ear"/>
</fileset>
</copy>

<copy file="{SOURCE}/ear/META-INF/
application.xml" tofile="{BUILD}/ear/META-INF/
application.xml"/>

<jar jarfile="{STAGING}/project.ear" basedir=
"{BUILD}/ear"/>
<java classname="weblogic.deploy" fork=
"yes" failonerror="yes">
<sysproperty key="weblogic.home" value=
"{WL_HOME}"/>
<arg line="-url {DEVELOPMENTSERVER} update
{password} projectname {STAGING}/
project.ear"/>
<classpath>
<pathelement path="{WL_HOME}/lib/
weblogic.jar"/>
</classpath>
</java>
</target>
</project>
```

Altoweb

www.altoweb.com

Using EJBGen



BY
DION ALMAER

AUTHOR BIO...

Dion Almaer is a principal technologist for The Middleware Company (www.middleware-company.com), one of the nation's leading training companies in EJB/J2EE and B2B technology training. The Middleware Company also built and maintains TheServerSide.com, the leading online J2EE community.

CONTACT...

dion@middleware-company.com

Sick of working with all of those different Java files and deployment descriptors when developing EJBs for WebLogic Server? A couple of tools out there allow you to work with just the bean code, and use special Javadoc comments to define what should be in the other interfaces (Home, Remote), and the XML deployment descriptors (`ejb-jar.xml`, `weblogic-ejb-jar.xml`). After using these tools, you quickly realize how clean it is to just have one Java source file representing your EJB.

Requests are made, bugs are found, and Cedric fixes them in short order. If you need an open source solution, then you can use XDoclet (<http://sourceforge.net/projects/xdoclet>), which is derived from EJBDoclet, written by JBoss creator Rickard Oberg.

In this article, I create a set of WebLogic 6.1 deployable EJBs representing a simple Order system. There will be two entity beans, which represent an order and its items. These will use EJB 2.0 CMP relationships, allowing us to get access to the items from the order itself. We'll have a session bean that manipulates these entity beans to get work done.

Using EJbGen

At a simple level, EJbGen allows you to place information about your EJBs in Javadoc tags, which are embedded in the bean class. The special tags are formatted as follows:

```
/**
 * @ejbgen:tagname attribute = value attribute2
 = value2 ...
 */
```

These @ejbgen tags are placed either at the class level, or at the method level, depending on the type of tag. At the class level you can declare the EJB's name, the JNDI name, and attributes for the entire bean. The method level attributes flag information on the given method. The most common tag you'll use is for the method to be part of a Remote or Local interface (using @ejbgen:remote-method, or @ejbgen:local-method).

```
/**
 * @ejbgen:remote-method
 */
public String viewMaxOrders() {
```

After your bean classes have been created, and the Javadoc tags have been plugged in, you can run EJbGen through your Javadoc client:

```
% javadoc -docletpath /path/to/ejbgen.jar -
doclet EJbGen YourEJBs.java
```

There are also command line options that tweak the way EJbGen generates its files. Many of them allow you to change the prefixes and suffixes of the various elements.

To generate a remote interface named FooRemote.java instead of Foo.java, you would run the following command (note the -remoteSuffix):

```
% javadoc -docletpath ejbgen.jar -doclet EJbGen
-remoteSuffix Remote FooEJB.java
```

The full set of command line options are documented on the EJbGen site.

EJbGen Features

EJbGen also has the following features:

- Fully EJB 2.0 compliant (local interfaces/ CMP relationships/message driven beans)
- Generates WebLogic 6+ deployable code and deployment descriptors
- Can generate Value objects for the Entity Beans
- Property files: properties can be defined and used inside the bean class. A property is used via `{ property name }`
- Supports Javadoc tag inheritance. A base adapter class can set a standard values, then your actual bean inherits the behavior, as well as the methods
- Generates ANT build files

If you generate an ANT build file, after running EJbGen, you can run

```
% ant -buildfile ejbgen-build.xml
```

You can tweak variables such as:

- **ant.buildFileName:** Name of the buildfile
- **ant.projectName:** Name of the project
- **ant.docletPath:** Path to ejbgen.jar

Creating a Session Bean That Uses Javadoc Tags

Now let's get into the real example, starting with the session bean that manipulates the entity beans.

The OrderViewerEJB.java will hold the following methods:

- **viewMaxOrders():** Returns the maximum orders, which will be an <env-entry>, to show how we set them up in the class-level javadoc comment
- **viewOrders():** Returns the orders in the system
- **addOrder(orderId, orderBy, itemName):** Inserts the order entry
- **addOrderItem(orderID, itemID, itemName, amount):** This will insert an item for the given order
- **deleteOrders():** Deletes all of the orders and items within the orders

CLASS-LEVEL DEFINITIONS

At the class level, we need to define the fact that this is a session bean, and information relevant to that: the JNDI name; EJB local references to the entity beans; and environment entries.

TAG: @EJBGEN:SESSION

The only required attribute here is to tell EJbGen the name of the bean. We'll also define the maximum beans in the free pool (WebLogic caching info), and that the default transaction attribute will be "Required."

Introducing EJbGen

In this article we'll look at a tool for generating WebLogic 6.1-compatible EJBs, EJbGen, that was developed by Cedric Beust, a BEA WebLogic architect (www.beust.com/cedric/ejbgen). It can easily be added to your WebLogic build process and is IDE neutral – you just add Javadoc tags in your bean class. Unfortunately, the tool is closed source, since it was developed solely by Cedric, but he is accessible once you join the mailing list.

```
/**
 * OrderViewer Bean views orders:
 *
 * @ejbgen:session
 *   ejb-name           = OrderViewer
 *   max-beans-in-free-pool = 100
 *   default-transaction = Required
 *
 * ... other tags ...
 */
public class OrderViewerEJB implements SessionBean {
```

The full set of attributes for the ejbgen:session tag is shown in Table 1.

TABLE 1

WEBLOGIC ATTITUDE	DESCRIPTION
ejb-name	The name of this Session bean.
default-transaction	The transaction attribute to be applied to all methods that don't have a more specific transaction attribute setting.
idle-timeout-seconds	Maximum duration an EJB should stay in the cache.
initial-beans-in-free-pool	The initial number of beans in the free pool.
max-beans-in-cache	The maximum number of beans in the cache.
max-beans-in-free-pool	The maximum number of beans in the free pool.
trans-timeout-seconds	The transaction timeout (in seconds).
type	(Stateless Stateful) The type of the Session bean. If this attribute isn't specified, EJBGen will guess the right type by looking at the ejbCreate() methods on your class.

Attributes for ejbgen:session tag

TAG: @EJBGEN:JNDI-NAME

This tag is very important; not only does it define the JNDI names, but EJBGen also uses it to know whether to generate a remote interface, a local interface, both, or neither! Since we want to expose the session bean to remote clients, we'll use the remote attribute:

```
/**
 * OrderViewer Bean views orders:
 *
 * ... other tags ...
 *
 * @ejbgen:jndi-name
 *   remote = ejbgenexamples.OrderViewer
 *
 * ... other tags ...
 */
public class OrderViewerEJB implements SessionBean {
```

For a local interface we would add the attribute "local = local-jndi-name."

TAG: @EJBGEN:EJB-LOCAL-REF

Our OrderViewer will use the entity beans that represent an order, and an order item. EJB 2.0 introduced the idea of "local interfaces" which force the components to "talk" in the local VM, not across remote boundaries. When working with entity beans, it's a good practice to have session beans (which are remote capable) work with entity beans locally. We'll define the two

EnginData Presents

2002 Developer Market Survey Reports

\$2,995



Java
Developer
Market Survey
2002

\$2,495



Web Services/XML
Developer
Market Survey
2002

\$1,995



Wireless
Developer
Market Survey
2002

Our comprehensive reports offer insight and strategy to guide your most critical business decisions in today's fastest growing technologies...

- Establish your product and marketing strategy
- Understand your customer's needs
- Evaluate Technology & Trends

Preview and order reports at www.engindata.com



local references to the two entity beans (Listing 1).

With these local references, we can get access to the Home interface by looking up "java: comp/env/<name>" to look up the OrderLocal Home:

```
OrderLocalHome orderHome = (OrderLocalHome) new
InitialContext().lookup("java:/comp/env/ejb/Order-
Home");
```

To reference EJBs with remote contracts, use the @ejbgen:ejb-ref tag.

TAG: @EJBGEN:ENV-ENTRY

To define environment entries you use the ejbgen:env-entry tag. We'll define an environment entry to hold the integer value of the maximum orders:

```
/**
 * OrderViewer Bean views orders:
 *
 * ... other tags ...
 *
 * @ejbgen:env-entry
 *   name = maxOrders
 *   type = java.lang.Integer
 *   value = 10
 *
 * ... other tags ...
 */
public class OrderViewerEJB implements SessionBean {
```

The type must always be fully qualified (java.lang.Integer, not Integer).

METHOD LEVEL DEFINITIONS

We want to tag all of the methods that we want available to a remote client. As an option, you can also pass in values for the transaction attribute of the method (i.e., override the default value that we set up at the class level) and the isolation level. Here we tag the addOrder() method:

```
/**
 * Add an order
 *
 * @ejbgen:remote-method
 */
public void addOrder(Integer id, String orderBy, String
name)
```

We'll see that the entity beans will tag their methods @ejbgen:local-method. Now our session bean is good to go!

Creating the OrderItem Entity Bean

Working with entity beans is similar to working with session beans. There are a couple of different class level tags; we'll flag the fields that the container should manage, and the primary key. We'll generate an OrderItem entity that includes three fields – itemid, itemname, and itemamount – that will persist.

Corda Technologies

www.corda.com

The database table will also contain an ORDER_ID that's a foreign key into the Order entity (discussed later). For now, we'll ignore the fact that the OrderItem will have a relationship with an Order.

CLASS LEVEL DEFINITIONS

At the class level, we will define the entity definition, the JNDI name, and the entity finder signature, and we'll create the tables in the DB.

TAG: @EJBGEN:ENTITY

As with the session bean, we need to define the name of the entity bean and can define attributes like the default transaction value. We also need to define the name of the table in the database, the primary key class, and the data source to use:

```
/**
 * @ejbgen:entity
 *   ejb-name           = OrderItemEJB
 *   data-source-name  = orderPool
 *   table-name        = OrderItems
 *   prim-key-class    = java.lang.Integer
 *   default-transaction = Required
 *
 * ... other tags ...
 */
public abstract class OrderItemEJB implements
EntityBean {
```

The full set of attributes for the ejbgen:entity tag are shown in Table 2.

TABLE 2

WEBLOGIC ATTITUDE	DESCRIPTION
ejb-name	Name of this Entity bean
data-source-name	Name of the JDBC DataSource
prim-key-class	Class name of the primary key
table-name	Name of the database table
abstract-schema-name	Abstract schema name for this EJB. If not specified, the ejb-name value will be used
concurrency-strategy	Defines the concurrency strategy for this bean (ReadOnly Exclusive Database)
db-is-shared	True or False
default-transaction	The transaction attribute to be applied to all methods that don't have a more specific transaction attribute setting.
delay-database-insert-until	ejbCreate or ejbPostCreate
idle-timeout-seconds	Maximum duration an EJB should stay in the cache
initial-beans-in-free-pool	The initial number of beans in the free pool
invalidation-target	The ejb-name of a read-only Entity bean that should be invalidated when this Container-Managed Persistence Entity EJB has been modified
max-beans-in-cache	The maximum number of beans in the cache
max-beans-in-free-pool	The maximum number of beans in the free pool
prim-key-class-nogen	If specified, EJBGGen won't generate the primary key class
read-timeout-seconds	The number of seconds between each ejbLoad() call on a Read-Only Entity bean
reentrant	True or False
trans-timeout-seconds	The transaction timeout in seconds

Attributes for the ejbgen:entity tag

TAG: @EJBGEN:JNDI-NAME

This is identical to the session, but since we're using entities we'll create only a local interface:

```
/**
 * ... other tags ...
 *
 * @ejbgen:jndi-name
 *   local = ejbgenexamples.OrderItemLocalHome
 *
 * ... other tags ...
```

TAG: @EJBGEN:FINDER

Every finder method must have an ejbgen:finder tag. You must specify the method signature, including any exceptions that you may want to throw. Also, pass in the EJB Query Language that you wish to use. We'll define a findItemsByName() method that retrieves every item with a given name:

```
/**
 * @ejbgen:finder
 *   signature = "Collection findItemsByName(String name)"
 *   ejb-ql    = "WHERE name = ?1"
```

TAG: @EJBGEN: CREATE-DEFAULT-DBMS-TABLES

For development it's nice to have WebLogic create the database tables for you:

```
* @ejbgen:create-default-dbms-tables
```

METHOD LEVEL DEFINITIONS

All of the abstract get() methods must be flagged as container-managed persistence fields, and we give the column name mapping. The primary key must be flagged, and if we want to be able to call these methods from a component interface, we must flag that too (remote or local). Here's the primary key for the OrderItem:

```
/**
 * @ejbgen:cmp-field column = id
 * @ejbgen:local-method
 * @ejbgen:primkey-field
 */
public abstract Integer getId();
public abstract void setId(Integer id);
```

Creating the Order Entity Bean, and Relating It to the OrderItem

WebLogic 6.1 supports EJB 2.0 relationships, a feature not seen in many other J2EE servers. WebLogic, however, has no deployment descriptor tools to deploy your CMP relationship entities (XML must be manually manipulated), so we'll use EJBGGen's features to accomplish this.

We'll have a one-to-many relationship in which an order can have multiple items within it. From the order, we'll be able to get a Collection of items via getItems(), and from an item we can get back to the order via OrderLocal.getOrder(). As well as defining these abstract methods, we need to configure ejb-

gen:relation tags in each class (OrderEJB, and OrderItemEJB). The steps for this relationship are:

1. **OrderItem:** Configure ejbgen: relation tag
2. **OrderItem:** Create container managed relationships
3. **Order:** Configure ejbgen:relation tag
4. **Order:** Create container managed relationships

ORDERITEM: CONFIGURE EJBGEN: RELATION TAG

This item is the “many” side of the relationship. Both sides of the relationship share the same name, and the target-ejb is the name of the bean for which this relationship is to be used. Since we have the target-ejb attribute, you could place all relation tags in one EJB bean class if you wanted to. Since our multiplicity is many, we will tell EJBGen the foreign key column and the field that maps to the container-managed relationship. When an order is deleted, we want the items to be deleted automatically, which is where the cascade-delete option comes in:

```
* @ejbgen:relation
* multiplicity = many
* name =
OrderCanHaveMultipleItems
* target-ejb = OrderItemEJB
* fk-column = order_id
* cmr-field = order
* cascade-delete = True
```

ORDERITEM: CREATE CONTAINER MANAGED RELATIONSHIPS

We can get to the order from an item within this order. We tag the CMR field and declare it as a local method:

```
/**
 * @ejbgen:cmr-field
 * @ejbgen:local-method
 */
public abstract OrderLocal getOrder();
public abstract void setOrder(OrderLocal order);
```

ORDER: CONFIGURE EJBGEN: RELATION TAG

This order is the “one” side of the relationship. The relation has to have the same relation name:

```
* @ejbgen:relation
* multiplicity = one
* name =
OrderCanHaveMultipleItems
* target-ejb = OrderEJB
```

ORDER: CREATE CONTAINER MANAGED RELATIONSHIPS

Since an order has many items, we work with a Collection of OrderItems:

```
/**
 * @ejbgen:cmr-field
 * @ejbgen:local-method
 */
public abstract Collection getItems();
public abstract void setItems(Collection items);
```

If you’ve ever worked with CMP relationships then you may have cursed a few times. However, EJBGen makes CMP relationships in WebLogic rather easy. I’d never want to build the XML for the relationships manually. I want to use a GUI to build my relationships, or I can use EJBGen. To take EJBGen a step further, there are GUIs that build EJBGen-aware classes. Check out the freeware Middlegen (<http://boss.bekk.no/boss/middlegen/>).

Deploying the System

Now we have three Java source files: OrderEJB.java, OrderItemEJB.java, and OrderViewerEJB.java. From these we’ll use EJBGen to generate the interfaces and XML deployment descriptors. Here’s a build script that will package an ejb-jar file:

```
javac -docletpath c:\ejbgen\ejbgen.jar -doclet EJBGen
ejbgenexamples\Order*EJB.java
cd ejbgenexamples
javac -classpath %CLASSPATH%;.. *.java
cd ..
mv *.xml META-INF
jar cvf c:\bea\wlserver6.1\config\mydo-main\applications\ejbgen.jar *
```

We must have a data source configured in the application server – under the name “orderPool” – before the application can be run. It would also help to have a client, which can run the methods on the OrderViewer session bean. Listing 2 shows the main() method of a client that adds, views, and then deletes an order containing a couple of items.

Summary

We’ve seen how to develop EJBs for WebLogic 6.1 using EJBGen. This tool allows us to have one file to worry about per EJB rather than three Java files and multiple descriptors. It’s a nice clean preprocessing system – and there are even other tools that can generate EJBGen-aware code! EJBGen is a winner for any WebLogic development environment.

SUBSCRIBE AND SAVE

XML JOURNAL

Offer subject to change without notice

ANNUAL NEWSSTAND RATE	
\$83.88	
YOU PAY	
\$77.99	
YOU SAVE	
\$5.89	Off the Newsstand Rate

DON'T MISS AN ISSUE!

Receive 12 issues of **XML-Journal** for only **\$77.99!** That's a savings of **\$5.89** off the annual newsstand rate.

Sign up online at www.sys-con.com or call 1 800 513-7111 and subscribe today!

In February **XML**:

SOAP Messages with Attachments

How this emerging W3C note can be used with the Apache SOAP implementation

Applied Templates

Simplify XSLT code with applied templates

XML Development Environments

How to create a source environment for XML

Q&A: DIVE into XML

by Trace Galloway

I Am the Official Mascot of XSLT

A transforming robot looking for work

Intelligent XML Content Firewalls

Semantic-based filtering of digital content



Listing 1

```
/**
 * OrderViewer Bean views orders:
 *
 * ... other tags ...
 *
 * @ejbgen:ejb-local-ref
 *   home      = ejbgenexamples.OrderLocalHome
 *   local     = ejbgenexamples.OrderLocal
 *   jndi-name = ejbgenexamples.OrderLocalHome
 *   name      = ejb/OrderHome
 *   type      = Entity
 *
 * @ejbgen:ejb-local-ref
 *   home      = ejbgenexamples.OrderItemLocalHome
 *   local     = ejbgenexamples.OrderItemLocal
 *   jndi-name = ejbgenexamples.OrderItemLocalHome
 *   name      = ejb/OrderItemHome
 *   type      = Entity
 *
 * ... other tags ...
 */
public class OrderViewerEJB implements SessionBean {
```

Listing 2

```
public static void main(String[] args) throws Exception {
    Context ic = getInitialContext();

    OrderViewerHome home = (OrderViewerHome)
        ic.lookup("ejbgenexamples.OrderViewer");
    OrderViewer v = home.create();

    System.out.println("Add Order");
    v.addOrder(new Integer(1), "Dion", "Books");
    v.addOrderItem(new Integer(1), new Integer(1),
        "EJB v3", 36.55f);
    v.addOrderItem(new Integer(1), new Integer(2),
        "Mastering EJB", 45.95f);

    System.out.println("The Max Orders: " + v.viewMaxOrders());

    System.out.println("Display Orders:\n" + v.viewOrders());

    System.out.println("Delete Orders");
    v.deleteOrders();
}
```

WLDJ ADVERTISER INDEX

ADVERTISER	URL	PHONE	PAGE
Allaworks	www.allaworks.com	603-598-2582	9
Alloweb	www.alloweb.com	650-251-1500	45
BEA	www.developer.bea.com	408-570-8000	2,3 26,27
BEA e-world	www.bea.com/events/eworld/2002	408-240-5506	59
Corda Technologies	www.corda.com	801-805-9400	49
EnginData	www.engindata.com		48
Java Developer's Journal	www.sys-con.com	800-513-7111	43
Mongoose Technologies	www.mongooseotech.com	281-461-0099	7
Report Mill Software	www.reportmill.com	214-513-1636	21
Sitraka	www.sitraka.com	416-594-1026	15, 60
SYS-CON Media	www.sys-con.com	800-513-7111	57
WebLogic Developer's Journal	www.sys-con.com	800-513-7111	57
Web Services Edge	www.sys-con.com	201-802-3069	30,31 32,33
Web Services Journal	www.wsj2.com	800-513-7111	53
Wily Technology	www.wilytech.com	888-GET-WILY	4
Wireless Edge	www.sys-con.com	201-802-3069	38,39
Wireless Business & Technology	www.wbt2.com	800-513-7111	55
XML Journal	www.sys-con.com	800-513-7111	51

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

NeXT MONTH



FOCUS ON eWORLD

- An introduction from CEO Alfred Chuang
- An exclusive look at some of the newest features of WebLogic

PLUS

- An interview with Scott Dietzen, CTO of BEA Systems
- Tyler Jewell talks about architecting



WebServices JOURNAL

Special
Introductory
Offer
SAVE \$13.89*

The world's leading independent
Web Services information resource

Get Up to Speed with the Fourth Wave in Software Development

- Real-World Web Services: Is It Really XML's Killer App?
- Demystifying ebXML: A Plain-English Introduction
- Authentication, Authorization and Auditing: Securing Web Services
- Wireless: Enable Your WAP Projects for Web Services
- The Web Services Marketplace: An Overview of Tools, Engines and Servers
- Building Wireless Applications with Web Services
- How to Develop and Market Your Web Services
- Integrating XML in a Web Services Environment
- Real-World UDDI
- WSDL: Definitions and Network Endpoints
- Implementing SOAP-Compliant Apps
- Deploying EJBs in a Web Services Environment
- Swing-Compliant Web Services
- and much, much more!

Only \$69.99 for 1 year (12 issues)
Newsstand price \$83.88 for 1 year



 **SYS-CON
MEDIA**

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

*Offer subject to change without notice



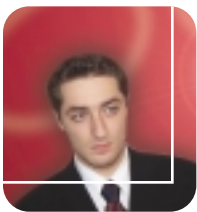
Upgrades

Are you planning to upgrade from WebLogic version 5.1 to version 6.1? Do you want to speed up the process of redeploying your EJBs, servlets, and JSPs to a newer version of an application server?

J2EE app migration from BEA WebLogic Server 5.1 to 6.1

FACILITATING REDEPLOYMENT

BY DMITRIY YAKUBOVICH



AUTHOR BIO...

Dmitriy Yakubovich started working with Java when it came out and has worked with many Java and Java-related technologies, mostly in the field of distributed computing. He has extensive experience with WebLogic, from version 4.5 to 6.1; is a BEA Weblogic 6.0-certified developer, and is Sun Certified. He is the cofounder of a successful software company.

CONTACT...

dmitriy@preciseny.com

The J2EE specification doesn't define any standard deployment procedures. These differ from vendor to vendor and even from version to version of a particular application server. Read on to find out which steps you should take to redeploy your components and estimate the amount of work to be done.

Migrating and Redeployment

Every J2EE application server allows you to install an EJB if you provide a bean implementation, a remote interface, a home interface, and a deployment descriptor. However, the deployment procedures are vendor-specific and sometimes vary between different versions of the same application server. Everybody wants to move along with the technology and utilize all the new services, that a current J2EE specification provides. It would be beneficial to have standard deployment procedures across all J2EE application servers. Unfortunately, the current EJB specification does not define such policies, so we all have to adjust and follow the guidelines for redeploying our components. This first occurred to me when I was working for a start-up company. We had a solid infrastructure build on top of an application server, but for a number of reasons we had to migrate our system to a newer version of the same application server. I turned my attention to the redeployment issues. In this article I'll concentrate on how to migrate a J2EE application

from WebLogic Server 5.1 to 6.1. No programmatic changes need to take place in order to redeploy your components. It's the deployment procedures that differ. There are a number of deployment benefits in WLS 6.1, including deployment formats, consoles for administrating and deploying applications, and EJB and Resource adapter components.

Version 6.1 supports multiple formats for deployment. A J2EE application can be deployed on WebLogic Server either as an Enterprise Application Archive (EAR) file or in an exploded directory format. I shall address the process of migrating and deploying the application as an enterprise application archive further in the article. For now, let me outline high-level procedures in the migration process:

1. Convert WebLogic license file
2. Create a new domain in WLS 6.1
3. Convert WebLogic.properties to XML files
4. Modify start/stop script to use new domain
5. Migrate servlets & JSPs
6. Migrate EJB applications
7. Create Web Application Archive (.war)
8. Create Enterprise Application Archive (.ear)

Now, let's revisit each step and examine it in more detail.

Convert WebLogic License File

The XML-format license file (WebLogicLicense.XML) and the Java-format license file (WebLogicLicense.class) used in pre-6.0 versions (5.1 or earlier) of WebLogic Server are no longer supported. If a WebLogicLicense.class license file is used in your existing WebLogic Server installation, perform the following tasks before you install WebLogic Server 6.1:

1. Convert the WebLogicLicense.class license file to a WebLogicLicense.XML file, using the licenseConverter utility provided by BEA at www.WebLogic.com/docs51/techstart/licenseConverter.
2. Convert the WebLogicLicense.XML file as described in "Converting a WebLogicLicense.class License" at <http://edocs.bea.com/wls/docs61/install/1036487>.

Create New Domain in WebLogic Server 6.1

A new domain must be created. The administration console provides an interface for creating a new domain. WebLogic will create a directory specifically for your domain. Note that the directory structure

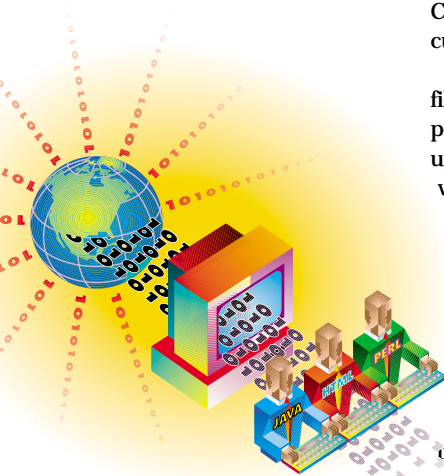


differs from version 5.1. You should refer to the WLS documentation to get familiar with the directory structure.

CONVERT WEBLOGIC.PROPERTIES TO XML FILES

This step – converting WebLogic.properties files into XML files – is probably the most important task. The administration console provides a conversion utility to convert a weblogic.properties file to appropriate .xml files. It's important to understand what

happens behind the scenes. Unlike in WLS 5.1, all the configuration files and deployment descriptors are XML-based. A large weblogic.properties file should be split into a number of XML files. Depending on your application, you may have config.xml, weblogic.xml, web.xml, and application.xml.



happens behind the scenes. Unlike in WLS 5.1, all the configuration files and deployment descriptors are XML-based. A large weblogic.properties file should be split into a number of XML files. Depending on your application, you may have config.xml, weblogic.xml, web.xml, and application.xml.

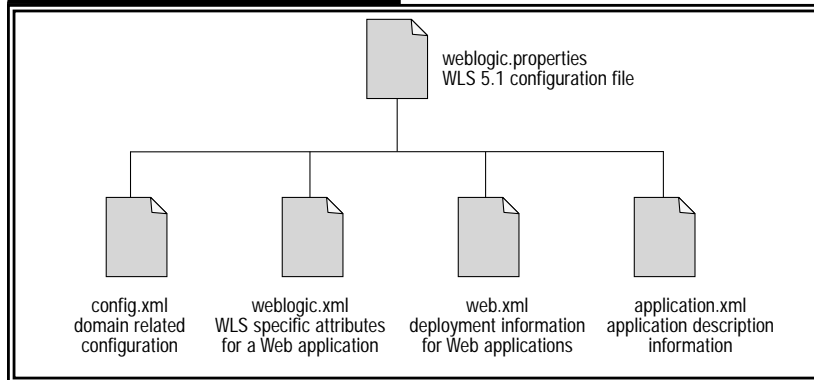
Having different kinds of information in a single configuration file makes it impossible to read and tough to main-

tain. Now, starting with WLS 6.0, all the related information is grouped together in a particular XML file. Your config.xml describes your domain. You should provide a description of the servers in a particular domain, including Listen Ports for connections, SSL-specific information, log file names, security information, and JDBC-related information such as JDBCConnectionPool and JDBCData-Source. The description might also include entries for EJB and Web app components, JMS servers, JMS Connection Factories, LDAP realms, custom realms, and so on. Generally, you shouldn't update this file manually. A configuration console provides an interface to create and update entries in this file. A weblogic.xml file should include WLS-specific attributes, such as jsp-descriptor, security-role-assignment, and a reference-descriptor for a Web application. The deployment information for Web apps should be specified in web.xml. Servlet descriptions, servlet initial parameters, and servlet mappings should be included in this file. The description of your applications, including Web and EJB modules, and additional entries to describe security roles and application resources such as databases should be specified in application.xml. The diagram shown in Figure 1 should assist you in summarizing the described process:

Modify Start/Stop Scripts

The next step, modifying start/stop scripts for a new domain, is fairly straightforward and needs no special attention. There will be default scripts

FIGURE 1



Configuration files

SAVE 30% off the annual newsstand rate

BUSINESS & TECHNOLOGY
wireless

Offer subject to change without notice

ANNUAL NEWSSTAND RATE
\$71.88
YOU PAY
\$49.99
YOU SAVE
30% Off the Newsstand Rate

DON'T MISS AN ISSUE!

Receive 12 issues of **Wireless Business & Technology** for only \$49.99! That's a savings of 30% off the cover price. Sign up online at www.sys-con.com or call 1 800 513-7111 and subscribe today!

Wireless Business & Technology:

Going Wireless: An Introduction to Wireless Security
Kevin Wittmer provides an overview of security technologies available today for LAN and WAN wireless networks

The World Phone: When Will It Truly Work?
A look at the primary competitors and what their strategies are vis-à-vis world phones

Wireless Internet: The Next Generation
NTT SOFT is bringing their unified vision of wireless and Internet technology to the U.S.

Boingo Jumps In:
This company has what it takes to kick-start the Wi-Fi service industry

Product Technology:
Pocket PC Is Enterprise-Ready with a Built-in VPN Client



provided by WLS for your domain. You'll need to modify them for your particular environment. You'll have to set some important environment variables, such as WL_HOME, JAVA_HOME, CLASSPATH, and PATH.

Migrate Servlets, JSPs, and EJBs

CREATE WEB AND ENTERPRISE APPLICATIONS
As noted earlier – a J2EE application can be deployed on WebLogic Server as either an Enterprise Application Archive (EAR) file or in exploded directory format. Shown in figure 2, an Enterprise Application Archive file contains all of the .jar and .war component archive files for an application, as well as an XML descriptor that describes the bundled components.

As noted before, the META-INF/application.xml deployment descriptor contains an entry for each Web and EJB module, and additional entries, meant to describe security roles and application resources such as databases. The JAR files are packaged EJBs. The WAR file (Web Application Archive) is organized in a specified directory structure. All servlets, classes, static files, and other resources belonging to a Web Application are organized under a directory hierarchy. The root of this hierarchy defines the document root of the project. All files under this root directory can be served to the client, except for files under the special directory WEB-INF and META-INF, located in the root directory. The diagram in Figure 3 illustrates the directory structure of any Web Application.

The following steps must be performed to package a project as an Enterprise Application file:

1. Create directory structure for the WAR file and place the Web Application files inside. Follow the directory structure for WAR files above.
2. Create web.xml file with the description of the Web components.
3. Package WAR file using a JAR utility.
4. Copy the WAR and EJB JAR files into the Enterprise Application directory. Follow the directory structure for the EAR file above.
5. Create a META-INF/application.xml deployment descriptor for the application, using a DTD supplied by Sun Microsystems. The application.xml file contains a descriptor for each component in the application.
6. Create the Enterprise Archive using a JAR utility.
7. Click on the Install Applications link under the Getting Started heading in the home page of the console and place the .ear file in the [wls6.1]/config/[your_domain]/applications directory.

Stateful Session Failover

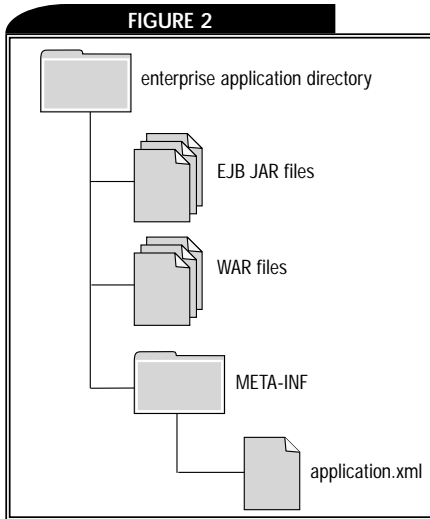
Although there might be more things to take care of while migrating a particular application, one important thing to note is that the WLS 6.1 EJB container provides clustering support for stateful session beans, whereas in WLS 5.1 only the EJBHome is clustered. Replication support for stateful session EJBs is transparent to clients of the EJB. To replicate the position of a stateful session EJB in a WebLogic Server cluster, ensure that the cluster is consistent with the EJB. In other words, deploy the same bean to every WebLogic Server instance in the cluster, using the same deployment descriptor. Hetero-geneous clusters are not supported for In-memory replication.

By default, WebLogic Server does not replicate the state of stateful session EJB instances in a cluster. To enable replication, set the replication-type deployment parameter to InMemory in the weblogic-ejb-jar.xml deployment file. For example:

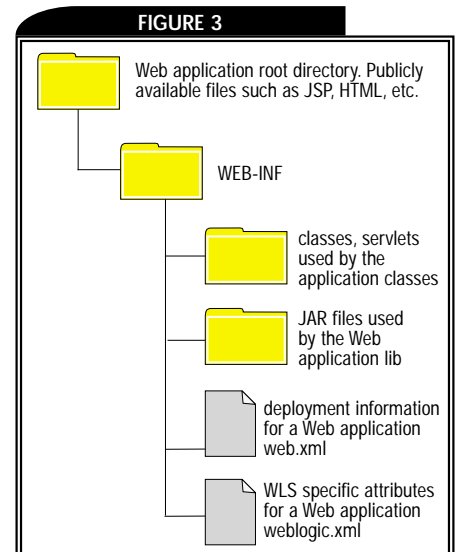
```
<stateful-session-clustering>
...
<replication-type>InMemory</replication-type>
</stateful-session-clustering>
```

Conclusion

At best, the technique discussed in this article facilitates the EJB-migrating of whole applications. At worst, it's a useful pattern to perform the routine aspects of EJB migration. We'll just have to hope that the future EJB specifications will standardize the deployment process to make our applications more portable and vendor-independent.



Enterprise Application



Web Application

Special
online offer

PICK
4
Subscribe

for
One
Special Low
Price

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

WWW.SYS-CON.COM

SYS-CON
MEDIA

OFFER SUBJECT TO CHANGE WITHOUT NOTICE

Now in More than 5,000 bookstores worldwide

subscribe Now!

FORFAST
DELIVERY

The World's leading
Independent WebLogic
Developer Resource
FOR WLS DEVELOPERS BY WLS DEVELOPERS
WebLogic
DEVELOPER'S JOURNAL
An introduction to...

Go
Online
and
Subscribe
Today!

Helping
you enable
inter-company
collaboration
on a global scale

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

SPECIAL
INTRODUCTORY OFFER
SAVE \$31*
HURRY, DON'T DELAY! OFFER EXPIRES DECEMBER 31, 2001

WebLogic Journal.com

SYS-CON Media, the world's leading publisher of *i*-technology magazines for developers, software architects, and e-commerce professionals, brings you the most comprehensive coverage of WebLogic. *Only \$149 for 1 year (12 issues) regular price \$180.

SYS-CON
MEDIA

News & Developments

BEA, Deloitte Consulting to Speed Deployment of e-Business Solutions

(San Jose, CA) – BEA Systems, Inc. and Deloitte Consulting, one of the world's leading consulting firms, have announced a global strategic alliance to streamline the delivery of e-business solutions. Deloitte Consulting is working with BEA software to develop several repeatable solutions, including one to facilitate B2B collaborative commerce, built on the BEA WebLogic E-Business Platform.



The alliance includes an agreement for the companies to engage in joint marketing and sales activities, and for Deloitte Consulting to demonstrate built-on-BEA solutions through its global network of Solution Centers. The centers provide an infrastructure for virtual project teams that can span multiple client, solution center, and business partner sites. Under the agreement, Deloitte Consulting plans to train its e-business technical practitioners on the BEA WebLogic E-Business Platform for implementation of real-time, portal, and enterprise infrastructure solutions. www.bea.com
www.deloitteconsulting.com

i:FAO Starts 2002 With the Debut of cytric v7

(Dearborn, MI) – i:FAO, the corporate-side provider of business travel eProcurement software, is starting installation and deployment of cytric v7, its Internet software for booking and management of business travel.

i:FAO's cytric v7 is the first business travel eProcurement solution to combine all the



benefits of online applications with an open, enterprise-class infrastructure. It's specifically designed to meet the needs of corporations seeking to reduce travel cost. It's written entirely in Java and uses the latest BEA

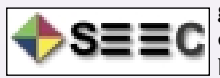
WebLogic Application



Server technology. cytric v7 offers advanced integration, customization, and administration capabilities. www.iFao.net
www.cytric.com

BT Ignite Solutions Deploys Consolidated Billing Gateway Developed by SEEC

(Pittsburgh, PA and Reading, England) – BT Ignite Solutions, the eBusiness and communications solutions unit within BT Ignite, is rolling out a new XML-based consolidated billing



gateway developed by SEEC Europe Limited, the European subsidiary of SEEC, Inc.

The billing gateway, built using SEEC Mosaic Studio, will make it easier and cheaper to launch new products and will improve the processing of orders by providing a seamless interface between BT Ignite Solutions' order management and billing systems. It was the first project within the company to use the BEA WebLogic Server platform. www.btignitesolutions.com



SeeRun Releases SeeRun Acquire for BEA WebLogic Commerce Server (San Francisco) – SeeRun Corp. has released its CRM application Acquire for the BEA

WebLogic Commerce Server. SeeRun Acquire empowers marketing professionals, developers and IT personnel with the ability to track, analyze, and control the entire customer acquisition process in real time, rapidly improving conversion rates and significantly reducing acquisition costs.



SeeRun Acquire is part of the SeeRun Customer Lifecycle Management Suite, and is tightly integrated with other modules in the suite to provide the fullest and richest view of each and every customer throughout the entire lifecycle: targeting, acquisition, retention, service, and growth. www.seerun.com

BEA Systems to Help Businesses Jump-Start Trading

(San Jose, CA) – BEA Systems, Inc. has announced new lightweight trading partner software for BEA WebLogic Integration. The new technology, called BEA WebLogic Integration Business Connect, is designed to help businesses accelerate and automate the connection of supply and demand chains down to the smallest partners and suppliers. Originating from an original equipment manufacturer (OEM) agreement with Cyclone Commerce, BEA WebLogic Integration Business Connect will be available in early 2002.

BEA WebLogic Integration Business Connect provides the security and tools required to automate processes between business partners. It streamlines order management, invoice processing, exception handling, and customer service. www.bea.com



WebLogic Selected as Application Infrastructure Offering

(San Jose, CA) – BEA Systems, Inc., has won a contract to provide BEA WebLogic Integration and BEA WebLogic Personalization Server technology to moneysupermarket.com, an innovative UK-based financial services portal. Moneysupermarket.com reviewed a selection of application infrastructure solutions, including IBM's WebSphere, before selecting BEA WebLogic.



BEA's software will power moneysupermarket.com's expanding e-business operations and allow the company to diversify its service offerings for business partners, consumers, and independent financial advisers. The first of its kind in the UK market, Moneysupermarket.com's account aggregator service – based on BEA WebLogic – is set to launch early in 2002. www.moneysupermarket.com

Events

BEA eWorld 2002

February 25-27, 2002

San Diego Convention Center
San Diego, CA



The CIO Forum – Financial Services

May 6-9, 2002

Queen Elizabeth 2
New York City



BEA e-world

www.bea.com/events/eworld/2002

Sitraka

www.sitraka.com