ANNOUNCING WEBSERVICESEDGE 2001 P.103

web services EDGE conference&expo

JAVA DEVELOPER'S JOURNAL

The World's Leading Java Resource

April 2001 Volume: 6 Issue: 4

JAVADEVELOPERSJOURNAL.COM

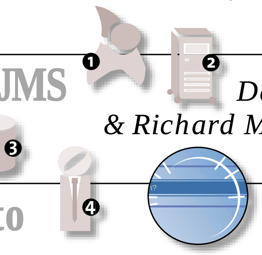
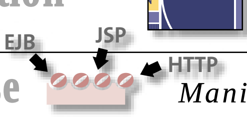
Sept 23–26, 2001 New York, NY
JAVA EDGE conference&expo 98

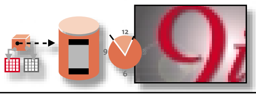
Oct 22–25, 2001 Santa Clara, CA
XML EDGE conference&expo 2001 104

RETAILERS PLEASE DISPLAY UNTIL JUNE 30, 2001
$4.99US $6.99CAN

SYS-CON MEDIA

WEB SERVICES ABSOLUTELY REAL



E D I T O R I A L   O F F I C E S
SYS-CON MEDIA 135 CHESTNUT RIDGE RD., MONTVALE, NJ 07645
TELEPHONE: 201 802-3000 FAX: 201 782-9600
JAVA DEVELOPER'S JOURNAL (ISSN#1087-6944) is published monthly (12 times a year) for $69.99 by SYS-CON Publications, Inc., 135 Chestnut Ridge Road, Montvale, NJ 07645. Periodicals postage rates are paid at Montvale, NJ 07645 and additional mailing offices. POSTMASTER: Send address changes to: JAVA DEVELOPER'S JOURNAL, SYS-CON Publications, Inc., 135 Chestnut Ridge Road, Montvale, NJ 07645.

SEAN RHODY, FOUNDING EDITOR/CHIEF CORPORATE EDITOR

Living on the Edge

This is my first note to let everyone know about the plans for our very own, **SYS-CON**–sponsored **JavaEdge2001 International Java Developer Conference & Expo**. This year's conference will be held in New York City from September 23 to September 26. As a corporation, **SYS-CON** has committed to providing leading-edge information, training, and exhibits in a number of areas that we serve via publications. In addition to the JavaEdge conference, we've also scheduled conferences for XML (XMLEdge2001), wireless (WirelessEdge2002), Linux (LinuxEdge2002), and ColdFusion (ColdFusionEdge2002). It's our plan to offer developers, managers, and businesspeople additional choices and opportunities to meet vendors, understand technology, network with their peers, and leverage the resources of the entire Java community.

This year I have the honor of serving as technical chairman for the JavaEdge2001 conference. We've decided to present five session tracks designed to maximize your ability to learn and leverage. The tracks are *Wireless Java and J2ME*, *J2SE*, *J2EE*, *Working with I-Technology*, and *Practical Business Solutions*. Obviously, the first three tracks strongly mirror the way Java itself is structured.

With the explosion of cell phones, the growing presence of networked PDAs, and an increase in the sophistication of non-PC devices, J2ME is positioned to become an integral part of the post-PC world. The *J2ME* track will explore wireless technology, including WAP and other protocols. We'll also have sessions on some of the KVMs available, and information on porting applications to small devices. Important issues such as memory constraints and display limitations will also be covered.

The *J2SE* track will focus on application development using Java. There will be some overlap with *J2EE*, but this track will focus on aspects of Java programming that are more pertinent to pure Java applications. Topics will include AWT and Swing, 2D and 3D programming, speech and telephony, mail, and that dreaded topic – multithreaded programming.

Our *J2EE* track will focus on the core technologies of the Enterprise Edition – EJB, JSP, JMS, JDBC, and so on. In particular, we'll cover the latest EJB standards, patterns for EJB design, deployment issues, and JMS.

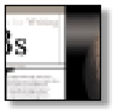
The fourth track, *Working with I-Technology*, will focus on the processes and procedures of software engineering using Java. Topics such as "UML," "Object Modeling," "Project Planning," and "Testing and Deploying Applications" will enlighten managers and senior technologists.

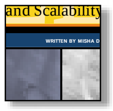
Practical Business Solutions, our final track, showcases the use of Java in industry and the availability of products and solutions along any number of lines. A variety of vendors will present on topics ranging from "Financial Services" and "CRM" to "Supply Chain." Also included will be Java solution vendors for subjects like "Content Management," "Personalization," and "Portals."

In each track we plan to have a mix of several experience levels in order to accommodate the beginner, the experienced developer, the expert, and the project manager or CIO.

The conference will also have an exhibit floor, where you can interact with numerous vendors to discuss their products and solutions in a traditional setting. We're looking forward to an authors day, where we'll have the authors of a number of current Java books, as well as our own authors, available to meet with you and sign autographs.

As the technology chair, I'm interested in what *you* have to say. The reality is, this is *your* conference, and I want to make sure you get the most you can out of it. We have a call for papers open at the moment on our Web site (www.sys-con.com/javaedge/papers.cfm) where industry professionals can submit possible topics. We're also actively working with a variety of sources to line up the most appropriate content for the conference. But I want to know what you, our community, need. Please drop me a line with your suggestions at sean@sys-con.com. See you on the Edge.





sean@sys-con.com

AUTHOR BIO
Sean Rhody is the founding editor of Java Developer's Journal. *He is also a respected industry expert and a consultant with a leading Internet service company.*

ALAN WILLIAMSON, EDITOR-IN-CHIEF

Shouldn't It Be dot.org Instead?

I'm sitting here cross-legged on a fresh San Francisco afternoon, 8,000 miles away from my family, wondering how the industry we live and breath in is shaping up. When I left Scotland, it was the weekend and the American markets didn't have too much to report. After a 9-hour flight, lane jumping on Highway 101, I arrived at my apartment to discover the news is raging about how the markets have taken a serious downturn. How tech stocks have gone way out of favor, with even the blue chips struggling. Should we worry? This is a question people keep asking me, and to be honest my answer up to now has been: "Nah, things are okay for us Java dudes." But are they?

I'm over here on a pilgrimage, you might say. I'm here to meet and talk with some of the makers and shakers of the Java universe. I want to know what they think. Is it media hype or is there indeed a little fire where the smoke is? Sometimes a good sign of how well an industry is doing is to take a look at what the recruitment world is up to. I had a chat with some of these people and they told me that it's business as usual, except the high packages on offer aren't quite the same as they used to be in the good old days. I chortled at this, and inquired what period were we referring to when we mentioned "good old days." Only 12 months ago was the answer. Damn, this industry moves fast…

While I stared out at the Golden Gate Bridge I had a wee thought regarding the term *dot-com*. If you remember, the initial domain structure was to have profit-making companies use the .com domain whereas nonprofit companies had to use the .org name. I think some companies perhaps indulged in a little wishful thinking…maybe we should be pointing our browsers toward amazon.org instead!

But don't panic just yet. By all accounts we still have a major shortage of people who actually can do a day's development as opposed to those who just think they can. I'll have more to report next month once I've had my chats with various people around the Bay area. If indeed there is writing on the wall, I'll find it!

As you know, we're pushing toward a new release of your beloved ***Java Developer's Journal*** with JDJ2.0 penciled in for the JavaOne issue. At this moment in time the team and I are working very hard to collate all the materials required to make this happen. One of the first things I set about doing was to assemble an Advisory Panel. Its purpose will be to steer the magazine through the changes and ensure that we're offering content that is relevant. And from time to time we'll peer into a crystal ball to see what's around the corner.

Each month we'll publish the remarks from the Advisory Panel and invite you all to give your input. I'm proud to say that the Panel has now been chosen and a dialog has begun.

Success is so much easier to achieve when you have a good team around you, and I am fortunate enough to have a great team with me.

- First and foremost I'd like to introduce you to our editorial director, **Jeremy Geelan**. Jeremy is a constant inspiration to me, and keeping up with this man's thought processes is a full-time job.
- A name that's already familiar to the readers of ***JDJ*** is, of course, **Ajit Sagar**. Ajit, the new J2EE editor, is working on building the J2EE content for JDJ2.0 – let me tell you, it's in great hands.
- A name that's virtually synonymous with "product review" is **Jim Milbery**. As our new, official product editor, Jim will coordinate all the reviews for ***JDJ***.

I'll introduce the rest of the people on the team next month. To do so now would make this look more like an Oscar ceremony than an editorial.

• • •

There is, of course, one other member of the team I haven't mentioned: *you*, the most important member. Without your input we're merely killing time here. So make yourself heard. Join our mailing list at http://myjdj.sys-con.com/mailman/listinfo/myjdj and let us know what you think.

Your ***JDJ*** needs you.

alan@sys-con.com

AUTHOR BIO
Alan Williamson holds the reins at n-ary (consulting) Ltd, one of the first companies in the UK to specialize in Java at the server side.

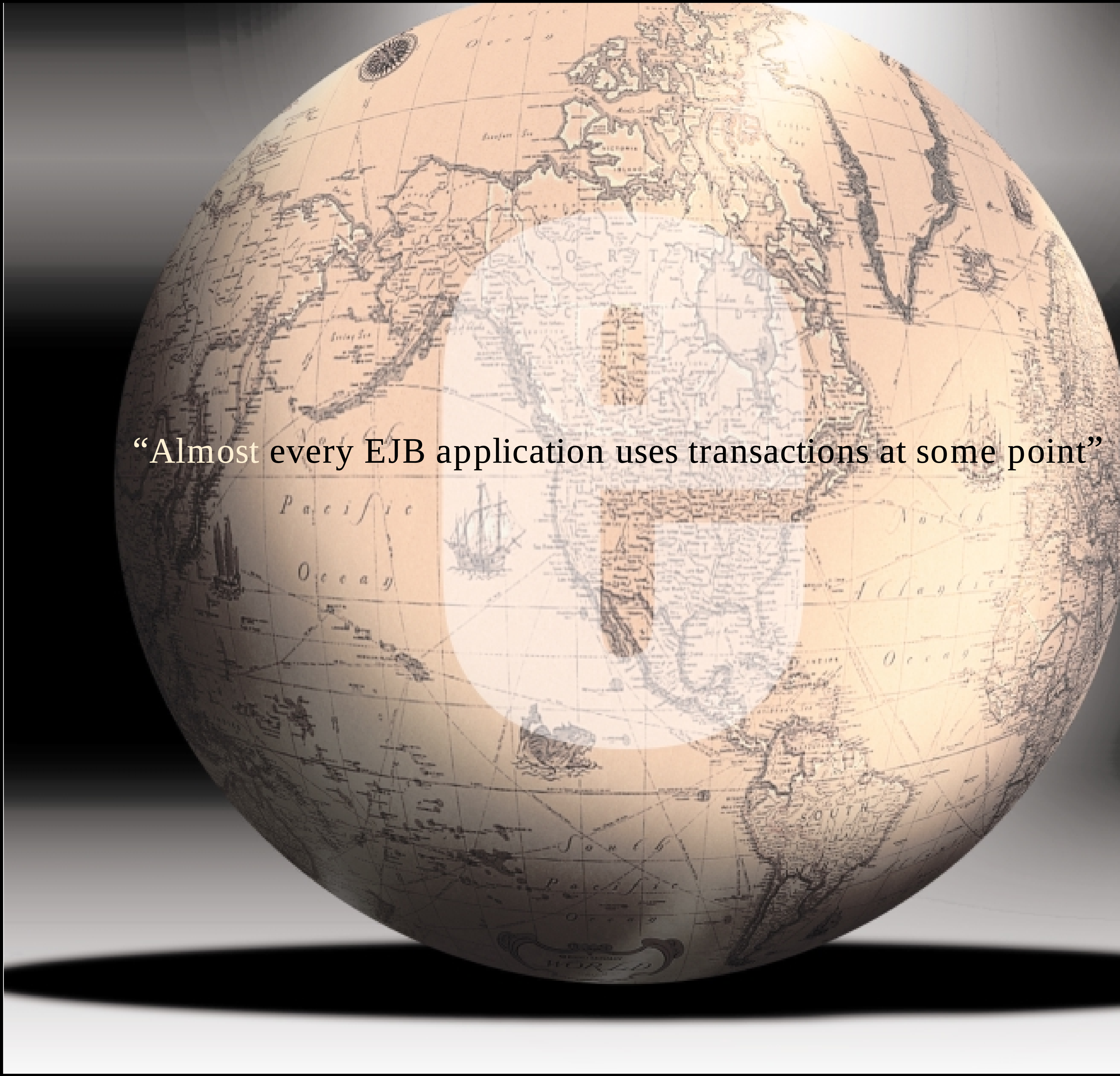
Best Practices for Writing

EJBs

Written by Sandra L. Emerson, Michael Girdley & Rob Woollen

If you write EJBs for app servers, these best practices are for you

JavaSoft defined the Enterprise JavaBeans specification to give Java developers a foundation for building distributed business components. EJBs are Java components that implement business logic and follow a contract designated in the EJB specification. Enterprise JavaBeans live inside an EJB container that provides a set of standard services, including transactions, persistence, security, and concurrency. This means that the application programmer is freed from developing these services from scratch.



"Almost every EJB application uses transactions at some point"

To get the most out of using EJBs in an enterprise-level distributed application supported by a J2EE-compliant application server, programmers should:
- Closely conform to the EJB specification.
- Use available tools for bean development and compliance checking.
- Learn to benefit from the experiences of others.

As outlined in this article, best practices for EJBs are based on experience gained through implementing J2EE for BEA WebLogic Server and helping BEA's customers develop or migrate their multitier applications to the J2EE platform.

EJB Overview

There are four types of Enterprise JavaBeans in EJB 2.0:
1. ***Stateless session beans:*** Provide a service without storing a conversation state between method calls.
2. ***Stateful session beans:*** Maintain state; each instance is associated with a particular client.
3. ***Entity beans:*** Represent an object view of persistent data, usually rows in a database. They have a primary key as a unique identifier. There are two operational styles for entity beans: *container-managed persistence* (CMP) and *bean-managed persistence* (BMP).
4. ***Message-driven beans:*** Added in EJB 2.0. These EJBs, the integration between JMS (Java Message Service) and EJB, are used to perform asynchronous work within the server.

Best Practices

For your coding pleasure…here are selected best practices for creating EJBs for an enterprise-level distributed application supported by a J2EE-compliant application server such as the BEA WebLogic Server. Our assumption is that the developer is an intermediate-to-expert Java programmer who's familiar with writing EJBs for an application server. We present best practices for:
- Transactions
- EJB security
- Creating a primary key class
- When not to use stateful session beans
- Coding business interfaces
- Using message-driven beans in transactions

Best practices are shown in summary form as tinted text following the relevant section.

Tips for Transactions

Almost every EJB application uses transactions at some point. Transactions ensure correctness and reliability and are essential to e-commerce applications. Misuse, however, can affect performance and even produce incorrect results. It's important to understand that session beans themselves can't be transactional.

A common misperception is that the member variables of the session bean will be rolled back when their transaction aborts. Instead, session beans merely propagate their transaction to any resource they acquire. For instance, at the beginning of a transaction a session bean has a member variable with a value of zero. During the transaction, the member variable is set to two, and a row is inserted into the database. If the transaction rolls back, the member variable won't be reset to zero. However, the row will no longer be in the database. Since a database is a transactional resource, it will participate in the session bean's transaction, and a rollback will abort any associated work.

Session bean state is not transactional.

This article is adapted from Chapters 8–10 of the forthcoming book *J2EE Applications and BEA WebLogic Server*, to be published by Prentice Hall.

User Interactions and Transaction Performance

Transactions can cause performance problems if they span too many operations. In particular, a transaction should never encompass user input or user think time. If a user starts a transaction and then goes to lunch or even visits another Web site, the transaction isn't committed. Instead, it continues to hold valuable locks and resources within the server.

In general, all transaction demarcation should occur within the server. There are a number of well-known techniques to avoid keeping long-running transactions open. When you ask a user to submit a form, break up the operation into two transactions. The Web page should read the data in a single transaction along with a version stamp. This transaction is committed before the form is returned to the user. The user can now modify the data as desired. The form update occurs in a new transaction.

Transactions should never encompass user input or think time.

Container-Managed vs Bean-Managed Transactions for Entity Beans

The EJB specification allows a session bean to choose either container-managed or bean-managed transactions. In the former the bean writer declares transaction attributes in the deployment descriptor. The EJB container then automatically starts and commits transactions as requested. The bean writer doesn't have to write any code to manage transactions. In the latter the bean writer uses the user transaction interface to explicitly start and commit transactions. Container-managed transactions should always be the bean writer's first choice. In bean-managed transactions the bean writer must ensure that the transaction is committed or rolled back. While the BEA WebLogic Server includes a transaction timeout, the bean writer shouldn't rely on this feature, but release transaction resources as soon as possible. In the case of container-managed transactions this is handled automatically by the EJB container.

Use container- instead of bean-managed transactions.

Best Practices for EJB Security

EJB provides a declarative security support as well as a simple programmatic interface for explicit security checks within the bean code. In practice, EJB security settings need to be considered within the entire application's security model. It's common for Web-based applications to handle authentication within the Web tier. In this environment the EJB tier may contain few security constraints. This arrangement simplifies the EJB design, and since the security checks are localized to the presentation layer, the application may modify security policies without modifying the EJB tier.

Applications with stand-alone programmatic clients often access session beans directly. Since there is no intermediate tier, the security access control must be handled in the EJB tier.

Declarative security control is preferred for simple applications. Since the security constraints are declared in the deployment descriptor, the bean classes' business logic is not polluted with security checks. The declarative security model is based on security roles that are declared in the deployment descriptor. Declarative security works best when the number of roles is fixed and doesn't depend on the number of clients. For instance, an application might include a user role and an administrator role. Since there are only two access domains, it's feasible to declare these roles in the deployment descriptor.

However, declarative security shouldn't be used when each user requires individual security constraints. Such applications require programmatic security checks within the EJB code. It's also common to combine both security models. For instance, an Account bean may use declarative security to ensure that only registered users access any methods. The bean code then includes additional constraints to ensure that each user gains access only to his or her account.

Use declarative security checks when an application contains few roles.
Choose programmatic security when each user needs individual security checks.

How to Write a Primary Key Class for Entity Beans

The EJB primary key class serves as its unique identifier both in the persistent store and in the EJB container. Usually, the primary key class fields map directly to the primary key fields in a database. If the primary key is only a single entity bean field that is a Java primitive class (such as java.lang.String), the bean writer doesn't have to write a custom primary key class. Instead, in the deployment descriptor, the bean writer specifies the name of the class and the name of the primary key field.

If the primary key maps to a user-defined type or to multiple fields, the bean writer must write a custom primary key class. The class must implement java.io.Serializable and contain the primary key fields. For CMP entity beans the field names must match the corresponding primary key field names in the bean class. This allows the EJB container to assign the appropriate CMP fields to their corresponding fields in the primary key class.

For instance, we might define an employee's primary key as a compound key using the first name, last name, and office number. Our compound key would look like Listing 1.

The primary key class consists of the primary key fields, which must be public, and a no argument constructor. The class must also implement the hashCode and equals methods. The EJB container uses a number of data structures internally, many of which are indexed by the primary key class. It is vital that hashCode and equals be implemented correctly and efficiently in the class.

The hashCode method is implemented by returning an integer using the primary key fields. The goal of this function is to produce an integer that can be used to index tables. The hashCode for a primary key should never change. Therefore, the hashCode should be constructed only from immutable values.

A common strategy is to XOR the hashCode of the primary key elements together. OR should never be used, since ORing several values will generally have most or all bits set to 1. Similarly, AND should not be used since most or all bits will converge to 0.

The hashCode method must be implemented such that two equal objects have the same hashCode. However, two objects with the same hashCode aren't necessarily equal. This hashCode implementation stores the hashCode in a member variable to avoid computing it every time hashCode is called.

It can also be tricky to implement equals correctly. The first line of any equals method should check the passed reference against this. This optimization simply checks whether equals has been called against itself. While this sounds strange at first, it is a common operation when the container has a primary key object and is checking to see if it already exists in a data structure.

Next, the equals method should ensure that the passed parameter is its own type. If the primary key class is final, a simple instanceof check can be used. If the primary key class isn't final, the passed parameter might be a subclass of our primary key class. In this case the equals method must use getClass().equals to ensure that the class types match exactly. It is recommended that primary key classes be final, since using instanceof is cheaper than comparing classes.

Primary key classes should be final.

When Not to Use Stateful Session Beans

Stateful session beans represent a stateful conversation between a single client and a bean instance. Stateful session beans can't be shared between multiple users. You shouldn't model a shared cache or any shared resource as a stateful session bean. If multiple clients need to access a single EJB instance, use an entity bean.

Stateful session beans are not shared by multiple users.

Since each client requires its own stateful session bean instance, the number of bean instances and the associated resource requirements can grow quickly. If an application can tolerate the stateless programming model, stateless session beans are easier to scale than stateful session beans.

Applications should always call *remove* after finishing with a stateful session bean instance. This allows the EJB container to release container resources as soon as possible. If the remove call is omitted, the EJB container will eventually passivate the bean, but this involves extra disk access.

Stateless session beans are easier to scale than stateful session beans.

Stateful session bean writers must also be careful when integrating their stateful session beans with Web applications. These beans should not allow concurrent method calls. As mentioned earlier, it's possible for multiple requests to cause concurrent calls on a stateful session bean. Unfortunately, this error usually shows up under load, so it's often missed in testing. For this reason use stateful session beans only within the scope of a request. Use entity beans or servlet sessions for applications that need to store data between requests.

Coding Business Interfaces

Many new EJB programmers are confused by the relationship between the remote interface and the EJB class. This arrangement is necessary for the container to intercept all method calls to the EJB. One confusing aspect is that the EJB class implements the methods defined in the remote interface, but the EJB class doesn't implement the remote interface itself. In fact, the EJB class should *never* implement the remote interface. While the EJB specification allows this practice, it can cause serious but subtle bugs. The problem with having the EJB class implement the remote interface is that the class can now be passed as a parameter to any method that expects the remote interface as a parameter.

Remember that the remote interface exists to allow the container to intercept method calls in order to provide necessary services such as transactions or security. If the Bean class is used, the method calls arrive directly on the bean object – creating a dangerous situation in which the container cannot intercept method calls or intervene in case of error. If (as recommended) the EJB class doesn't implement the remote interface, this problem becomes apparent at compile time. The Java compiler will reject the attempt to pass the bean class as a parameter of the remote interface's type.

Never implement the remote interface in the EJB class.

–continued on page 16

Listing 1: Defining a compound primary key

```
public final class EmployeePK implements java.io.Serializable {

  public String lastName;
  public String firstName;
  public int officeNumber;

  private int hash = -1;

  public EmployeePK() {}

  public int hashCode() {
    if (hash == -1) {
      hash = lastName.hashCode() ^ firstName.hashCode()
       ^ officeNumber;
    }
    return hash;
  }
  public boolean equals(Object o) {
    if (o == this) return true;

    if (o instanceof EmployeePK) {
      EmployeePK other = (EmployeePK) o;
      return other.hashCode() == hashCode() &&
             other.officeNumber == officeNumber &&
             other.lastName.equals(lastName) &&
             other.firstName.equals(firstName);

    } else {
      return false;
    }
  }
}
```

Download the Code!
www.JavaDevelopersJournal.com

The WebLogic Server provides an EJB compliance checker to catch any methods that are defined in the remote interface but not implemented in the EJB class.

Using Transactions with Message-Driven Beans

Message-driven EJBs are the integration between EJBs and the JMS. Like other EJB types, message-driven EJBs live within an EJB container and benefit from EJB container services such as transactions, security, and concurrency control. However, a message-driven EJB doesn't interact directly with clients. Instead, message-driven EJBs are JMS Message Listeners. A client publishes messages to a JMS destination. The JMS provider and the EJB container then cooperate to deliver the message to the message-driven EJB.

Like other EJBs, message-driven beans make use of the EJB container's transaction service. Since these beans never interact directly with clients, they never participate in the client's transaction.

Like session beans, message-driven EJBs may specify either bean-managed or container-demarcated transactions in their ejb-jar.xml deployment descriptor. With container transactions a message-driven EJB may specify either Required or NotSupported. (Since there is no client transaction, there is no reason to support the other transaction attributes.) If the transaction attribute is NotSupported, the message-driven EJB will not participate in a transaction.

If the Required attribute is specified, the EJB container automatically starts a transaction. The message receipt from the JMS Queue or Topic is included in this transaction. The message-driven bean's onMessage method is then called in the transaction context. When the onMessage method returns, the EJB container commits the transaction. If the transaction aborts, the JMS message remains in the JMS destination and is delivered again to the message-driven EJB.

Error Handling in Message-Driven Bean Transactions

Message-driven beans with the Required transaction attribute need to be careful when aborting transactions. A transaction aborts either because it was explicitly marked for rollback or because a system exception was thrown. One potential issue is known as the "Poison Message." In this scenario a message-driven EJB receiving stock trade orders from an order queue might encounter a stock symbol that doesn't exist. When the message-driven EJB receives the error message, the underlying logic might be to abort the transaction because the symbol is invalid. When the JMS implementation delivers the message again in a new transaction, the process repeats. Clearly, this is not the desired behavior.

A good solution for this potential problem is to separate application errors from system errors. An application error, such as an invalid stock symbol, should be handled by sending an error message to an error JMS destination. This allows the transaction to commit, and the "Poison Message" leaves the system. A system error might be that the back-end database has failed. In this case our transaction should roll back so that this message is still on the queue when the database recovers.

Conclusion

EJBs are server-side Java components that leverage the standard transaction, persistence, concurrency, and security services provided by the EJB container. What we've described here are best practices for coding to the J2EE standard, based on experience with the BEA WebLogic Server.

AUTHOR BIOS

Sandra L. Emerson is a technical writer and consultant with 20 years' experience in the software industry. She is a coauthor of The Practical SQL Handbook *(Addison-Wesley).*

Michael Girdley is senior product manager for BEA WebLogic Server at BEA Systems, Inc. He has extensive experience programming with Java, HTML, C, and C++, and is a coauthor of Web Programming with Java *(SAMS);* Java Unleashed, 2nd ed. *(SAMS); and* Java Programming for Dummies *(IDG Books).*

Rob Woollen is a senior software engineer at BEA Systems, Inc. He is currently the lead developer for the BEA WebLogic Server EJB Container and is a coauthor of the forthcoming book J2EE Applications and BEA WebLogic Server *(Prentice Hall).*

semerson@igc.org

michaelg@bea.com

rwoollen@bea.com

Guaranteed Messaging with JMS

How to make sure your business-critical data reaches its destination

WRITTEN BY
DAVID CHAPPELL &
RICHARD MONSON-HAEFEL

Part 2 of 3

The notion of guaranteed delivery of Java Message Service messages has been lightly touched on in other recently published articles on JMS. But what really makes a JMS message "guaranteed"? Should you just take it on faith, or would you like to know what's behind it?

This article answers these questions via a detailed discussion of message persistence, internal acknowledgment rules, and message redelivery. Using excerpts condensed from the book we coauthored, *Java Message Service*, we'll explain how JMS guaranteed messaging works – including once-and-only-once delivery semantics, durable subscriptions, failure and recovery scenarios, and transacted messages.

JMS Guaranteed Messaging

There are three key parts to guaranteed messaging: message autonomy, store-and-forward, and the underlying message acknowledgment semantics. Before we discuss these parts, we need to review and define some new terms. A JMS client application uses the JMS API. Each JMS vendor provides an implementation of the JMS API on the client, which we call the *client runtime*. In addition to the client runtime, the JMS vendor provides some kind of message "server" that implements the routing and delivery of messages. The client runtime and the message server are collectively referred to as the *JMS provider*.

This article is adapted from *Java Message Service* by Richard Monson-Haefel and David Chappell (O'Reilly) and appears here by permission of the publisher.

"Provider failure" refers to any failure condition that is outside the domain of the application code. It could mean a hardware failure that occurs while the provider is entrusted with the processing of a message, an unexpected exception, the abnormal end of a process due to a software defect, or network failures.

Message Autonomy

Messages are self-contained autonomous entities. A message may be sent and re-sent many times across multiple processes throughout its lifetime. Each JMS client along the way will consume the message, examine it, execute business logic, modify it, or create new messages in order to accomplish the task at hand.

Once a JMS client sends a message, its role is completed. The JMS provider guarantees that any other interested parties will receive the message. This contract between a sending JMS client and the JMS provider is much like the contract between a JDBC client and a database. Once the data is delivered, it is considered "safe" and out of the hands of the client.

Store-and-Forward Messaging

When messages are marked persistent, it is the responsibility of the JMS provider to utilize a store-and-forward mechanism to fulfill its contract with the sender. The storage mechanism is used for persisting messages to disk to ensure that the message can be recovered in the event of a provider failure or a failure of the consuming client. The forwarding mechanism is responsible for retrieving messages from storage, and subsequently routing and delivering them.

Message Acknowledgments

The message acknowledgment protocol is key to guaranteed messaging, and support for acknowledgment is required by the semantics of the JMS API. The acknowledgment protocol allows the JMS provider to monitor the progress of a message so that it knows whether the message was successfully produced and consumed. With this knowledge the JMS provider can manage the distribution of messages and guarantee their delivery.

The acknowledgment mode is set on a JMS session when it is created, as indicated below:

```
tSession =
  tConnect.createTopicSession(false,
Session.CLIENT_ACKNOWLEDGE);

qSession =
  qConnect.createQueueSession(false,
Session.DUPS_OK_ACKNOWLEDGE);
```

AUTO_ACKNOWLEDGE

Let's start by examining the

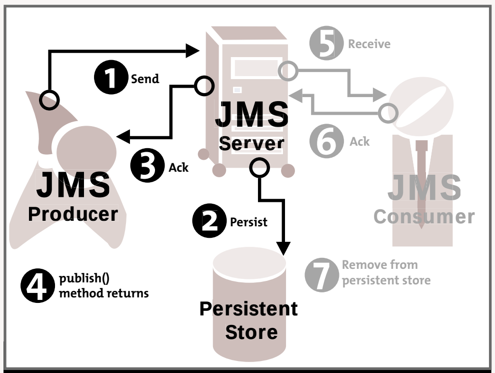


FIGURE 1 Send and Receive are separate operations.

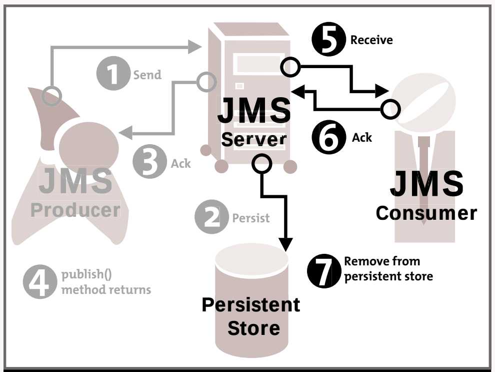


FIGURE 2 Message removed from persistent store

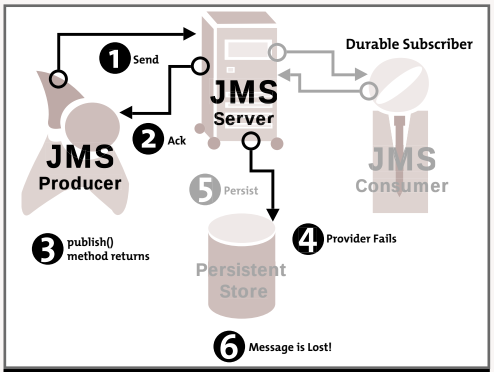


FIGURE 3 When a nonpersistent message can be lost.

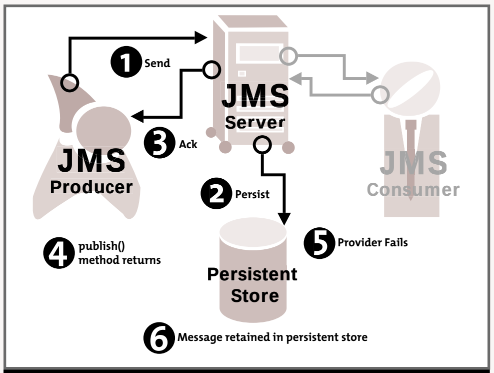


FIGURE 4 Persistent messages won't be lost during a provider failure.

AUTO_ACKNOWLEDGE mode. In the loosely coupled asynchronous environment of JMS, senders and receivers (producers and consumers) are intentionally decoupled from each other. Hence, the roles and responsibilities are divided among the message producer, the message server, and the message consumer.

The Producer's Perspective

Under the covers, the TopicPublisher.publish() or QueueSender.send() methods are synchronous. These methods are responsible for sending the message and blocking until an acknowledgment is received from the message server. If a failure condition occurs during this operation, an exception is thrown to the sending application and the message is considered undelivered.

The Server's Perspective

The acknowledgment sent to the producer (sender) from the server means that the server has received the message and has accepted responsibility for delivering it. From the JMS server's perspective, the acknowledgment sent to the producer is not tied directly to the delivery of the message. They are logically two separate steps. For persistent messages, the server writes the message out to disk (the store part of store-and-forward), then acknowledges to the producer that the message was received (see Figure 1). For nonpersistent messages this means the server may acknowledge to the sender as soon as it has received the message and has the message in memory. If there are no subscribers for the message's topic, the message may be discarded.

In a publish/subscribe model, the message server delivers a copy of a message to each of the subscribers. For durable subscriber, the message server doesn't consider a message fully delivered until it has received an acknowledgment from all of the message's intended recipients. It knows on a per-consumer basis which clients have received each message and which have not.

Once the message server has delivered the message to all of its known subscribers and has received acknowledgments from each of them, the message is removed from its persistent store (see Figure 2).

If the subscriptions are durable and the subscribers aren't currently connected, the message will be held by the message server until either the subscriber becomes available or the message expires. This is true even for nonpersistent messages. If a nonpersistent message is intended for a disconnected durable subscriber, the message server saves the message to disk as though it were a persistent message. In this case the difference between persistent and nonpersistent messages is subtle, but very important. For nonpersistent messages the JMS provider could fail before it's had a chance to write the message out to disk on behalf of the disconnected durable subscribers. Messages may be lost (see Figure 3).

With persistent messages a provider may fail and recover gracefully, as illustrated in Figures 4 and 5. Since the messages are held in persistent storage, they're not lost and will be delivered to consumers when the provider starts up again. If the messages are sent using a p2p queue, they're guaranteed to be delivered. If the messages were sent via publish/subscribe, they're guaranteed to be delivered only if the consumers' subscriptions are durable.

The Consumer's Perspective

There are also rules governing acknowledgments and failure conditions from the consumer's perspective. If the session is in AUTO_ACKNOWLEDGE mode, the JMS provider's client runtime must automatically send an acknowledgment to the server as each consumer gets the message. If the server doesn't receive this acknowledgment, it considers the message undelivered and may attempt redelivery.

Message Redelivery

The message may be lost if the provider fails while delivering a message to a consumer with a nondurable subscription. If a durable subscriber receives a message, and a failure occurs before the acknowledgment is returned to the provider (see Figure 6), the JMS provider considers the message undelivered and will attempt to redeliver it (see Figure 7). In this case the "once-and-only-once" requirement is in doubt. The consumer may receive the message again, because when delivery is guaranteed, it's better to risk delivering a message twice than to risk losing it entirely. A redelivered message will have the JMSRedelivered flag set. A client application can check this flag by calling the getJMSRedelivered() method on the Message object. Only the most recent message received is subject to this ambiguity.

Point-to-Point Queues

For point-to-point queues, messages are marked by the producer as either persistent or nonpersistent. If the former, they are written to disk and subject

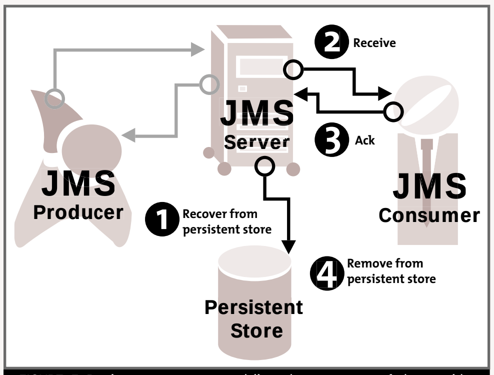


FIGURE 5 Persistent messages are delivered on recovery of the provider.

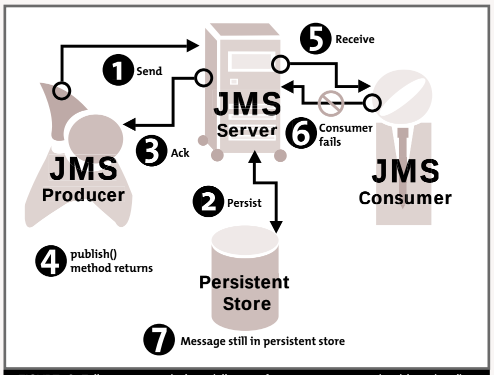


FIGURE 6 Failure occurs during delivery of a message to a durable subscriber.

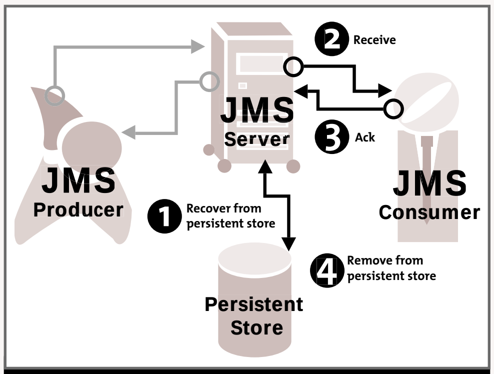


FIGURE 7 Durable subscriber recovers

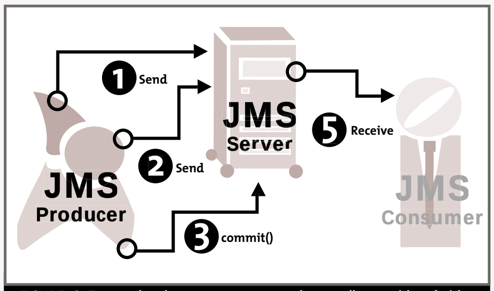


FIGURE 8 Transactional messages are sent in an all-or-nothing fashion.

to the same acknowledgment rules, failure conditions, and recovery as persistent messages in the publish/subscribe model.

From the receiver's perspective the rules are somewhat simpler since only one consumer can receive a particular instance of a message. A message stays in a queue until it is delivered to a consumer or expires. This is somewhat analogous to a durable subscriber in that a receiver can be disconnected while the message is being produced without losing the message. If the messages are nonpersistent they aren't guaranteed to survive a provider failure.

DUPS_OK_ACKNOWLEDGE

Specifying the DUPS_OK_ACKNOWLEDGE mode on a session instructs the JMS provider that it is okay to send a message more than once to the same destination. This is different from the once-and-only-once or the at-most-once delivery semantics of AUTO_ACKNOWLEDGE. The DUPS_OK_ACKNOWLEDGE delivery mode is based on the assumption that the processing necessary to ensure once-and-only-once delivery incurs extra overhead and hinders performance and throughput of messages at the provider level. An application that is tolerant of receiving duplicate messages can use the DUPS_OK_ACKNOWLEDGE mode to avoid incurring this overhead.

In practice, the performance improvement you gain from DUPS_OK_ACKNOWLEDGE may be something you want to measure before designing your application around it.

CLIENT_ACKNOWLEDGE

With AUTO_ACKNOWLEDGE mode the acknowledgment is always the last thing to happen implicitly after the onMessage() handler returns. The client receiving the messages can get finer-grained control over the delivery of guaranteed messages by specifying the CLIENT_ACKNOWLEDGE mode on the consuming session.

The use of CLIENT_ACKNOWLEDGE allows the application to control when the acknowledgment is sent. For example, an application can acknowledge a message – thereby relieving the JMS provider of its duty – and perform further processing of the data represented by the message. The key to this is the acknowledge() method on the Message object, as shown in Listing 1.

The acknowledge() method informs the JMS provider that the message has been successfully received by the consumer. This method throws an exception to the client if a provider failure occurs during the acknowledgment process. The provider failure results in the message being retained by the JMS server for redelivery.

Transacted Messages

JMS transactions follow the convention of separating the send operations from the receive operations. Figure 8 shows a transactional send in which a group of messages are guaranteed to get to the message server, or none of them will. From the sender's perspective the messages are cached by the JMS provider until a commit() is issued. If a failure occurs or a rollback() is issued, the messages are discarded. Messages delivered to the message server in a transaction are not forwarded to the consumers until the producer commits the transaction.

The JMS provider won't start delivery of the messages to its consumers until the producer has issued a commit() on the session. The scope of a JMS transaction can include any number of messages.

JMS also supports transactional receives, in which a group of transacted messages are received by the consumer on an all-or-nothing basis (see Figure 9). From the transacted receiver's perspective the messages are delivered to it as expeditiously as possible, yet they are held by the JMS provider until the receiver issues a commit() on the session object. If a failure occurs or a rollback() is issued, the provider will attempt to redeliver the messages, in which case the messages will have the redelivered flag set.

Transacted producers and transacted consumers can be grouped together in a single transaction if they are created from the same session object, as shown in Figure 10. This allows a JMS client to produce and consume messages as a single unit of work. If the transaction is rolled back, the messages produced within the transaction won't be delivered by the JMS provider. The messages consumed within the same transaction won't be acknowledged and will be redelivered.

Unless you're doing a synchronous request/reply, you should avoid grouping a send followed by an asynchronous receive within a transaction. There could be a long interval between the time a message is sent and the related message is asynchronously received, depending on failures or downtime of other processes involved. It's more practical to group the receipt of a message followed by the send of another message.

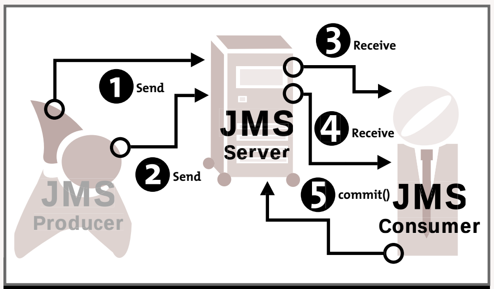


FIGURE 9 Message can be received on all-or-nothing basis.

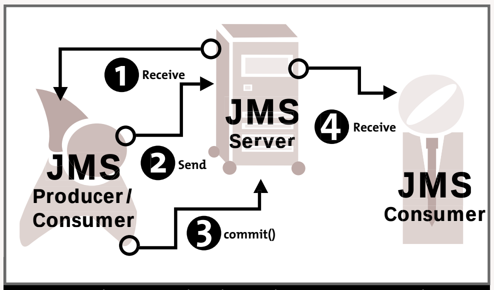


FIGURE 10 When transacted producer and consumers are grouped as one.

Creating a JMS Transaction

Enabling a JMS transacted session happens as part of creating a Session object, as shown in Listing 2.

The first parameter of createTopicSession() or createQueueSession() method is a Boolean indicating whether this is a transacted session. That is all we need to create a transactional session. There is no explicit begin() method. When a session is transacted, all messages sent or received using that session are automatically grouped in a transaction. The transaction remains open until either a session.rollback() or a session.commit() happens, at which point a new transaction is started. An additional Session method, isTransacted(), returns a Boolean true or false indicating whether the current session is transactional.

Distributed Transactions

Having all producers and all consumers participate in one global transaction would defeat the purpose of using a loosely coupled asynchronous messaging environment.

Sometimes it's necessary to coordinate the send or receipt of a JMS transaction with the update of another non-JMS resource, like a database or an EJB entity bean. This typically involves an underlying transaction manager that takes care of coordinating the prepare, commit, or rollback of each resource participating in the transaction. JMS provides JTA transaction interfaces for accomplishing this.

JMS providers that implement the JTA XA APIs can participate as a resource in a two-phase commit. The JMS specification provides XA versions of the following JMS objects : XAConnectionFactory, XAQueueConnection, XAQueueConnectionFactory, XAQueueSession, XASession, XATopicConnection, XATopicConnectionFactory, and XATopicSesion.

Conclusion

Our discussion of message acknowledgment shows that producers and consumers have different perspectives on the messages they exchange. The producer has a contract with the message server that ensures that the message will be delivered as far as the server. The server has a contract with the consumer that the message will be delivered to it. The two operations are separate, which is a key benefit of asynchronous messaging. It is the role of the JMS provider to ensure that messages get to where they are supposed to go.

Through message acknowledgment, message persistence, and transactions, JMS provides a strict set of rules that guarantees that your business critical data will travel reliably throughout your global enterprise. *Note:* This material is condensed from our book, which contains full working examples with detailed explanations of the concepts presented here.

Author Bios

David Chappell is chief technology evangelist for Progress Software's SonicMQ and coauthor of O'Reilly's Java Message Service.

Richard Monson-Haefel is the author of Enterprise JavaBeans *and coauthor of* Java Message Service, *both from O'Reilly. He is also the lead architect of the OpenEJB server.*

chappell@progress.com

richard@monson-haefel.com

Listing 1: CLIENT_ACKNOWLEDGE mode requires an explicit acknowledge()

```
public void onMessage(javax.jms.Message message){
     int count = 1000;
     try{
     // perform some business logic with the message
          ...
          message.acknowledge();
          // perform more business logic with the message
          ...
     }catch (javax.jms.JMSException jmse){
          // catch the exception thrown and undo the results
          // of partial processing
          ...
     }
}
```

Listing 2: Creating transected jms session

```
// pub/sub connection creates a transacted TopicSession
javax.jms.TopicSession session
=connect.createTopicSession(true,Session.AUTO_ACKNOWLEDGE);
// p2p connection creates a transacted QueueSession
javax.jms.QueueSession =
          connect.createQueueSession(true,Session.AUTO_ACKNOWLEDGE);
      ...
}
```

Download the Code!
www.JavaDevelopersJournal.com

<from web sites to web services and wireless>

<written by phil costa>

how can app servers ease the transition?

During the past five years, application servers have emerged as a vital piece of the Web infrastructure. By providing a set of services common to all Web applications (e.g., state management, database connectivity) as well as a productive set of APIs or scripting languages, application servers have made building applications for the Web dramatically easier, not to mention more scalable and reliable.

As a result, businesses have been free to experiment with new ways of serving their customers and increasing their efficiency, resulting in turmoil and change unlike any seen since the industrial revolution.

Now, two new sets of technologies promise to have an equally powerful impact on the business and technology worlds. First, wireless standards such as WAP and i-mode are making it economical and practical to build real applications using mobile client devices such as phones and PDAs. This opens up new possibilities for applications that ensure employees, customers, or partners have access to information as well as the ability to act on it – whether they're at their desks or in their cars.

The second wave is popularly referred to as Web services, the ability to expose pieces of an application so that they can be invoked from almost anywhere on the Internet. Developers have been building network-based, distributed applications using technologies such as CORBA, DCOM, and Java RMI for years, but these have been restricted almost entirely to intracompany applications. The emergence of Internet-centric protocols such as the W3C's XML Protocol (also known as SOAP) will enable a new wave of intra- and intercompany integration as well as new business models for companies providing services that can be consumed (and hopefully paid for) by others.

As an example of how these new technologies might be used, let's look at an online travel service. Today, sites like Expedia or Travelocity allow customers to purchase tickets and look up arrival and departure times, all from within a Web browser. But much of the information they provide would be more useful if it were available away from the desk, such as from a phone. For instance, while visiting a client, many business travelers would like to be able to check the departure time of their flight or receive a page if the flight has been changed.

Similarly, an online travel site could gain additional revenue if it were able to resell its ability to book flights to others, such as a hotel site. Today, the cost and complexity of this type of integration has restricted it to a few large partnerships, but with the emergence of easily consumable Web services, it should be almost as common as syndicated content is today. This will open up new revenue streams and further change the balance of power between different types of businesses.

New Demands for Developers

From a business perspective, these changes are exciting but they also introduce a new level of complexity into the developer's life. Supporting two implementations of DHTML is difficult enough. Now, with Web services and wireless devices coming online, the development team also has to make sure the application they're building today can support the dozens of wireless devices as well as expose parts of its functionality as a Web service. For the online travel service, this means the code that checks a flight's estimated arrival time must be able to format its data so that it can be viewed in a PC browser, displayed on a mobile device, or passed to another application.

Fortunately, many of the same design principles that have served Web developers well in the past few years are also applicable to this new world – most important, a clean division between presentation logic and business logic. To a well-architected application, the addition of wireless clients or the creation of a Web service interface should involve little more than the addition of some new presentation logic specific to that client. Moreover, the investments developers have made in application servers will also carry forward, as the infrastructure services provided in the Java 2 Enterprise Edition (J2EE) specification and Microsoft's .NET architecture are vital to the successful construction of wireless applications and Web services.

That's the good news. The bad news is that application servers could still do a lot more to ease the transition to this new multiclient environment. For instance, even with a well-architected application, each new client requires additional code to handle the idiosyncrasies of managing state for a wireless browser (most of which don't support cookies) or of handling requests for Web services.

If this is really where we're headed (and I'm convinced it is), then application servers will have to add new capabilities to smooth the evolution from Web applications to wireless applications and Web services. After all, managing the kind of contextual information created by a new client technology was one of the main reasons application servers were developed in the first place.

In looking at these changes, I'll discuss wireless applications first, then move on to Web services. While both represent new types of clients, there are important differences. Finally, the last section will discuss how new capabilities of application servers should be exposed to developers in order to gain the greatest impact.

Supporting Wireless Applications

As wireless applications move into the mainstream, the first type of change we'll see in application servers is the addition of client brokering services. As the diversity of devices accessing the Web grows, application servers will need to be able to detect the type of client that's making a request and direct the request to the appropriate set of presentation services (i.e., pages that provide the formatting of data and handle things such as state management).

Thus, a browser request sent to www.mysite.com would receive the HTML home page, but a WAP phone would receive the WML home page and a Palm cHTML. This type of handler architecture is one that many Web developers have been forced to implement in order to support different browsers. Given the physical differences between a computer monitor and a PDA screen, the different home pages are unlikely to be mirrors of each other. However, forcing users to remember different URLs for each device is an unnecessary burden, especially for consumer sites.

<managing the kind of contextual information created by a new client technology was one of the main reasons application servers were developed in the first place>

The second type of change we'll see is more specific to the type of client being supported. In the case of wireless devices, the challenge will be to provide UI frameworks that help developers support the many different wireless devices on the market. Today, there's no standard form factor for the phone's display, not to mention multiple markup languages for displaying information. As a result, writing a wireless application often involves implementing a different set of presentation templates for each type of phone. In a world where new phones are introduced daily, this is quickly becoming unmanageable.

In response, a number of vendors have built frameworks that provide an abstraction layer above these differences. With most of these frameworks, a developer defines forms, menus, tables, and other UI components using XML or a visual layout tool. Then, at runtime, the framework uses a stylesheet, typically provided by the vendor, to map the model to the device making the request (see Figure 1).

The most advanced versions of these frameworks are available from start-ups such as AlterEgo Networks, iConverse, NetMorph, and many others. However, as wireless access becomes a common feature for many applications, these types of frameworks will likely be added to the application server as well. This way, their capabilities can be built directly into the development tools, and their services can be integrated with the others provided in the application server, such as security.

There are many other types of services that may become part of the application server as well, such as location services or support for synchronization protocols. However, the two discussed above are likely to be the most prevalent, not to mention the most fundamental.

Supporting Web Services

While the changes needed to support Web services are different from those required by wireless applications, they fall into the same two general categories: brokering services and frameworks.

Unlike with wireless devices, the benefit from a brokering service for Web services is not one of convenience. The likelihood of someone casually browsing a Web service is low. Instead, the main benefit lies in providing a clean separation between interface and implementation. Thus, a site that provides a Web service for storing data could change the back-end implementation, adding more servers or perhaps even outsourcing to another vendor, without affecting existing users (who will continue to see the same interface). Other

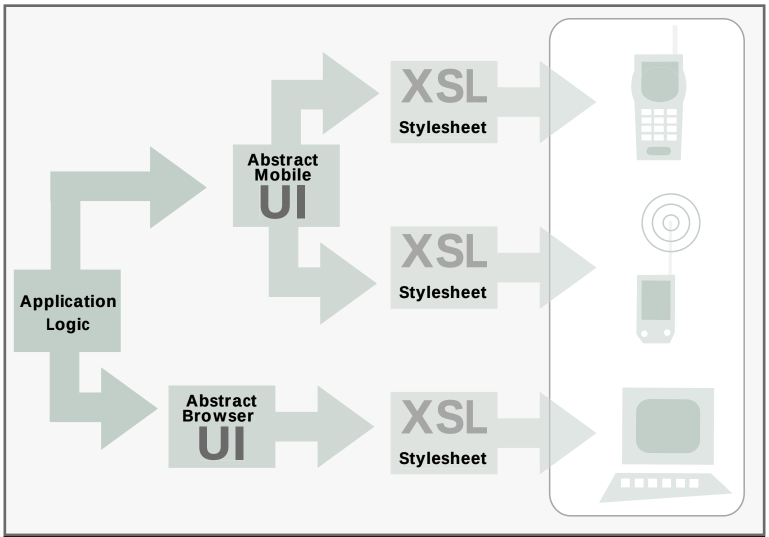


FIGURE 1 Managing differences in client size and capability will increase demand for abstraction frameworks

benefits of the separation between interface and implementation include a built-in mechanism for security and load balancing, among others. These are just a few of the reasons CORBA, DCOM, and RMI all use brokering services.

In contrast to UI frameworks, which are relatively immature and completely nonstandard, the frameworks for Web services are rapidly converging on a common set of technologies built around SOAP. However, like most standards, these technologies provide only the baseline technology required for interoperability. It will be up to application server and development tool vendors to provide mappings between the popular development languages and SOAP as well as tools that can automatically generate a Web service interface from a component method or an application function.

<the long-term effects of the move to a **web-services architecture** are likely to unfold for many years to come>

An early example of a such a framework is Allaire's Spectra, which can expose any part of a Web site's functionality as a Web service using HTTP and WDDX (an XML protocol for exchanging data between programming languages). Another example is IBM's Web Services Toolkit, which can generate SOAP interfaces from EJB interfaces, or Microsoft's Visual Basic.NET, which enables developers to build SOAP interfaces as easily as they can build Windows forms today.

These changes are only the tip of the iceberg. The long-term effects of the move to a Web-services architecture are likely to unfold for many years to come. Let's look at one further aspect of the application server – the development model itself.

The Application Server Development Model

Today, it's rapidly becoming apparent that two primary application architectures will compete for the hearts and minds of the development community: Microsoft's .NET and the J2EE specification.

While there are important differences between these models, both conform to the same design principle – the page and component model. In the Java world, this means JSPs and EJBs, while in the Microsoft world, this means ASPs and COM/COM+ components. Both .NET and J2EE provide powerful architectures because they offer a built-in separation of business and presentation logic, which, as we've seen, is important in enabling wireless computing as well as Web services.

The only problem is there are still very few people using both components and pages. If you look at any developer survey, the percentage using ASP or JSP versus COM or EJB is overwhelmingly in favor of ASP/JSP. This doesn't even take into account the thousands of developers using other technologies such as Perl, CFML, PHP, or other server-side scripting languages.

One could argue this is only because EJB and COM are relatively new, and that they'll enjoy much broader adoption as they reach maturity. However, I believe these technologies, while very powerful and absolutely necessary, will never enjoy the same broad adoption that page-based development models enjoy.

The reason is that in most cases EJB and COM+ force developers to shoulder a lot more responsibility than they really want. If you need to write transactional components and want to take advantage of the advanced persistence services of an EJB container, that's the way to go. But for most developers, writing in the page-based model will not only be sufficient, but also a more productive way to work.

Moreover, to assume that writing applications with a clean separation between presentation and business logic requires the use of EJB is to mistake implementation for design. J2EE and .NET are a set of APIs. The separation of presentation from business logic is a way to design an application. A well-designed application can be implemented in many different ways.

In fact, many developers using page-based scripting languages follow this design principle closely, building two layers of pages that communicate with each other or encapsulating business logic in custom tags in order to keep it separate from the presentation layer. Thus, any framework that creates Web services should be able to wrap a JSP or ColdFusion page with a service interface as easily as it can generate a stub for a method on a COM object or an EJB. Given their huge base of page-based developers, it's no surprise that both Microsoft and Allaire are pursuing this strategy.

The point of this discussion is not to downplay the importance of EJB or COM+, but to point to the third major change that must come to the application server world. In the past five years, application servers have come a great distance, providing greater scalability, integration, and reliability. Now, with the infrastructure pieces falling into place, the biggest challenge will be making the full power of that infrastructure accessible to the broadest range of developers.

In other words, if we're to convert the many visions of a networked future into reality, application servers will not only have to make the Internet development infrastructure (which includes Web pages, mobile devices, Web services, and whatever is next) richer, but faster and easier as well. The vendors that figure that out will make a lot of developers very happy.

AUTHOR BIO
Phil Costa is the senior manager of strategic marketing at Allaire Corporation. His responsibilities include marketing for the Allaire Business Platform as well as research into the future directions of the Internet software market.

pcosta@allaire.com

Next Month in *JDJ*. . .

Working with Dynamic XML Documents Using CORBA's XML/Value Mapping
By Jon Siegel

Effective Application Deployment with Embedded Java Technology
By Marc Erickson

Building Thread-Safe GUIs with Swing
by Neal Ford

Building a Telephone/Voice Portal with Java Part 1
By Kent V. Klinner III and Dale B. Walker

Growing JSP and Servlet Sites into EJB-Based Services
By Patrick Sean Neville

Book Review: *Database Programming with JDBC and Java* by George Reese
Reviewed by James McGovern

OMG's New Fault Tolerant CORBA Specification

Online redundancy can save the day

WRITTEN BY
JON SIEGEL

Any individual piece of computer hardware or software can fail. That's why we back up our hard drives. When the hard drive on my laptop failed last year, the tape backup got me up and running in a few days – the time it took to get a replacement drive and reload my files.

But some systems can't afford to be down for a few days…or even a few hours…or sometimes even a few minutes. For example:

- *Medical monitoring systems* deal with health-critical information constantly.
- *Fly-by-wire systems* must act in real time and must not fail (as you'll attest if you've ever flown in an airplane).
- When widely used *e-commerce systems* fail, companies lose thousands of sales, and possibly thousands of customers.
- *Financial trading applications* may lose unbelievable amounts of money for both customers and the responsible company if they go down at a critical moment – usually when the load is highest and potential for failure is greatest.

Online redundancy provides the timely robustness that meets these systems' needs. And to enable users of these systems to take advantage of CORBA, OMG members have recently standardized Fault Tolerant (FT) CORBA, which provides entity redundancy to CORBA systems.

The difference between running an application under normal conditions and under fault-tolerant conditions is simple: under normal conditions an application will fail occasionally. Under fault-tolerant conditions it should never fail. Every invocation by a client should produce a response with the correct result (subject to the limitations listed in the last section of this article).

OMG's FT CORBA specification enables products that support a wide range of fault tolerance from simple one-copy redundancy of key objects to highly reliable, geographically dispersed, multiply connected servers. FT CORBA applications are transparent to the user, to the operation of the client, and, to some extent, to the application programmer.

It's perhaps a little more surprising that the transparency extends partially to the application programmer who codes his or her usual CORBA program with the same set of objects that would have been coded for a nonfault-tolerant system, adding to each application object a few interfaces that let the FT infrastructure synchronize the state of each set of object replicas. Running this same code on reliable, redundant hardware under the control of a fault-tolerant infrastructure makes it fault tolerant.

The transparency doesn't extend to the system administrator, who must install and configure the fault-tolerance product, provide redundant hardware and software, and configure the application to run redundant copies of its objects. As you'll discover, fault tolerance results from the way the application uses the redundant hardware to run redundant copies of its software.

This article is abridged from the forthcoming book *Quick CORBA 3*, by Jon Siegel, copyright 2001 by the Object Management Group and published by John Wiley & Sons.

FT Basics

FT CORBA supports applications that require a high level of reliability. Under FT CORBA, applications must not have a single point of failure.

FT systems provide this degree of assurance through

- *Redundancy (replication)*
- *Fault detection and notification*
- *Logging and recovery*

The FT CORBA infrastructure allows individual object instances to be replicated on multiple processors at different locations on the network – even in different cities or on different continents – in a very flexible way. Even when calls are cascaded, with replicated objects calling other replicated objects, propagation is controlled so that no copy executes the resulting call more than once.

Faults are detected by a number of mechanisms, including both *push* (heartbeat) and *pull* monitoring. To avoid scalability problems, a Fault Detector on each host monitors the individual objects on it, and a global (replicated, of course) Fault Detector monitors the individual host Fault Detectors.

Replication and Object Groups

The replicas of an object are created and managed as an object group. To render this replication transparent to the client, the copies are created and managed by a Replication Manager and addressed by a single Interoperable Object Group Reference (IOGR). I won't present details of the IOGR here – they're hidden from the application and serve only to allow the ORB and FT infrastructure to work together to deliver FT support.

Fault Tolerance Domains make it practical to create and manage large FT systems. An FT Domain may contain a

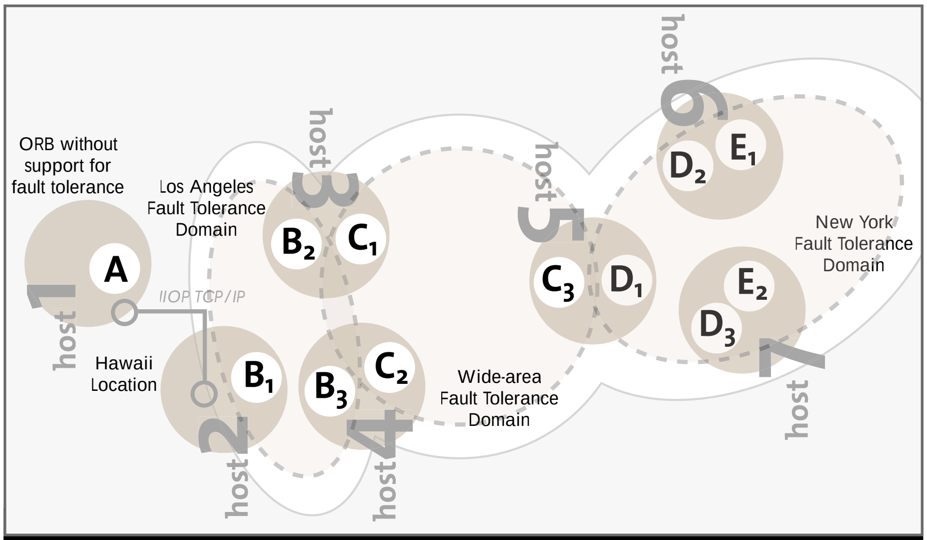

ORB without support for fault tolerance
Los Angeles Fault Tolerance Domain
New York Fault Tolerance Domain
Hawaii Location
Wide-area Fault Tolerance Domain
IIOP TCP/IP
host 1
host 2
host 3
host 4
host 5
host 6
host 7
A
B_1 B_2 B_3
C_1 C_2 C_3
D_1 D_2 D_3
E_1 E_2

FIGURE 1 Structure of fault-tolerance domains

number of hosts and many object groups, although a single host may support multiple domains. There is a single (replicated) Replication Manager for each domain.

In Figure 1 lightly shaded ovals denote the extent of FT domains, darker ones denote hosts, and circles denote object instance replicas. Capital letters within an object's circle denote its object group; the set of circles with the same letter is thus the set of replicas of an individual instance. Hosts may participate in one (Host 2) or more (e.g., Hosts 3 and 4) domains, but all objects in a group must be in the same domain because they are all created and managed by that domain's Replication Manager.

Every object group has a set of FT properties – ReplicationStyle, MembershipStyle, ConsistencyStyle, FaultMonitoringStyle, and six others – that may be set either domain-wide, per type, or per group. Unlike, for example, POA properties, some of these may be modified after the group is created.

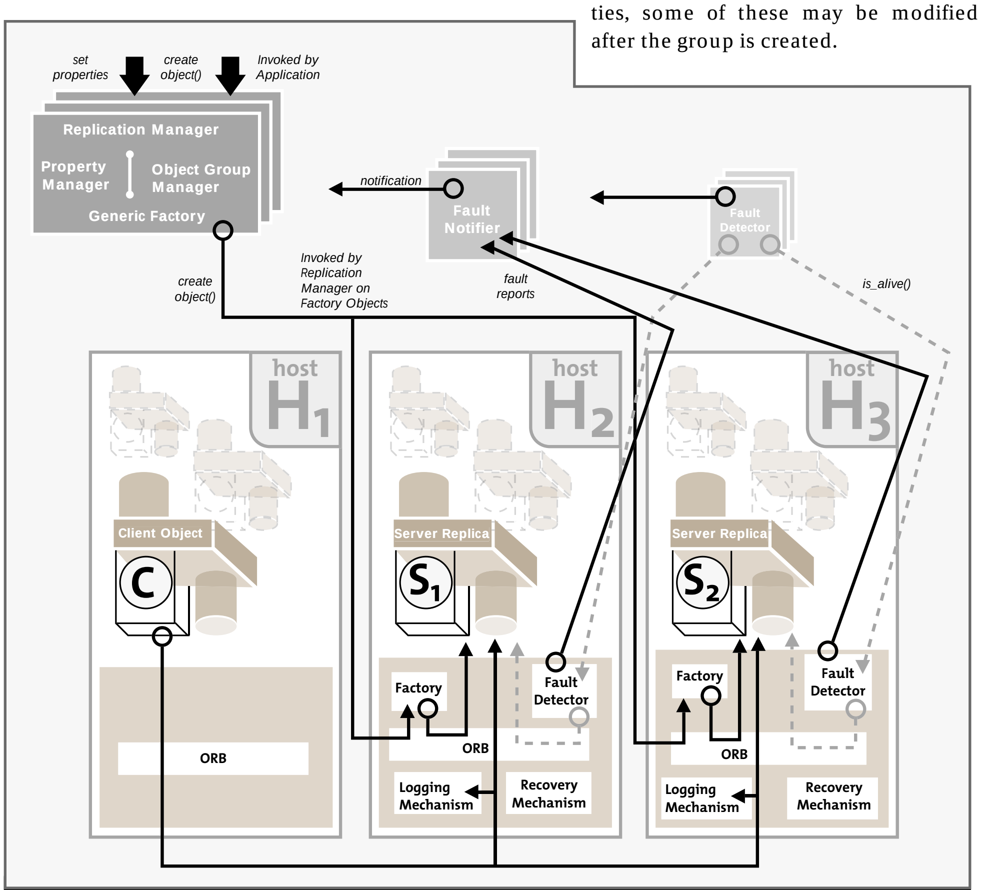


FIGURE 2 Possible architectural configuration of fault-tolerant system

Active vs Passive Replication Styles

There are two major styles of replication: active and passive. Passive then subdivides into two styles of its own, but let's look at the difference between active and passive first.

- Every active replica executes every invocation independently, in the same order as every other replica. The replication mechanism inhibits duplicate invocations when one replicated object invokes another. Active replication is faster and cheaper when the cost of computation is less than the cost of saving/restoring state, and necessary when recovery has to be instantaneous.
- Only one passive replica of a replicated object, the primary member, executes invoked methods, saving its state periodically. When a fault occurs, a backup member is promoted to primary, and its state is restored from a log and the replaying of request messages.

Warm passive replication lessens the recovery delay somewhat by loading state into backup objects periodically during execution; *cold* passive replication does no loading until the primary member fails.

Another mode, *Stateless*, applies to objects that have no dynamic state.

ACTIVE_WITH_VOTING, the only replication style we haven't covered yet, is interesting for two reasons:

1. It isn't fully supported by the current specification and represents a possible future extension.
2. This mode (and the fact that it's not supported) points out a limitation: the current specification protects only against crash faults, that is, when an object issues no response.

The specification doesn't protect against what it terms *commission* faults, where an object generates incorrect results, or against what it terms *byzantine* faults, where an object or host generates incorrect results intentionally. The ACTIVE_WITH_VOTING replication style, used with sophisticated algorithms, can protect against both types of faults, albeit at considerable cost in network traffic.

Strong Replica Consistency is the principle of FT CORBA that guarantees that objects have identical state. Supported by the specification, it guarantees that for passive replication all members of an object group have the same state following each state transfer, and that for active replication all members have the same state at the end of each operation.

For this to work, applications must

be *deterministic,* or must be *sanitized* to give the appearance of being deterministic: for a given starting state a given invocation with a given set of parameters must produce the same output and the same internal state. (*Sanitize* means to *clean up* a set of replicas so that all give the identical answer to an invocation, removing all sources of nondeterministic behavior. Although this may be simple for a routine that queries a database row or performs a calculation, it's not so easy to sanitize an object against a query whose result varies with even slight differences in invocation delivery timing, or depends on a hardware serial number or some other value that differs from one machine to another.)

The specification requires that the FT infrastructure deliver the same set of invocations in the same order to every member of an object group. The deterministic behavior of the object instances, combined with the consistent behavior of the FT infrastructure, guarantees strong replica consistency.

The Replication Manager

Figure 2 shows a sample configuration for a fault-tolerant system. I'll use this configuration to discuss the way each element works.

The Replication Manager, itself replicated as the figure shows by the nested boxes, inherits three interfaces defined separately in the specification: PropertyManager, ObjectGroupManager, and GenericFactory. The PropertyManager interface lets you define fault-tolerance properties for your object groups, possibly using a GUI. Of the properties defined, two are especially relevant here: MembershipStyle and ConsistencyStyle. MembershipStyle defines whether the infrastructure or application controls membership in an object group, and ConsistencyStyle defines whether the infrastructure or the application controls consistency of state for members of an object group.

Fault Detector and Fault Notifier

Figure 3, copied from the specification, shows diagrammatically the sequence of events that ensues when a fault is detected.

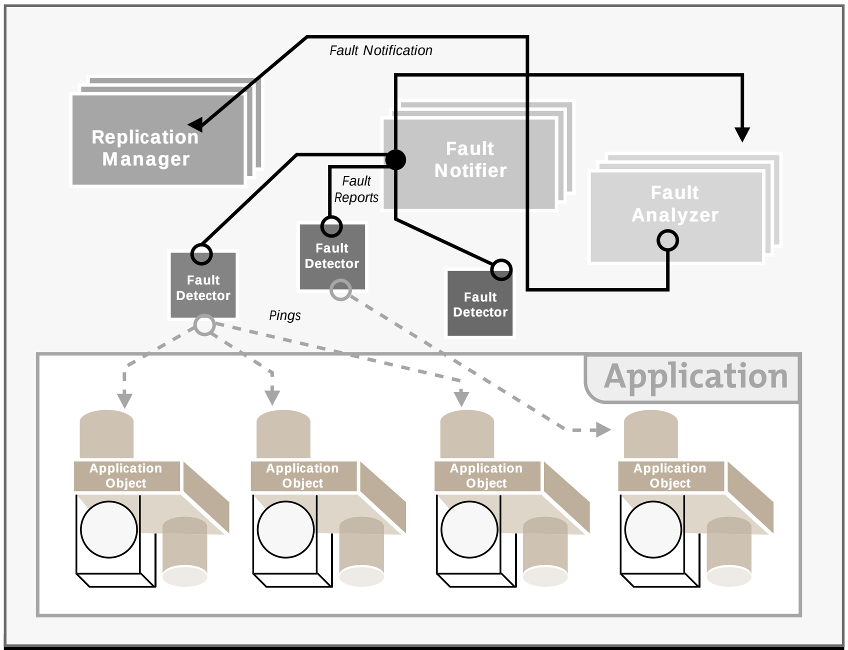


FIGURE 3 Interactions among Fault Detectors, Fault Notifier, Fault Analyzer, and Replication Manager

As mentioned before, at least one Fault Detector on each host periodically pings each object replica on that host to see if it's functioning. Applications can configure additional Fault Detectors as needed. When a Fault Detector detects a fault, it pushes a report to the Fault Notifier using the FaultNotifier interface, a subset of the Notification Service interface.

Logging and Recovery

Everything covered so far was an introduction to this part. Here's where the service fixes up your application when an object crashes.

If you're running in one of the passive replication styles (either warm or cold), only the primary member of each of your object groups actually executes an invocation and sends back replies. When the Fault Detector suspects that the primary member has failed, it signals the Replication Manager to restart the primary member or to promote a backup member to primary.

At this point the primary member needs to have its state restored. If you've chosen the application-controlled consistency style, the application must fetch and restore the state of the new primary member. This is easy to describe: it's application-dependent, and you have to code it yourself. If you've chosen infrastructure-controlled consistency style, things are more automatic: using the Checkpointable interface with its operations set_state( ) and get_state( ), the system keeps state current in a log and uses the most recent values to restore the member during recovery.

If you've chosen active replication, none of this applies at failure/recovery time; the system just has one less duplicate response than usual and goes on with its processing without missing a beat. However, the logging and recovery mechanisms still use get_state( ) and set_state( ) to maintain a log that is used to synchronize new instances that might be added to an object group at runtime.

Fault Analyzer

The Fault Analyzer registers with the Fault Notifier to receive fault reports. It then correlates fault reports and generates condensed fault reports using an application-specific algorithm.

Even though reporting is orthogonal to recovery and your system is (hopefully) chugging along without missing a beat even when redundant copies or hosts fail, you'll still want to know everything that's gone wrong during execution. The Fault Analyzer is the system component that takes care of this. Although a simple fault analyzer may report every fault it receives, it's much better if it performs correlations and condenses several thousand nearly simultaneous instance fault reports into a single host fault report!

Limitations and Conformance

Some limitations to FT CORBA are given in the following abbreviated list. By analyzing them you'll gain additional insight into the workings of an FT system.

- ***Legacy client ORBs:*** An unreplicated client hosted by a CORBA 2.3 or earlier ORB can invoke methods on a replicated server, but won't participate fully in FT (for reasons that space constraints preclude discussing).
- ***Common infrastructure:*** For this first release of the specification, all hosts within a FT domain must use the same FT product and ORBs from the

"Strong Replica Consistency is the principle of **FT CORBA that guarantees** that objects have identical state"

same vendor to ensure FT behavior. Between domains, full FT behavior is guaranteed only if all employ the same FT product and the same ORB, although some improvement can be expected even when different products are used.

- ***Network partitioning faults:*** A network partitioning fault divides the system into two parts, each able to operate and communicate within itself but unable to communicate with the other. The inherent nature of the problem doesn't allow assured detection and recovery from these, which are therefore not covered by FT CORBA.
- ***Commission and byzantine faults:*** As mentioned earlier, commission faults occur when an object or host generates incorrect results, and byzantine faults occur when this happens intentionally or maliciously. These fault types are neither detected nor corrected by FT CORBA, although the ACTIVE_WITH_VOTING replication style provides a mechanism that can be used to build a solution. The solution, however, will be expensive in terms of resource.
- ***Correlated faults:*** These faults cause the same error to occur simultaneously in every replica or host, and typically result from failures in design or coding. They can happen anywhere: application, operating system, storage, network, hardware, or any other replicated part of the system. FT CORBA provides no protection against such faults.

FT CORBA defines two conformance points. An implementation must support at least one, and may support both:

1. Passive replication FT CORBA products support only the COLD_PASSIVE, WARM_PASSIVE, and STATELESS replication styles.
2. Active replication FT CORBA products support only the ACTIVE and STATELESS replication styles.

That's all the fault tolerance we have space for here. I know it isn't enough to get you programming in FT mode, but I think it's more than enough to trigger an investigation into whether FT CORBA can provide your enterprise with the assurance you need to survive in today's world of business dependence on computing.

Acknowledgment

I'd like to thank Dr. Louise Moser and Dr. Michael Melliar-Smith of Eternal Systems, Inc. (www.eternal-systems.com), for technical review of this article and for their contributions to the Fault Tolerant CORBA specification on which this article is based.

siegel@omg.org

AUTHOR BIO

Jon Siegel, director of technology transfer at Object Management Group, also writes articles and presents tutorials and seminars about CORBA. His book, CORBA 3 Fundamentals and Programming, was published last year by John Wiley & Sons.

CREATE A FOUNDATION

build an application

server infrastructure

written by matt creason

As most of you reading this article know, the application server market is growing and every company, large or small, can visualize the benefits an application server infrastructure could bring to their organization. But why then, even with the vast amount of benefits available, have companies not adopted an application server as the foundation for their organization?

The reason is they're making it more complex than it really is. Though it's by no means a small chore, it is a large puzzle, but one that can be pieced together by maintaining focus. The focus should be on the three major inherent benefits an application server infrastructure will bring to a company.

- ***Scalability:*** The ability to meet system demands
- ***Robustness:*** Confidence in being efficient, stable, and proven
- ***Security:*** The competence to reduce loss of information and quash customer fear

To remain focused on these benefits that will create the foundation for your application server infrastructure, you must continue to answer three simple, yet complex, questions:

- Who is the target audience?
- What is the agreed or implied level of service expected?
- What level of security must you maintain in your environment?

These questions can usually be answered definitively, but only for a certain point in time. Therefore, to ensure that you maintain a strong foundation, your company must revisit these questions to keep up with a moving target. Every time you pose these questions to your organization there will most likely be negotiation and compromise, but remember that a solid foundation is a balanced one. Keeping this in mind, let's discuss three application server infrastructure models that your company could use: development, small-business, and enterprise.

Development Model

The development model is an all-in-one solution, literally. The Web server, the JSP engine, and the application server all run on the same box. Though this model works, it tends to construct more roadblocks than remove them. The only positive argument is a financial one. It's cheap and your company only has to invest in one piece of hardware and in a limited amount of CPU- or MHz-priced software licenses to house and run the different pieces of the infrastructure mentioned above.

Because everything runs on one box, points of failure begin to compound as the development model is built. First, there's no contingency for software or hardware redundancy. Without Web server redundancy, should the Web server go down, there's no point of entry into the application(s), given the premise that these are browser-based application(s).

This same theory applies if the application server software process should fail, but with even more severe consequences. Instead of just affecting browser clients, all the application(s) become unusable, therefore disabling all supported client models for the affected application(s). Not only is this model risky on the surface, the risk is compounded by the high memory and I/O intensive processes these servers place on one piece of hardware. These discrepancies directly impact an infrastructure's ability to be scalable and reliable.

In addition, this model lacks any implementation of a firewall. As a result, there's no allowance for a demilitarized zone, which makes this infrastructure highly vulnerable to hackers. Security must come from either a custom authentication application or utilization of the application server vendor's security protocols. Though vulnerable and maybe not the most scalable or robust application server infrastructure, this configuration is a solid and cost-effective development environment. It could even be run as a successful departmental infrastructure designed to serve a small target audience that expects a reasonable level of service, and requires minimal security.

Small-Business Model

A small-business model divides tasks among multiple pieces of hardware to gain high throughput and redundancy capabilities. In this configuration, the Web server and the application server have been separated, and the JSP engine can reside with either one based upon the chosen vendor's software ability to integrate a JSP engine into their respective product. Though to get the highest reliability, scalability, and security, the JSPs should be run within your application server in order to take full advantage of the inherent pooling and security mechanisms provided. This is the most common model used when companies choose to build an application server infrastructure. The popularity of this model is a direct result of a balance of maintaining scalability, robustness, and security with the financial impact of purchasing and supporting additional hardware.

The first and most obvious benefit of separating the Web and application server processes across at least two pieces of hardware is that you now have the ability to implement a firewall between the two. This provides your organization and customers with the necessary security to protect both entities in the e-business space. A second benefit of this model is that you alleviate the stress placed on the "all-in-one" or development model of those high I/O and memory-intensive processes by splitting them across separate pieces of hardware. As a result, your infrastructure eliminates the Web and application server processes from "stepping" on each other, thereby greatly reducing the possibility of the aforementioned catastrophic failure. Finally, this configuration allows your organization to easily scale your infrastructure to meet system demands. Ease of scalability directly stems from the inherent increase in manageability and the ability to monitor with finite precision the appropriate Web or application server process.

With this enhanced sense of infrastructure awareness your organization can remove the finger of blame being thrown around between the two processes, which usually correlates directly to organizational structure, and focus on solving the problem rather than discussing whose problem it is. By concentrating on the issues, your organization will be able to react to a projected or actual loss of level of service by bringing online only those additional server resources that are required.

The drawbacks to the small-business model are financial and security. There's an increased financial burden in terms of the additional capital expenditures of hardware and the manpower to maintain these systems. In terms of security, even though this model incorporates a firewall, it does not build a demilitarized zone required by some organizations. But by maintaining the focus on our target audience of a small business, most do not require such a strict security construct. Despite these facts, the small-business model is a proven one and has been shown to balance fiscal responsibility with an ability to be highly scalable, robust, and secure.

Enterprise Model

Now we come to the granddaddy of the infrastructure models, the enterprise model. The enterprise model is the one that every company strives to achieve, but can't always afford. This model is often large and complex, and therefore places a high fiscal demand on a company, which must be taken into consideration. Though theoretically there is only a single hardware addition of a firewall between the application server and the database, the complexity of being able to manage this behemoth grows exponentially because it usually involves multiple clusters and redundancy.

"The enterprise model is the one that every company strives to achieve, but can't always afford"

This growth affects your bottom line by dramatically increasing your need for manpower to manage and monitor this infrastructure. In addition, there's the additional hardware cost of adding a firewall. With this said, and though these costs can be significant, most organizations that are in the enterprise space already know that in order to play, you must pay.

The benefits this model brings to an organization are simple. It incorporates and provides the same level of high reliability and scalability that the small business model would, but wraps a stringent security framework around the application server infrastructure. By adding this security "wrapper," the enterprise model creates the highly sought-after demilitarized zone. This is a transition zone from one world to another, or from the Internet to inside your organization, which allows only certain protocols on specific listeners to enter into your environment. An enterprise model provides a company with the most scalable, robust, and secure infrastructure available that can serve any space, provided your organization can justify the financial costs.

As presented, these models hinge not only on the technical aspects of scalability, robustness, and security, but have a high dependency on financial requirements. This is emphasized in no short order due to the "dot-bombs" that have fallen by the wayside lately. Yes, some of the companies were doomed from the start, but most did not plan and build a scalable infrastructure that was fiscally feasible. Because even for "techies" this is a reality, and I thought it important that you know not only how to build your company's application server infrastructure, but that you also have some understanding of the financial investments required to implement this foundation.

In closing, keep your eye on the ball while you build and maintain an application server infrastructure by continually revisiting and focusing on:

- Who is the target audience?
- What is the agreed or implied level of service expected?
- What level of security must you maintain in your environment?

AUTHOR BIO
Matt Creason is a system consultant with the Internet Applications Division of Sybase, Inc. He is a current member of the IEEE and the IEEE Computer Society.

matt.creason@sybase.com

Jlink: Cybelink's Framework for Creating Reusable Enterprise Components Using J2EE

Create scalable enterprise-wide Web architecture

Part 2 of 3

WRITTEN BY
MANI MALARVANNAN

In Part 1 of this article (*JDJ*, Vol. 6, issue 2) we discussed the problems associated with J2EE's Servlet/JSP container. In Part 2 we'll discuss Cybelink's Jlink architecture and how it solves those problems.

Jlink Architecture

Jlink is built on various architectural and software design patterns. In this section we'll discuss the software patterns, specifically, the Model-View-Controller (MVC) patterns.

MVC Architectural Pattern

MVC is an architectural pattern that's been widely used in architecting distributed component-based applications. It facilitates the division of the application into logical components that can be designed and developed independently. This division increases the reusability of components by reducing the couplings between them.

In the MVC pattern the Model components represent the business logic (e.g., JavaBeans and EJBs), the View components represent the UI that displays the processing results of the business logic, and the Controller components manage the coordination between the Model and View components. Figure 1 shows the relationship between the various components of the MVC pattern. The Model components maintain the state of the business object and when the state of the Model object changes, they notify the View components, which update the UI with new data. It's also possible for the View components to request state changes directly from the Model components. The View components enable the Controller components to change the display according to the user's request. The Controller components map the user's actions from the View to the Model components.

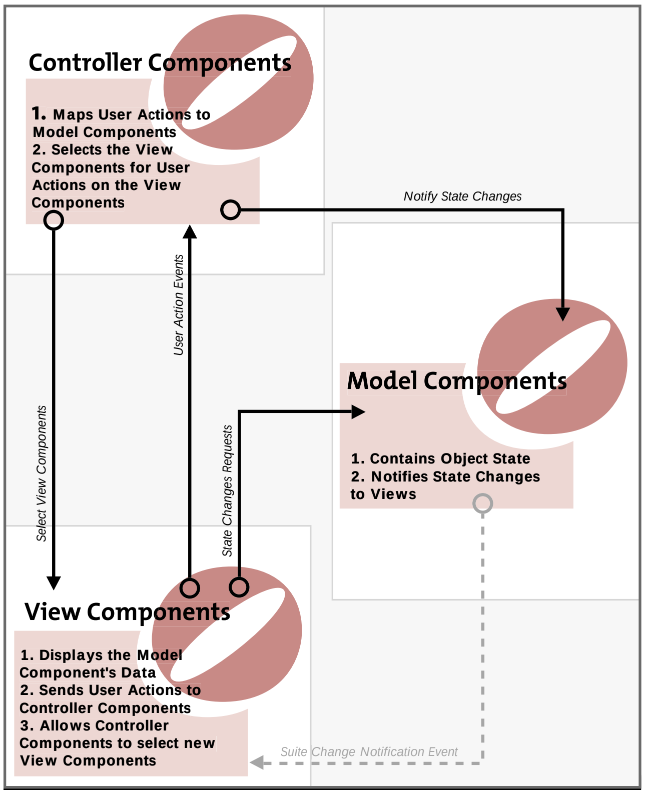

Controller Components
1. Maps User Actions to Model Components
2. Selects the View Components for User Actions on the View Components
Notify State Changes
User Action Events
Select View Components
State Changes Requests
Model Components
1. Contains Object State
2. Notifies State Changes to Views
View Components
1. Displays the Model Component's Data
2. Sends User Actions to Controller Components
3. Allows Controller Components to select new View Components
Suite Change Notification Event

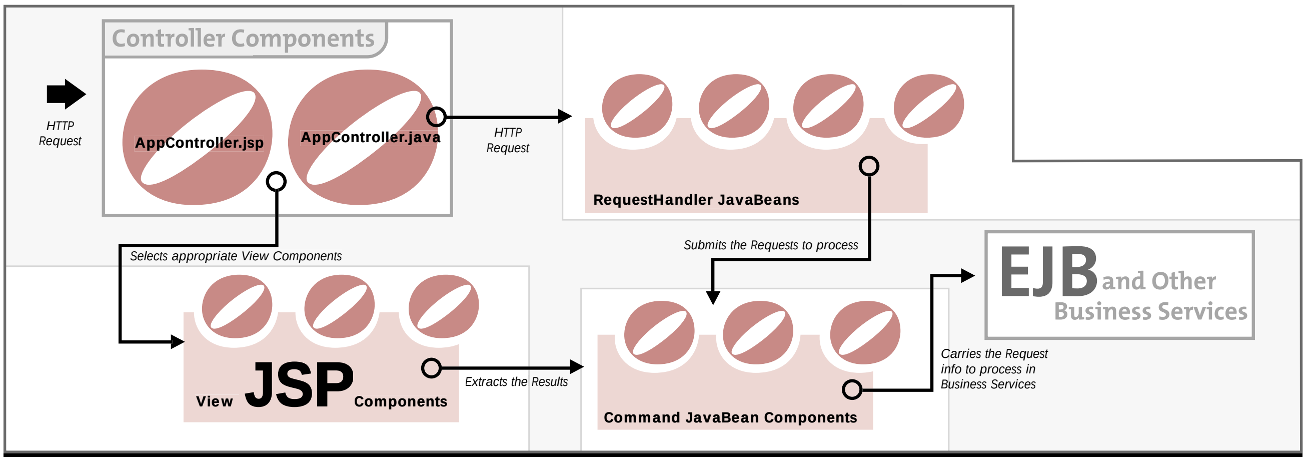
FIGURE 1 MVC architecture

Mapping MVC with Servlets and JSP

In Part 1 we discussed the problems associated with Sun's Servlet/JSP container; in the previous section we discussed the MVC pattern. Now that we understand the Servlet/JSP Container model and the MVC pattern, we can start applying the MVC to the Servlet/JSP Container model. Traditionally, the MVC pattern was used to build client/server applications; in this section we'll see how we can use it to develop Web-based applications using Servlets/JSP. In our Jlink we'll use the JSP and JavaBean as the Controller components, the JSP pages as the View components, and JavaBeans as Model components.

JSP and JavaBeans as Controller Components

The AppController.jsp (see Listing 1) and AppController bean (see Listing 2) are used as Controller components that are responsible for accepting the HTTP requests and passing them to the appropriate RequestHandler object. (Listings 1–7 can be found on the *JDJ* Web site, www.JavaDevelopersJournal.com.)The RequestHandler objects are designed using JavaBeans. These beans strip the browser requests and send them to the appropriate Command JavaBean components. The Command beans are modeled after Command Design patterns. Before handling the request to the RequestHandler objects, the AppController.jsp creates a ClientContext (see Listing 3) object for each request and passes it to the RequestHandler object. The



Controller Components
HTTP Request
AppController.jsp
AppController.java
HTTP Request
RequestHandler JavaBeans
Selects appropriate View Components
Submits the Requests to process
EJB and Other Business Services
View JSP Components
Extracts the Results
Carries the Request info to process in Business Services
Command JavaBean Components

FIGURE 2 Jlink architecture

ClientContext contains javax.servlet .HttpRequest, javax.servlet.HttpResponse, and javax.servlet.http.HttpSession objects. Thus the ClientContext encapsulates the Client state by maintaining the necessary objects in one place, eliminating the spaghetti code discussed in Part 1. Other objects should get parameters, such as Request and Query, from the ClientContext object.

JSP as View Components

In our Jlink, JSP pages represent the View components. These pages interact directly with the JavaBean Model components to get the results to display on the Browser.

Model Components

The Jlink has been architected in such a way that we can use JavaBeans, EJB, C++, or any other component as the Model. The Controller and View components are not affected when we change our Model components from JavaBeans to EJBs. The Command JavaBean objects shield the Controller and View components from the Model components.

Command JavaBean

The Command JavaBean shields the Controller and View components from the Model components. They get the requests from the handler objects and pass them to the Model components located either in the same machine as the Command JavaBean or in a different one. Once the Model components execute the requests, they collect the results, which are then retrieved by the View components. The RequestHandler objects or the JSPs will worry about issues such as transactions, the Model object's location, and more. The Command JavaBean shields these issues from the Controller and View components, thereby increasing the reusability and coupling of the components.

AUTHOR BIO

Mani Malarvannan is CEO and cofounder of Cybelink (www.cybelink.com), a Minnesota-based company that specializes in e-business consulting and training. Mani has several years of OO analysis and design experience and has been working in Internet and Java technologies since their inception. He holds a BS and an MS in computer science.

In our Jlink all HTTP requests go to the AppController.jsp, which creates the AppController JavaBean for the first request (see Figure 2). The javax.servlet.ServletContext object is set in the AppController JavaBean. For a Web application the AppController.jsp creates one AppController JavaBean that will be shared by all the Model and View components in the Jlink. The AppController.jsp acts as a bridge between the browser and the AppController JavaBean. For each HTTP request, it creates a ClientContext object and prompts the RequestHandler to call the handleRequest method by passing the ClientContext object.

Figure 2 shows the architecture of the Jlink. If you compare Figure 1 with Figure 2 you can see how Jlink is modeled after the MVC pattern.

Now that we've discussed the high-level workings of Jlink, we'll look at individual classes and interfaces in more detail.

Jlink Classes and Interfaces

This Jlink is built on JSP, Java interfaces, and abstract and concrete classes. In this section we'll discuss the individual classes and interfaces. Figure 3 shows the UML class diagram of the Jlink.

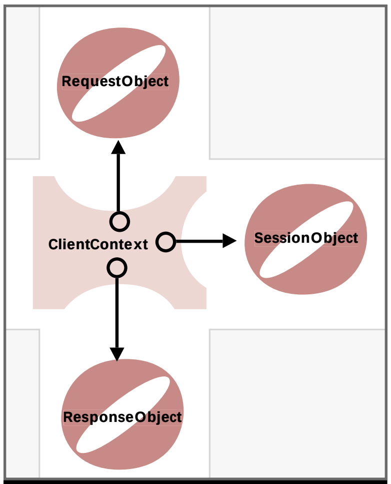


FIGURE 3 ClientContext object structure

AppController.jsp

As mentioned earlier, the AppController.jsp (see Listing 1) creates the AppController JavaBean the first time the HTTP request comes from the browser. After creating the bean, it sets the ServletContext object to the AppController bean. For each request, the AppController creates a new ClientContext object by passing the HttpServletRequest, HttpServletResponse, and HttpSession objects. Using the ClientContext object it gets the RequestHandler object from the AppController JavaBean and calls the handleRequest method by passing the ClientContext object. The RequestHandler computes the result and selects a JSP page that's appropriate for displaying it. The AppController.jsp retrieves the JSP page and passes it to the browser. Note that neither the Controller components (AppController.jsp and AppController bean) nor the View components (JSP pages) know how the results are computed in the RequestHandler object.

Controller Interface

This interface (see Listing 4) provides basic methods that all AppControllers need to implement in order to qualify as an AppController.

AppController JavaBean

The AppController JavaBean implements the Controller interface (see Listing 2). One AppController object is created per Web application the first time an HTTP request comes to that application. This AppController manages application-wide objects. The AppController JSP sets the ServletContext object to the AppController bean, which is used to store application-wide objects and can be accessed by all the JavaBeans within the application.

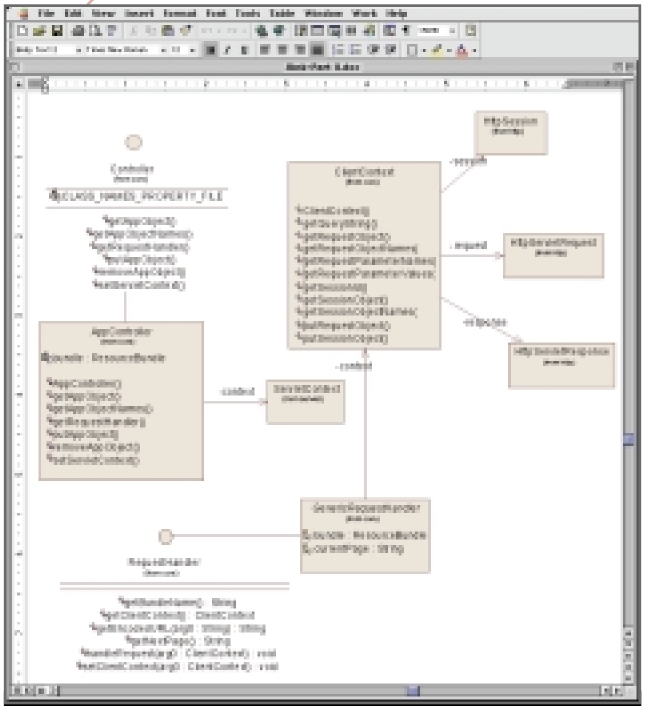


FIGURE 4 UML class diagram for Jlink

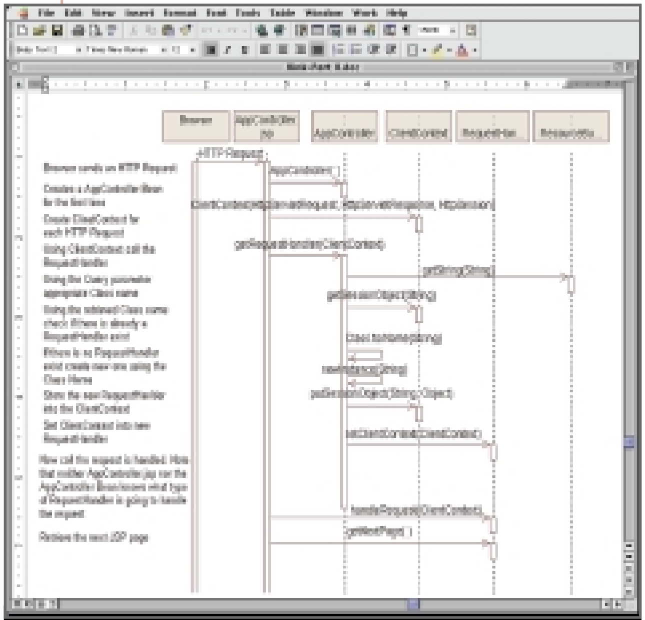


FIGURE 5 UML sequence diagram for Jlink

The java.util.ResourceBundle object is used to read the resource names and paths from a property text file. The methods getAppObject and putAppObject retrieve and add application-wide objects to the ServletContext. This provides any JSP or JavaBean within the Web application with the flexibility to add or retrieve the objects. The important method in this class is geRequestHandler, which we'll discuss in detail. Listing 5 shows the implementation of the getRequestHandler.

getRequestHandler accepts a ClientContext and checks if a RequestHandler already exists for the session in which the ClientContext object was created. If the RequestHandler doesn't exist, a new RequestHandler is created. It uses the forName static method from the class to get the Class object and the newInstance method on the Class object to create the appropriate RequestHandler object. The forName method takes the name of the class as a parameter. The class name is obtained from the property file using the key obtained from the query parameter, which is passed from the browser. This design solves our problem of mapping browser requests to proper Servlet/JSPs and avoids the big if-then-else statement in the AppController bean. Thus we can add more RequestHandler objects as the Web site grows.

Request Handler Interface

This interface (see Listing 6) provides basic methods that all RequestHandler classes must implement to qualify as a RequestHandler.

GenericRequestHandler

This is an abstract class (see Listing 3) that implements the RequestHandler interface and provides a generic implementation for handling HTTP requests. The Constructor is called from the AppController JSP for each HTTP request. The Constructor creates a java.util.ResourceBundle object. The name for the ResourceBundle object comes from the method getName, an abstract method in the GenericRequestHandler object. The subclasses of GenericRequestHandler must provide the actual name for the ResourceBundle. This approach provides the subclasses of the GenericRequestHandler with the flexibility to have their own ResourceBundle names. The methods getClientContext and setClientContext retrieve and set the ClientContext object. The method getNextPage returns the JSP page (View component). The handleRequest method does nothing in this abstract class. The subclasses of the GenericRequestHandler must provide the actual implementation for this method. This approach allows Jlink users to have their own project-specific implementation for the RequestHandler objects and still use the infrastructure provided by Jlink.

ClientContext

The ClientContext is a crucial object in our Jlink (see Listing 7). The Constructor takes HttpServletRequest, HttpServletResponse, and HttpSession objects. The methods getSessionObject and putSessionObject retrieve the objects using the HttpSession object. As explained earlier, the HttpSession object and all other objects added to the HttpSession become potential candidates for garbage collection if the browser that created the HttpSession object is timed out. The methods getRequestObject and setRequestObject retrieve the HttpServletRequest. Including the HttpServletRequest object, all other objects that are added to it become potential candidates for garbage collection when a new HTTP request comes from the browser. The methods getRequestParameterNames() and getRequestParameterValues() retrieve the names and values of the parameters set by the browser. The method getQueryString() returns the query parameter that's set by the browser. As explained earlier, the Query string is used to find the appropriate Class Name to create the RequestHandler object.

Thus the ClientContext object (see Figure 4) provides a wrapper for HttpServletRequest, HttpServletResponse, and HttpSession objects and the outside world can access them through the ClientContext object. This design avoids spaghetti code in the JSP and JavaBean source code described in Part 1.

Figure 4 shows the structure of the ClientContext object. For each HTTP request a new ClientContext object is created.

Jlink Class Diagram

In any Java-based framework Java interfaces play a critical role in creating the frameworks that provide tools for the architects, designers, and developers to create reusable components. Jlink is no different; Figure 5 shows the class diagram for the Jlink interfaces and classes described earlier. The AppController implements the Controller interface and we can create several application-specific AppController classes by implementing it. The Controller interface provides coordination among all the AppController objects within the framework. This mechanism allows us to add more application-specific AppController classes, thereby making the framework adaptable to various types of Web applications.

The GenericRequestHandler implements the RequestHandler interface and provides an abstract behavior for all the handler classes in a particular Web application. Depending on the Web application's needs, we can create several application-specific GenericRequestHandler classes by implementing the RequestHandler interface.

How Jlink Works

In the previous section we discussed the static aspects of Jlink. In this section we'll discuss the dynamic aspects. An HTTP request comes from a browser to the AppController.jsp (see Figure 5). On the first request for a Web application the AppController.jsp creates an AppController bean and sets the ServletContext object. This ServletContext object lives as long as the Web application is alive in the Servlet/JSP container. Next, the AppController JSP creates a ClientContext for every HTTP request by passing the HttpRequest, HttpResponse, and HttpSession objects. Then it uses the ClientContext to get the RequestHandler from the AppController bean.

At this level the AppController.jsp deals only with the interface RequestHandler, but the AppController bean creates an object of the class it inherits from the GenericRequestHandler class. When the AppController.jsp asks for a RequestHandler, the AppController bean gets the Session ID from the ClientContext and checks if the RequestHandler already exists by passing the Session ID to the getSessionObject method on the ClientContext. If the RequestHandler is already created, getSessionObject retrieves it from the HttpSession object stored in the ClientContext. Even though a new ClientContext is created for each new HTTP request in the AppController.jsp, the HttpSession is the same for a single browser connection. Only the HttpServletRequest and HttpServletResponse objects are created every time an HTTP request comes to the AppController.jsp. Finally, once a RequestHandler object is created, it remains as long as the corresponding browser connection is valid.

“...creating the frameworks that provide tools for the **architects, designers, and developers** **to create reusable components**”

Conclusion

In Part 2 we discussed the Jlink's architecture and how it solves the problems of creating scalable enterprise-wide Web architecture. Jlink was built using industry best practices such as OO methodology and software design patterns. Using Jlink, organizations can create reusable component-based architecture that will evolve with the technology and the organization.

References

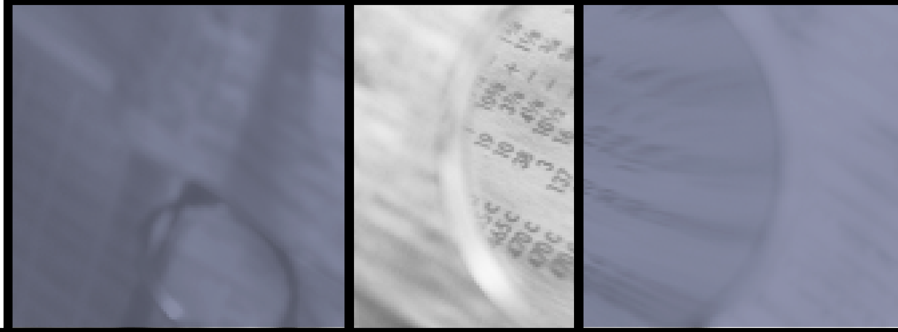
- *Sun J2EE Blueprint:* http://java.sun.com/j2ee/blueprints
- Gamma, E., Helm, R., Johnson, R., and Vlissides, J. (1995). *Design Patterns – Elements of Reusable Object-Oriented Software*. Addison-Wesley.
- Buschmann, F., et al. (1996). *Pattern-Oriented Software Architecture*. John Wiley & Sons.
- Fields, D.K., and Kolb, M.A. (2000). *Web Development with JavaServer Pages*. Manning Publications.
- *Servlet/JSP API:* http://java.sun.com/products/servlet/2.2/javadoc/index.html

mani@cybelink.com

Building J2EE Applications for Performance and Scalability

techniques, optimizations, and examples

WRITTEN BY MISHA DAVIDSON



The highly structured nature of applications built using J2EE technologies lends itself well to design patterns for performance optimization. This article examines a number of such patterns and suggests optimal ways of using them to improve latency, throughput, and overall scalability of J2EE applications.

Application Performance Defined

The standard performance metrics for computer systems are latency and throughput. In the context of J2EE applications, latency is the time that elapses between the moment a client request is received by the server and the moment a response is sent back. Throughput is the number of client requests that are handled per second.

Both latency and throughput are important. To achieve high performance, you want to minimize the latency of your application while increasing throughput. In practical terms, subsecond latency is usually sufficient for Web systems. Acceptable throughput rates vary depending on the application; however, throughput of hundreds of e-commerce (e.g., shopping cart) transactions per second is considered quite good.

Both latency and throughput define the performance of a system under a particular load. However, J2EE systems operate under rapidly changing loads, at times coping with an influx of client requests an order of magnitude greater than normal. The system's ability to cope with such an increased load is called *scalability*.

Typically, system performance degrades with increased client load: latency increases and throughput falls. When a poorly designed system is confronted with an unexpectedly high load (see Figure 1), its latency grows nonlinearly and becomes unacceptably high while throughput falls sharply. An ideal system maintains a constant latency regardless of the load level, while its throughput scales linearly with the load. While ideal systems don't exist, a well-designed system should scale well, maintaining low latency and high throughput under a wide range of loads. This article examines techniques for building such applications.

Structure of J2EE Applications

J2EE applications are comprised of three tiers of components – client, server, and enterprise information system (EIS) resources. The client can be a static page or an applet running in a browser (HTML or WML), or a J2EE client application running inside a J2EE client container. On the server tier, servlets and JavaServer Pages (JSPs) that run inside a Web container generate the application Web UI. EJBs that run in an EJB container encapsulate application logic and data access. A typical J2EE application uses a relational database as its EIS-tier. Figure 2 shows the relationship between J2EE components.

This article presents techniques that affect the performance of Java and database-backed systems in general, optimizations that apply to individual J2EE component types, and a number of J2EE application patterns that improve performance.

J2EE Application Performance Basics

At their core, J2EE applications are Java programs that utilize a database. As such, J2EE applications are subject to all performance principles that apply to traditional Java and database programming. Good coding techniques and efficient use of the database are a prerequisite for writing high-performance J2EE applications.

Database Usage Basics

Databases can be huge – thousands of tables and millions of rows of data. Improper use of a database in a J2EE application drastically affects performance. Keep the following points in mind when architecting the database portion of the application:

- ***Avoid* n*-way joins:*** Every join has a multiplicative effect on the amount of work the database has to do to execute a SQL statement. Multiway joins degrade the performance of an applica-

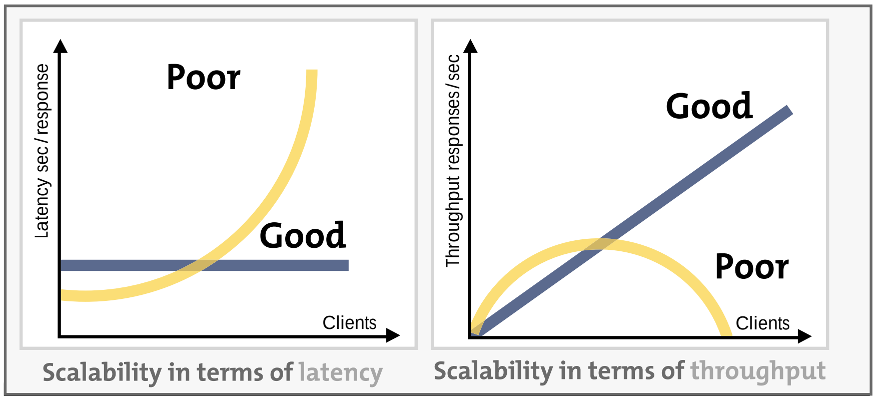


FIGURE 1 Scalability in terms of latency and throughput

tion once the data set becomes sufficiently large. You may not notice the performance hit on a development system using a sample database, but it'll manifest itself in production. Make sure any multiway joins your application is using are really needed.

- ***Avoid bringing back thousands of rows of data:*** If you don't, your server will spend an inordinate amount of resources and time storing them in memory. The user whose actions caused the query will get a sluggish response, and everyone else's ability to do work will be impaired. If you absolutely have to bring back such a large data set, consider bringing back just the primary keys first and fetching individual rows on-demand. This approach doesn't reduce the total amount of work, but, it spreads it out, greatly reducing the immediate response time.
- ***Cache data when reuse is likely:*** Accessing a database is by far the most expensive operation an application server can perform. Thus, the number of such accesses should be minimized. In particular, if an application frequently refers to a certain subset of values from the database, those values should be cached.

Java Coding Guidelines

Coding techniques affect the performance of J2EE applications, though not as much as database usage. Here are a few basic points to keep in mind when writing J2EE applications:

- ***Avoid unnecessary object creation:*** Current implementations of JVMs are much more efficient in managing object creation. Still, it's one of the more expensive operations that JVMs perform.
- ***Minimize the use of synchronization:*** Synchronization blocks reduce scalability of applications by forcing serialization of all concurrent threads of execution. Setting up synchronization also requires a nontrivial amount of resources from the JVM. Therefore, serialized blocks should be kept small and used only when absolutely needed. The same rule applies to using standard Java classes. The JDK usually provides at least two implementations for each data structure, one with synchronization and one without (e.g., Vector is synchronized and ListArray is not). When using JDK classes, pick synchronized implementations only when synchronization is actually needed; use unsynchronized versions in all other cases.

J2EE Component Performance

Now that we've defined the basics of writing performant J2EE applications, let's look at specific performance techniques for writing servlets, JSPs, and EJBs.

Servlet Performance Hints

The first thing to note about writing servlets for performance is that the use of a SingleThreadModel interface should be avoided. SingleThreadModel is a marker interface that tells a Web container that the code inside the service() method of the servlet class is not thread-safe. Typically, the container will create and possibly cache multiple instances of the servlet class to handle the concurrent invocations of the service() method. Since multiple instances of the servlet class can be used at the same time, marking a servlet with a SingleThreadModel doesn't guarantee thread-safety of an application. Multiple instances of the servlet class may still be accessing the same underlying classes and data at the same time. Managing multiple instances of the servlet does, however, incur significant overhead for the Web container. Therefore, instead of writing bad servlet code and covering it up with a SingleThreadModel, it's preferable to write thread-safe code to begin with.

In some cases, it may be possible to speed up the writing of the output from a servlet by using an OutputStream instead of a PrintWriter (both can be obtained from the ServletResponse). In general, a PrintWriter should be used to ensure that all character set conversions are done correctly. However, if you know that your servlet returns only binary or ASCII data, you can use an OutputStream instead. An OutputStream writes data directly to the wire, thus forgoing the character set conversion overhead.

Using the getRemoteHost()method on a ServletRequest can dramatically increase the latency of your application. This call performs a remote DNS look-up on the IP address of the client. This operation typically takes seconds to complete and should be avoided at all costs.

JSP Performance Hints

Even though JSPs provide an elaborate functionality built on top of servlets, the performance of JSPs is roughly comparable to that of servlets. JSPs compile to servlets; compilation introduces a trivial amount of initialization code. Compilation happens only once and JSPs can be precompiled, eliminating any runtime overhead. JSPs are slower than servlets only when returning binary data, since JSPs always use a PrintWriter, whereas servlets can take advantage of a faster OutputStream.

JSP custom tag libraries can encapsulate arbitrarily complex Java functionality in a form that can be easily used by Web designers who aren't that knowledgeable about Java. Thus, JSP custom tag libraries are an effective tool for dividing the work between programmers and Web designers. However, every time a custom tag is used on a page, the Web container has to create a new TagHandler object or fetch it from the tag cache. In either case, excessive use of custom tags may create unnecessary processing overhead.

BodyTags are a special kind of custom tag. They have access to, and can do transformations on, the contents of their bodies. The use of BodyTags causes the contents of the page between the start and the

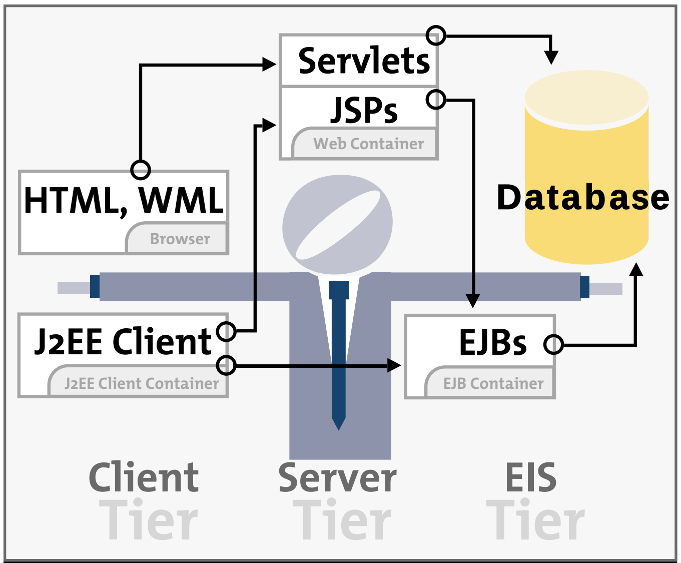


FIGURE 2 J2EE application components

end portions of the tag to be copied in memory. They can be nested and iterate over their bodies. Using multiple levels of BodyTags combined with iteration will likely slow down the processing of the page significantly.

EJB Performance Hints

Conversational session beans that write to the database should be designed to prevent conflicts when doing the update. The simplest way of achieving transactionality in this case is to lock the whole table, or just the rows that are being updated. However, this approach incurs high overhead and greatly reduces the scalability of EJBs written this way. Instead, optimistic concurrency control may be used, so there would be no need to lock the data that's being updated. Instead, all the updates to the database receive an additional WHERE clause containing the old values of all columns in the row being updated. In case of contention, the first writer succeeds while everyone else fails, because the WHERE clause doesn't match after the first update. The result is the same as when strict locking is used. However, optimistic locking involves smaller overhead and allows multiple updates to proceed in parallel, thus increasing the scalability of the system. Note that, as with most database operations, the benefits of optimistic locking degrade when there are many collisions and the application spends a lot of time dealing with failed updates.

Entity EJBs may form a "graph" of related beans. Not all parts of such an object graph may be needed by an application at all times. To reduce the overhead of loading such object graphs, a technique of "lazy-loading" dependent EJBs can be used.

For example, suppose we have an insurance policy bean that refers to holder and vehicle beans, and we're interested only in the identification of the vehicle in that policy, not the identity of the policyholder. In this case, loading the holder bean at the same time the policy bean is loaded would be wasteful. Instead, we load the dependent beans only when they're actually needed (see Listing 1).

Some application servers use optimistic locking and lazy loading internally in their container-managed persistence (CMP) implementations.

J2EE Application Performance

Good coding practices and efficient J2EE components are only a part of writing high-performance J2EE applications. Architecture has a critical effect on application performance. Here are some examples.

Read-Only Web Applications

Many real-world J2EE applications such as on-line reservation systems and catalogs provide read-only access to a large set of data. These applications handle a large number of queries from different users; some of the query results are large, but all are short-lived. Using entity EJBs is overkill for this type of application, as they individually read each row of data and the server expects to keep that data in memory for a while. Instead, this type of read-only Web application lends itself well to the following optimization. Embed SQL statements that access the data in JavaBeans. Use these beans directly from JSPs to fetch the data.

Going from JSPs to the database without involving the EJB layer significantly speeds up the application. It also reduces the overall complexity. This shortcut is acceptable because data access is read-only and there's little processing logic. This approach isn't suitable for applications that require complex processing or write to the database.

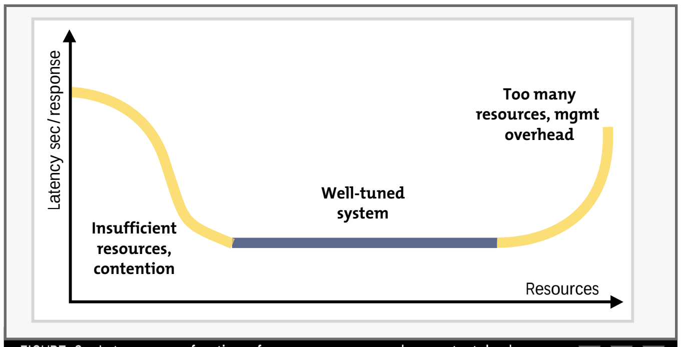


FIGURE 3 Latency as a function of server resources under constant load

Infrequently Changing Shared Data

On the other end of the application spectrum are applications that share a small subset of infrequently changing data. A typical example of such application functionality is a list of top stories or most popular items on a shopping site. Stateless session EJBs can be used to efficiently cache and manage such data.

"Good coding practices and efficient J2EE components are only a part of writing high performance J2EE applications. Architecture has a critical effect on application performance"

There are three types of objects involved in this scenario: stateless session EJBs that access and store the data; clients, for example, JSPs that use this data; and a special remote Cursor class that's owned by the client and used to keep track of which rows were read by that client. To cache the data, the session EJB in its ejbCreate() method performs a SQL query, reads in the result, and stores the individual rows as elements in a ListArray. To read a row of data, clients execute a business method, readNextRow(), on that session EJB. However, different clients need to access different rows stored by that session bean. To do that, clients pass in a special cursor object that holds the number of the last row read by that client. The session bean uses that number to find the next row to give to the client. It then increments the row counter inside the cursor before returning the row (see Listing 2). A more efficient implementation results if the client manages the row number inside the cursor.

To prevent the data from going stale, readNextRow() must implement a refresh mechanism. In the above example, the data is refreshed after it's been accessed a certain number of times. Alternatively, data can be refreshed after it's been cached for a certain duration.

Minimizing the Number of Network Round-Trips for J2EE Clients

J2EE client applications may call EJBs directly. Each call involves a JNDI look-up and a network round-trip, both of which can be expensive. When a J2EE client contains all the processing logic, it has to perform multiple JNDI look-ups and network round-trips to access the data presented by entity EJBs on the server. This overhead can be reduced to a single look-up and round-trip if the processing logic is encapsulated in a single session bean. That bean, in turn, encapsulates all the entity beans that would otherwise be accessed directly by the J2EE client. In this case, a J2EE client acts as a rendering engine for the results contained in a single object returned by the session bean.

Deployment Strategies

Application design and coding techniques determine the performance of the resulting systems. The way in which an application is deployed has an effect as well.

- ***Minimize interprocess communication:*** If possible, run the Web server and the application server in the same process. Separating the two can significantly increase the response time of your application.
- ***Use clustering to increase scalability:*** As long as your application doesn't require the servers in your cluster to communicate with each other, adding more servers keeps latency low while increasing throughput.
- ***Provide at least the minimal set of resources needed to run an application.*** This includes setting a sufficient size for the JVM's

heap, providing a reasonable number of database connections, and setting a reasonable size for thread, servlet, and bean pools. Failure to provide enough resources results in contention and severely degrades application performance. However, unreasonably high resource settings will degrade performance as well. For example, if the heap size is greater than the physical memory available, the JVM will thrash instead of doing useful work. Similarly, if the size of the database connection pool is greater than the number of connections allowed, the application will spend a lot of time dealing with what it perceives as database failures (see Figure 3).

Summary

J2EE applications, like any application, should be architected for performance and scalability. J2EE application designers can take advantage of the following common principles.

- Entity beans are too heavy to use with read-only data, use JDBC instead.
- Stateless session EJBs can be used to efficiently represent infrequently changing shared data.
- Minimize the network overhead for J2EE clients by locating the processing logic in a session bean on the server.

In building individual J2EE components, the following rules apply. For servlets:

- Avoid using a SingleThreadModel interface
- Use print OutputStream instead of PrintWriter to output binary/ASCII data
- Don't use a getRemoteHost() method on a ServletRequest

For JSPs:

- Be careful when using nested BodyTags

For EJBs:

- Use optimistic locking to improve performance when doing database updates
- Use lazy-loading to efficiently handle dependent beans

Because J2EE applications are Java programs that use databases, common programming principles are applicable to J2EE apps as well. Specifically, for JDBC access:

- Avoid *n*-way joins
- Avoid bringing back thousands of rows of data
- Cache data when reuse is likely

For Java programming:

- Avoid unnecessary object creation
- Minimize the use of synchronization

Finally, when deploying J2EE applications, provide them with sufficient resources and use clustering to achieve higher scalability.

References

1. *J2EE Blueprints:* http://java.sun.com/j2ee/blueprints/
2. Larman, C., and Guthrie, R. (2000). *Java 2 Performance and Idiom Guide*. Prentice Hall.
3. *Professional Java Server Programming J2EE Edition*. (2000). Wrox Press.
4. Roman, E. (1999). *Mastering Enterprise JavaBeans and the Java 2 Platform, Enterprise Edition*. Wiley.
5. Halter, S., and Halter, S.M. (2001). *Enterprise Java Performance*. Prentice Hall.

AUTHOR BIO

Misha Davidson is a product manager for SilverStream Software, Inc.

misha@silverstream.com

Listing 1: Lazy-loading dependent entity EJBs

```
public class PolicyBean implements javax.ejb.EntityBean {
  public Vehicle vehicle; // data we re interested in
  public Holder holder;

  public void ejbLoad() {} // Load nothing by default

  public String getVehicleId() {
    loadVehicle(); // insure that dependent beans are loaded
    return vehicle.getVIN();
  }

  private void loadVehicle() { // do this in-line for better per-
formance
    if (vehicle != null) {
      // get initCtxt here
      home = (VehicleHome) initCtxt.lookup("VehicleHome");
      vehicle = home.findByPolicy (policyID);
    }
  }
}
```

Listing 2: Using a stateless session EJB to cache shared data

```
public class SharedData implements SessionBean {
  private ListArray m_data; // cached data
  private int m_accessCnt; // access counter
  public void ejbCreate() {
    reloadData(); // get data for the first time
  }
  private void reloadData() {
    // do sql query, store result in m_data
  }
  public String[] readNextRow (Cursor curs) {
    if (m_accessCnt++ > 1000) { // refresh?
      m_accessCnt = 0;
      reloadData(); // re-get the data
    }
    int i = curs.incRow(); // update row
    return (String[]) m_data.elementAt(i);
  }
}
```

Download the Code!
www.JavaDevelopersJournal.com

Fitting the Pieces into the Enterprise Java Jigsaw

Providing an access control infrastructure for intranet applications

Part 1 of 3

WRITTEN BY
TONY LOTON

The choices can be overwhelming for a development team embarking on an Enterprise Java project. You've read the books, attended the classes, and now know the individual Java technologies pretty well, but how do you choose between them? Should your project be based on servlets, applets, EJBs, any two, or all three?

In this series I try to show how each technology can be used as part of an enterprise application to fit the pieces into the Enterprise Java jigsaw. For a practical perspective, I'll present an example that starts with a single Java servlet and finishes with a small working application that's composed of applets, servlets, EJB, and JDBC, and has a basic access control mechanism. My intention is not to provide a comprehensive tutorial on any individual technology, but to give just enough information to demonstrate some of the possible combinations.

The emphasis will be placed firmly on the three major architectures – applet, servlet, and EJB – that provide three alternative styles for enterprise application development. I'll mention some of the other J2EE technologies – such as XML and JSP – in passing, but not in detail.

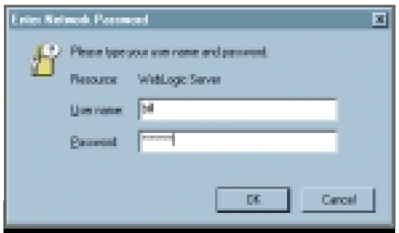
The code listings highlight the important techniques, and some routine statements were removed for clarity. Although you can't necessarily run these code snippets as presented, rest assured they're all based on applications that are known to work.

HTTP Authentication and a Login Servlet

In this article I start with a single servlet and use this as the basis for a simple but effective access control system based on HTTP authorization and servlet session tracking. By the end of this article, the security framework for my application will be in place and I'll have covered some simple JDBC along the way.

The access control solution that I offer is intended for use with in-house intranet applications with security requirements that are limited to authentication (asking a user to log in) and authorization (propagating the user's identity so application components can allow or deny their functionality to the user). If you're developing Internet applications that handle sensitive data, such as credit card numbers, you'll also need to think about encryption.

Getting a user to log in by supplying a username and password is quite straightforward with HTTP authentication. Every application server that I've


Enter Network Password
Please type your user name and password.
Resource: WebLogic Server
User name: bill
Password
OK
Cancel
FIGURE 1 HTTP authentication dialog box

used allows you to flag the URL path to any resource as being protected. When a user tries to access the HTML page or servlet that the URL represents, the server provokes the user's browser into displaying an authentication dialog box (see Figure 1). The target page will be displayed or the servlet executed only if the user supplies a valid username as the password. This is completely automatic; all you need to do is provide the server with a list of valid usernames and passwords.

Some servers allow you to use the same method but vary the way the authentication is actually done, maybe via an HTML form or using certificates. Bear that in mind when you build a real system; however, for my example I'll stick with the basic HTTP authentication.

For authentication I'll use a servlet as the protected resource and call it LoginServlet. When the user tries to run the LoginServlet via its URL, he or she will be prompted to authenticate. If authentication succeeds, the servlet will be executed. Listing 1 shows an extract of the code for the LoginServlet, which possesses no code for authentication since this is handled automatically by the application server and Web browser.

First I get the remote user name from the HttpRequest. If the user hasn't authenticated because a valid username and password were not supplied, or because I forgot to protect the servlet URL in the first place, the getRemoteUser() method returns null. This means that authorization (propagating the user identity to allow or deny certain

"Possibly, it represents the simplest approach to **providing an access control infrastructure** for intranet applications"

functions) won't work without prior authentication, which is exactly what we want.

Next I take the current HttpSession, which provides a context for me to pass the user identity and any other information from this servlet to other servlets that comprise the application. Any Web server supporting the servlet API will provide a session-tracking mechanism, probably via cookies, that supports an HttpSession context for each remote user.

Finally I write out an error message or a link to a user application as the HTML response, depending on the success or failure of the authentication. In the latter case I also write the user name into the HttpSession for future servlets to pick up.

An Application Servlet

The "Click here to run an application" link shown in the last statement of Listing 1 points to another servlet that I've created called AppServlet, which acts as a real application the user can run. An extract of the code is provided in Listing 2.

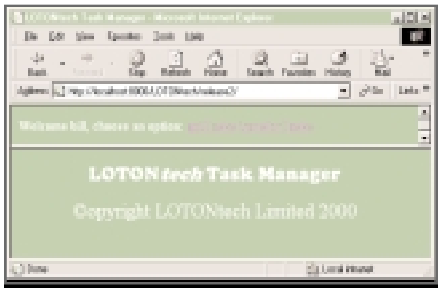
First I take the current HttpSession for the user, then extract the user's identity, which was stored by the LoginServlet, from the session. If all I wanted to know was the username, I could have called req.getRemoteUser() instead, but the point of using the HttpSession is to allow more complex information to be propagated – maybe a list of the user's access rights or presentation preferences, either of which may have been determined initially by the LoginServlet. Based on the user credentials, in this case his or her name, the application servlet does one of three things as indicated by the comments.

Because the user entry in the HttpSession will be null unless the LoginServlet has been visited, there's no need to make AppServlet or any other application servlet a protected resource, thus reducing the amount of configuration that's needed each time a new application is added. Another benefit is that the user's first port of call must be the LoginServlet, so you have a single point at which you can implement some application-independent initialization – maybe opening a database connection for the lifetime of the user's session, or preventing people from logging in at all during system downtime.

So far it appears as if this idea is limited to servlet-based applications, doesn't it? Well, you can also apply it to applications based on Java applets as long as you invoke each applet via a corresponding servlet rather than from a static HTML page. Each applet should have a servlet that creates the HTML containing the <applet> tag only if the servlet decides that the user is authenticated and authorized for that applet. There will be more about applets in my second article.

JDBC and a Menu of HTML Links

Being authorized to run only one application is not particularly useful, so I'll now beef up the LoginServlet a bit to display a list of options available to each user upon logging in. These authorized options are stored in a database table


Welcome bill, choose an option:

LOTONtech Task Manager

Copyright LOTONtech Limited 2000

FIGURE 2 Authorized options for "bill"

USERNAME	USEROPTION
bill	GetTasks
bill	transferTasks
ben	GetTasks

TABLE 1 Underlying database table

called *useroptions*, and I use JDBC to extract the relevant rows for the current user. Each option will be displayed as an HTML link (see Listing 3).

You might have noticed that I'm using JNDI to look up the name of a data source to use for my JDBC connection. This provides a level of indirection that allows the specification of the actual JDBC connection string to be deferred until deployment. Depending on your requirements and the application server you're using, you might want to connect using the more traditional approach, for example (for Oracle):

```
 DriverManager.registerDriver(new
oracle.jdbc.driver.OracleDriver());

Connection con =
DriverManager.getConnection(

"jdbc:oracle:thin:@(DESCRIPTION=(ADDRESS
_LIST=(ADDRESS=(COMMUNITY=tcp)

(PROTOCOL=TCP)(Host=123.456.789.10)(
Port=1521)))(CONNECT_DATA=(SID=MYDB
)))",
  "scott", "tiger");
```

AUTHOR BIO

Tony Loton works through his company – LOTONtec Limited (www.lotontech.com) – as an independent consultant, course instructor, and technical author. He's spent the past 10 years in IT, the last five devoted almost exclusively to Java, UML, and related technologies. He holds a degree in computer science and management.

On the presentation side, at this point I'll add an initial HTML page that has a frame for the login page (and options menu) and a separate frame for the initial splash screen and subsequent content.

Figure 2 demonstrates that when "bill" logs in, he sees two authorized options – getTasks and transferTasks.

The underlying database table contains the data shown in Table 1, so when another user, "ben," logs in he sees only the getTasks option. Take my word for it.

Conclusion

I've implemented these ideas with the J2EE Reference Implementation and WebLogic Server, and something similar in the distant past with Oracle Application Server and Java Web Server. Possibly, it represents the simplest approach to providing an access control infrastructure for intranet applications, taking advantage of the mechanisms already provided by the application server and adding just one additional component, the LoginServlet.

Every one of the application servers I've mentioned supports HTTP authentication and session tracking via cookies. One difference you might need to be aware of, however, is that while session tracking across servlets seems to work irrespective of the URL context in WebLogic, the J2EE demands that all servlets sharing the same HttpSession must be deployed in the same Web archive (WAR) file, and hence share the same URL context.

Finally, with the basic security mechanism in place and a promise that it can be used with applets as well as servlets, tune in next time to find out how to incorporate one or more applets into the architecture and set up a communication channel between the servlets and the applets that comprise the enhanced application.

tony@lotontech.com

Listing 1

```
public class LoginServlet extends HttpServlet
{
 public void doGet(HttpServletRequest req
 , HttpServletResponse res) throws IOException
 {
   // -- find out who the remote user is --
   String user=req.getRemoteUser();

   // -- get the current http session --
   HttpSession session=req.getSession(true);

   res.setContentType("text/html");
   PrintWriter out = res.getWriter();

   if (user==null)
   {
     // -- authentication failed, show error --
   }
   else
   {
     // -- save user name for future servlets --
     session.setAttribute("user", user);

     // -- respond with HTML including --
     // -- this link to an application --
     out.println("<a href='AppPage'>Click here to run an applica-
tion.</a>");
   }
 }
}
```

Listing 2

```
public class AppServlet extends HttpServlet
{
 public void doGet(HttpServletRequest req
 , HttpServletResponse res) throws IOException
 {
   // -- get the current http session –
   HttpSession session=req.getSession(true);

   res.setContentType("text/html");
   PrintWriter out = res.getWriter();

   // -- get the current user --
   String user=(String)
    session.getAttribute("user");

   if (user==null)
   {
     // -- not authenticated, show message --
   }
   else if (user.equals("bill"))

   {
     // -- we like this user, so run the app --
   }
   else
   {
     // -- authenticated, not allowed this app —-
   }
 }
}
```

Listing 3

```
public void doGet(HttpServletRequest req
, HttpServletResponse res) throws IOException
{
 // -- find out who the remote user is --
 String user=req.getRemoteUser();

 // -- get the current http session --
 HttpSession session=req.getSession(true);

 res.setContentType("text/html");
 PrintWriter out = res.getWriter();

 if (user==null)
 {
   // -- print the unauthorized message here --
 }
 else
 {
   // -- save the user name for future servlets --
   session.setAttribute("user", user);

   out.println("<html>");
   out.println("<head><title>LoginServlet</title></head>");
   out.println("<body>");

   Vector options=new Vector();

   try
   {
     // -- get DB connection from a datasource --
     InitialContext ic = new InitialContext();

     DataSource ds = (DataSource)
      ic.lookup("java:comp/env/jdbc/LOTONtech");

     Connection con = ds.getConnection();

     // -- select user s menu options --
     Statement st=con.createStatement();

     ResultSet results=st.executeQuery
      ("SELECT username, useroption
      FROM useroptions WHERE username='"+user+"'");

     while (results.next())
     {
       String useroption=results.getString(2);
       options.addElement(useroption);
     }
   }
   catch (Exception e)
   {
     out.println("ERROR: Connecting to database!");
   }

   //-- write the options out as html --
   out.println("Welcome "+user+", choose an option:");

   for (int opNum=0; opNum<options.size(); opNum++)
   {
     String thisOption=(String)
      options.elementAt(opNum);

     out.println("<a href='"+thisOption
      +"' target='main'><font size=3><b>"
      +thisOption+"</b></font></a>  ");
   }

   out.println("</body>");
   out.println("</html>");
 }
```

Download the Code!
www.JavaDevelopersJournal.com

APPLICATION SERVER METAMORPHOSIS

THE ESSENTIAL CAPABILITIES

written by Sudhakar Ramakrishnan & Christopher G. Chelliah

The Java 2 Platform, Enterprise Edition (J2EE), defines the standard for developing and deploying multitier enterprise applications. At the core of J2EE architecture are application servers – containers for your J2EE components.

This article explains the architecture of an application server that embraces standards such as J2EE, and focuses on ways to leverage the application server platform for caching, load balancing, scalability, and clustering services. It also provides an understanding of how to build J2EE applications, by looking at both design and runtime scenarios. It concludes with some best practices for designing and implementing enterprise applications.

The Metamorphosis: From Web Servers to Application Servers

During the early '90s, the advent of the Internet gave birth to myriad programming models like CGI and FastCGI. This enabled ubiquitous access to computing resources. A shift toward server-centric IT development took shape to leverage these resources efficiently, and resulted in Web servers that could run various kinds of applications.This gave birth to the application server.

Later, application servers provided proprietary interfaces such as NSAPI and ISAPI that enabled developers to access a server's internal services. However, these techniques were not standard across application servers, limiting flexibility and deployment choices. Developers facing these challenges embraced standards such as J2EE that enabled them to focus on business logic, and not the underlying plumbing and proprietary interfaces. This created a drive for the next-generation application server that supported standards such as J2EE for building e-business applications.

Application Server Architectures

Few companies have been able to create an application server that addresses the different functionalities necessary in a multitiered architecture in a cohesive way. In general, application servers should provide:

- Programming standards and models such as J2EE
- Access to all tiers
- Application partitioning
- Messaging, transactional, and distributed services
- Manageability

Additionally, application servers are required to provide support for multiple platforms, including Linux, Solaris, and Windows NT.

This article uses Oracle9*i* Application Server (Oracle9*i*AS) (see Figure 1) to describe the basic building blocks of an application server including:

- Mechanism to handle HTTP requests
- Scalable execution environment for Java programs
- Containers for J2EE components
- Performance optimizations, such as caching, load balancing, fault tolerance, and clustering

Mere support of J2EE alone is not sufficient. Advanced features such as wireless integration, portals, business intelligence, Web services, collaboration, and process integration can make it easier for developers to create e-business applications. We'll describe these concepts using Oracle9*i*AS in an upcoming issue of ***JDJ***.

Mechanism to Handle HTTP Requests

Oracle HTTP Server is powered by Apache, a popular Web server. The HTTP server not only serves files, but also provides functionality to execute programs to generate dynamic content. To facilitate e-business, HTTP engines support the HTTPS protocol, thereby providing a safe and secure transport. In choosing an HTTP server, one must pay attention to its ability to handle high loads.

Oracle9*i*AS builds on Apache's extensible architecture by adding modules to extend the core functionality. The modules abstract the server's operations to improve performance. Oracle HTTP Server comes with additional modules including mod_ssl, mod_perl, mod_ose (OracleServlet engine), and mod_plsql. Oracle9*i*AS has SSL support for both 40-bit and 128-bit encryption. Oracle provides customer support for all Apache modules shipped with Oracle9*i*AS.

Scalable Execution Environment for Java Programs

A JVM is an abstract machine that runs compiled Java programs. Every application server vendor that supports J2EE has to provide a concrete implementation that conforms to the JVM specification.

A platform is scalable if it provides consistent performance regardless of the number of concurrent users. As the load on standard VMs increases, congestion of requests can take place. As a result, more VMs have to be started to handle the increasing number of user requests. A few of these VMs can become overloaded and server response can start to fail. Oracle Enterprise Java Engine (EJE) addresses this problem with a session-oriented VM that reduces the response time by making sure each session has its own virtual Java VM, Java global variables, threads, and garbage collector. The underlying runtime provides the needed scalability and threading, enabling the VM to efficiently perform memory management utilizing a low session footprint.

Oracle9*i*AS includes a scalable EJE to provide an execution environment for Java components. It supports use of the standard JDK JVM for lightweight components and the Oracle EJE for heavy-duty applications. The Oracle EJE is the same engine featured in the database and has been proven for concurrency (see "Enterprise Java: Oracle8*i*JVM," ***JDJ***, Vol. 5, issue 10).

Containers for Various J2EE Components

The J2EE runtime environment is composed of containers and components. A container acts as a vessel to hold the component, and provides a J2SE runtime environment and implementation of J2EE APIs for services like transactions, messaging, and distributed protocols. In addition, application servers come with a number of tools to create, assemble, deploy, and manage applications to support J2EE roles.

Containers play a critical role in hosting J2EE components by shielding users from the complexities of the middle tier. Containers take away the necessity of writing plumbing code and allow developers to focus on business logic, delegating the container to handle transaction management, scalability, persistence, and security services.

Oracle9*i*AS has a complete implementation of J2EE containers for enterprise APIs, including Servlet, EJB, JSP, XML, JMS, and JNDI. The Oracle EJE programming environment supports SQL data access through JDBC and SQLJ, distributed applications through EJB, and dynamic Web site development using JavaServer Pages and servlets.

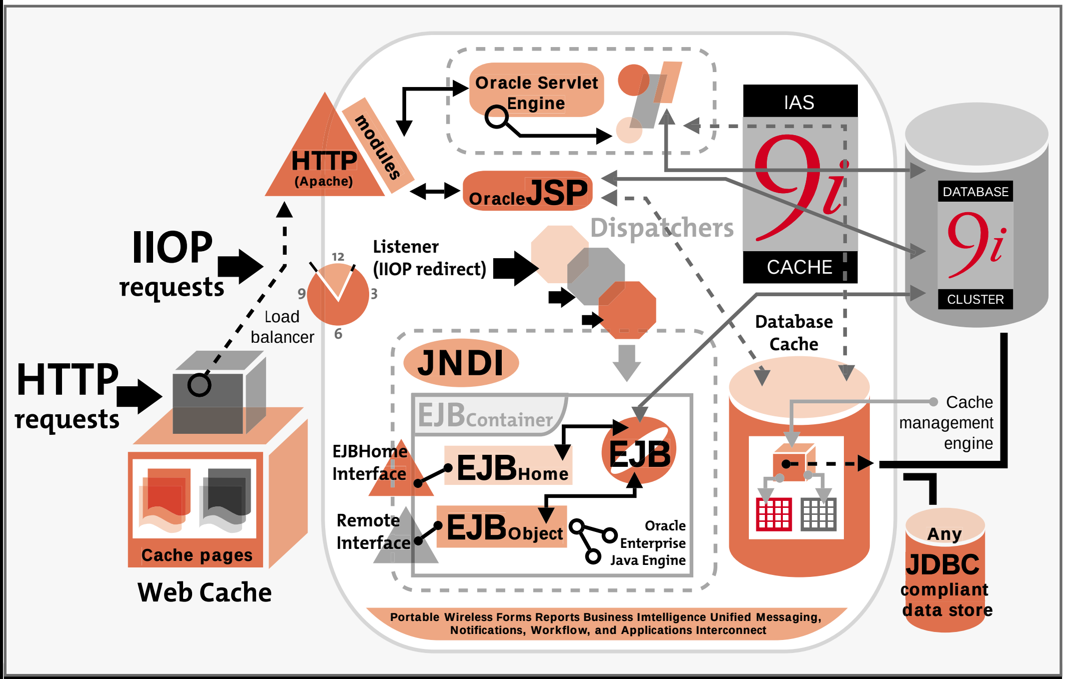


FIGURE 1 Oracle9*i* Application Server architecture

Performance Optimizations

Performance of an application server hinges on caching, load balancing, fault tolerance, and clustering.

Caching

One of the notable properties of the Internet is that, at times, an HTML page can get popular quickly, creating a hot spot on your Web site. These hot spots are dynamic in nature and keep moving around with time. A key performance measure for the Web is the speed with which content is served to users. As traffic on the Web increases, users are faced with increasing delays and failures in data delivery. Caching is one of the key strategies to improve performance.

Solutions should include a Web cache to improve application server performance and a database cache to improve the data access performance. Both technologies are necessary and work to complement each other. Oracle9*i*AS has both Web cache and database cache technologies.

Improving Application Server Performance Using Web Cache

An important issue in many caching systems is how to cache frequently changing data and provide fast response to users. The Oracle9*i*AS Web Cache solution includes:

- ***Dynamic content caching and content invalidation***: A popular myth is that dynamic data is uncachable. However, Oracle9*i*AS Web Cache allows cachability rules for both static and dynamic content created by J2EE components. Cache invalidation can be time-based, or triggered by sending XML messages in an HTTP post request. For example, if a column in a table is updated, a database trigger can send an invalidation request to the cache.
- ***Personalization of cached data:*** To provide personalization features, Oracle9*i*AS Web Cache embeds session IDs in every URL to make them unique for each user. Using these IDs and a substitution mechanism, the Web cache can insert personalized information into cached objects. For example, a Welcome page with the greeting "Hello Sudhakar" is generated by a JSP. When the page is served through Web cache, it parses out the parameter and caches a "template." When the next site is accessed, the parameter is substituted into the template and served by Web cache. The second request won't even hit the application server.
- ***Server acceleration:*** In this technique, cache servers are placed in front of a cluster of Web servers to intercept all HTTP requests. The cache stores objects returned by an origin server, handles thousands of concurrent users, and frees processing resources on the origin server.
- ***Quality of service:*** To prevent an overload on origin servers, caching systems assure performance by setting a limit on the number of concurrent connections that servers can handle. Oracle9*i*AS Web Cache helps distribute the load to origin Web servers using a heuristic algorithm and request queues.

Improving Database Tier Performance Using Database Cache

Many applications rely on EJB components for heavy-duty transaction operations. Acquiring JDBC connections is usually the slowest part of the process. To circumvent this problem, most EJB servers perform connection pooling and caching through JNDI and data sources. JDBC drivers that support connection pooling are useful, but this alone is not enough for better performance.

Minimizing the number of round-trips between the middle tier and data tier can influence the response time. The relative percentage of read-only data in the middle-tier application can have an effect on the number of round-trips your application has to perform. Database caches are effective in multitier environments where databases are located on separate nodes. These caches reduce the load on the database tier by processing the most common read-only requests to data sets on the application server tier.

A database cache should be an operational aspect of the Web site and transparent to the application developer. Oracle9*i*AS Database Cache keeps track of queries and can intelligently route to the database cache or to the origin database without any application code modification. A flexible synchronization policy determines how frequently the origin database is replicated to all middle-tier caches. The database cache stores tables, packages, procedures, and functions. Cache management is highly configurable and hit/miss statistics are available through the Oracle Enterprise Manager console.

Oracle9*i*AS Database Cache is both an in-memory and disk-backed cache. This combination doesn't limit the cache size of the memory footprints and also means that the cache is not "cold" immediately after a machine reboot.

"Advanced features such as wireless integration, portals, business intelligence, Web services, collaboration, and process integration can make it easier for developers to create e-business applications"

Load Balancing, Fault Tolerance, and Clustering

Load balancing deals with response times and the act of distributing connection loads across multiple servers. Fault tolerance ensures no single point of failure. Clustering involves grouping together independent nodes to work together as a single system, and architecting for high availability and scalability. Clustering is used to load balance and provide fault tolerance.

Oracle supports many ways of load balancing:

- ***Round-robin DNS for distributing HTTP requests across the cluster:*** This simple mechanism works well for Web traffic in which multiple IP addresses are assigned the same name in the DNS name space. Requests are alternated between the hosts that are presented in rotational order.
- ***Use of hardware load-balancing devices and IP sprayers:*** To address the scalability problem, a router or "IP-Sprayer" is placed between the clients and a cluster of application servers to spread the load evenly over the nodes.
- ***Spawning a number of HTTP processes to handle load:*** This increases the ability of the server to handle thousands of client requests. Load balancing is used to distribute the load among the running instances.
- ***Using Oracle HTTP Servers in reverse proxy mode:*** Oracle HTTP Server can be used in front of origin servers simply configured as a reverse proxy (i.e., proxy for content servers). This mechanism relieves the origin server load by taking advantage of the caching features of the proxy server.

Routing Requests to Improve Response Times

In addition, Oracle9*i*AS offers techniques to improve the performance of servlets, EJBs, and J2EE components delivering dynamic content. J2EE components rely on "HTTP" for communication

APPLICATION REQUIREMENTS	IMPLEMENTATION REQUIREMENTS
Access from different browsers/devices	***Presentation tier:*** Components to render the application for different browsers/devices
An interactive experience to create an itinerary	***Application tier:*** Session management components to dispatch user's requests and control flow through the application
Validation to create a "valid" itinerary	***Business tier:*** Business logic validation based on ticketing rules, user's profile, etc.
Manage user profiles	***Data tier:*** Components to query and store persistent business objects
Make secure purchases/refunds	Security and transaction management ***services***
Interact with other itinerary services (e.g., hotel and rental car reservations)	Messaging and transaction ***services*** that span the business and data tiers
Fault tolerant, minimize downtime	No single point of failure

TABLE 1 Logical partitioning of components

between Web clients and servers, and “RMI/IIOP” for communication between objects requesting service from each other in a distributed computing environment.

When an HTTP request arrives, the load-balancing device redistributes the load over Web caches that sit in front of the application server farm. Oracle9*i*AS Web Cache distributes HTTP requests according to the relative capacity of each application server, which is configurable. If one of the application servers in the farm were to fail, Web cache automatically redistributes the load among the remaining servers. Oracle9*i*AS Web Cache reduces application server load not only on Oracle9*i*AS, but on other application servers too.

Balancing the Load of Servlet Processes

Oracle HTTP Server load-balances servlet processes that use the Apache JServ servlet engine. Several JServ processes can serve single or multiple instances of Oracle HTTP Server using a weighted load-balancing algorithm that can be configured, depending on the load-handling capability of the target machines.

Providing Fault Tolerance Using Listeners

Oracle9*i*AS listener/dispatcher architecture provides a fault-tolerant, resilient environment without a single point of failure. With this architecture, each physical machine has a listener on a designated port and a number of dispatchers to service J2EE container requests. The bridge in this paradigm is that each dispatcher registers itself with any number of nodes in the application server farm. Thus, if a particular node is no longer able to service requests, the listener will send incoming requests to another dispatcher on another node. It’s important that an application server redirect both HTTP and IIOP requests intelligently for the purpose of load balancing. Redirection is essential to load balance at a protocol level; however, there’s no concept of “redirection” in HTTP – but there is one for IIOP. Oracle leverages its listener/dispatcher architecture with Apache modules to provide HTTP redirection. For example, mod_ose load balances servlet requests across nodes by redirecting HTTP requests.

Oracle9*i*AS provides mechanisms to load balance incoming requests be they in IIOP or other network formats in multithreaded server mode. This has great benefits to enterprise components residing on multiple nodes. The listener process has the ability to load balance across these nodes using IIOP redirection. This is possible because of the listener’s ability to inspect IIOP headers for session IDs and redirect them to the appropriate node. Incoming IIOP connections are redirected to dispatchers on the node with least load. The load is computed based on a number of parameters including CPU utilization and number of sessions in progress. With load balancing you can split the load across multiple servers. The Listener process within Oracle9*i*AS performs load balancing by distributing EJB lookup across multiple application servers. Services that are located in a number of places can be grouped together by specifying a service name in the URL. Listener handles HTTP connections by transferring such requests to configured dispatchers on a node.

“ Ideally, a developer balances the trade-off between **data transfer / network overheads and fault tolerance / scalability requirements** by distributing the components accordingly ”

A Glimpse into J2EE Best Practices

This section covers some of the best practices to keep in mind when writing J2EE applications. Using a flight reservations system deployed on an application server, several issues central to J2EE application development are addressed. The goal is to present a design based on the J2EE blueprint (http://Java.sun.com/j2ee/blueprints/) and to demonstrate the simplicity with which you can leverage the underlying application server for infrastructure services.

Design Architecture

To meet these requirements, the application should be carefully designed using interfaces and design patterns (for example, Model-View-Controller, façade, proxy). It should be logically partitioned as components in different tiers (see Table 1).

Figure 2 depicts the components for the end-to-end design of this J2EE application.

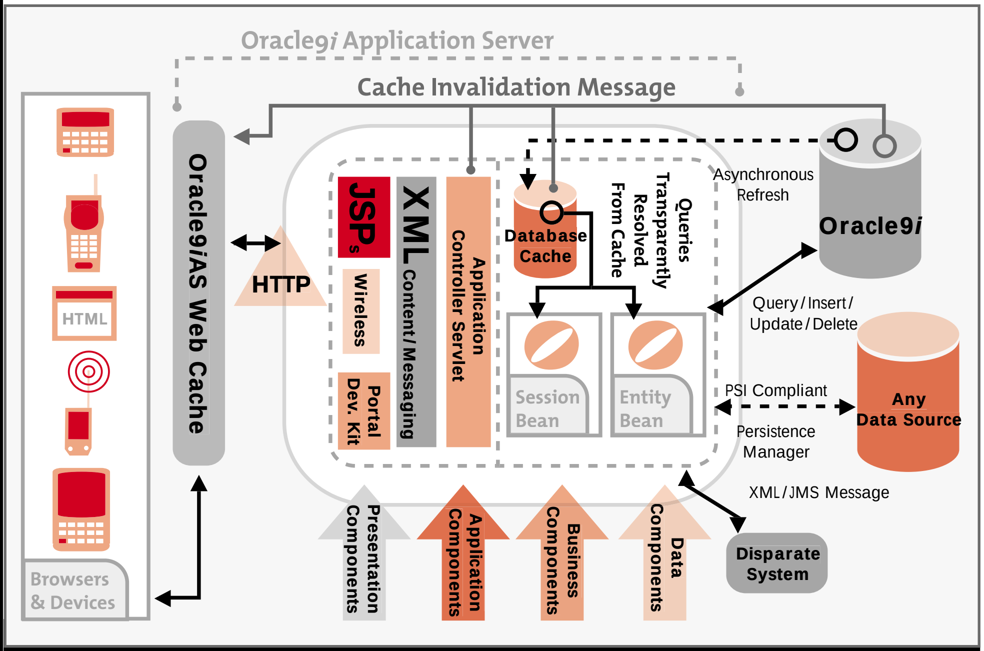


FIGURE 2 Flight reservations application design

Presentation Tier

XML is used as a standard for information exchange. For example, given the XML content for the “Pick a Flight” screen, it can be transformed (using XSLT) into HTML for a Web browser or WML for a WAP device.

For Web users, JSPs are ideal to render this content as a rich HTML presentation. Oracle9*i*AS includes an XML Development Kit (XDK) with a DOM/SAX parser, XSLT processor, Oracle XML Schema Processor, and other utilities. Oracle9*i*AS also includes a wireless and portal framework to seamlessly transform content to be part of a portal or rendered for any number of wireless devices (Palm, WAP, Rim). Thus, the “Pick a Flight” screen is generated once, but is available through a Web interface, user-customizable portal, and wireless devices.

Application Tier

Regardless of which browser/devices are used to access the application, requests from concurrent users should be coordinated and tracked by the application tier. This tier manages the state of each user in the system and logically guides them through the process of building their itineraries. For example, this tier will know that after a user has selected an itinerary, he or she should be offered "discount rentals," before being taken to the "checkout" screen.

Servlets are suited to dispatch browser requests and perform controller functions (MVC pattern). In its post() method, the servlet should accept an XML message and evaluate the message in a Finite State Machine (FSM). For example, when a customer accesses the site, the servlet receives an XML message and establishes an HTTP session for this user. It then uses an FSM to handle the message, and responds with an XML document containing the list of flights for those sectors. This is rendered as HTML and presented as a list of flights from which the user selects a single entry.

Typical servlet engines use threads to isolate concurrent users. This becomes a bottleneck when there are a large number of users and a stateful application. A good design will limit session state to only the essentials; however, even this can cause resource (thread) contention within the container. Oracle's approach is to use a JVM architecture that isolates concurrent users. The result is less contention and a container that's more resilient to application failures within an individual session.

Business Tier

When creating the itinerary, the Application Tier will make calls such as validateCustomer(Customer customer) and handleReservationEvent(ReservationEvent re) to the Business Tier to validate the user's entry.

These methods will evaluate against business rules stipulating what constitutes a valid reservation for a particular customer. When the itinerary is finished, a session bean will make the reservation via a method call such as createBooking(Itinerary itinerary).

In certain business scenarios the session bean may interact with remote systems through XML messages. Making a JMS call from a session bean frees the developer from worrying about the distributed transaction spanning these remote systems. The container will provide this service.

Business processes today involve getting many types of information to multiple people according to constantly changing rules. The Oracle Workflow engine is integrated into Oracle9*i*AS to automate and route information according to business rules. Workflow supports both synchronous and asynchronous processes. For example, when the booking has been completed, postnotification function executes to send an itinerary to the user.

Data Tier

The persistent objects underlying Flight Reservations (e.g., PassengerManager and FlightsController) are entity beans. These objects are stored as rows in database tables, but leverage the container to look up, store, and manage a read-consistent cache of this persistent data in the middle tier.

Most J2EE implementations would use a database cache in the middle tier to minimize the overhead of fetching data from the database tier. For example information about Daily Flight Specials, Airport Locations, and City Tour Spots (i.e., relatively static data) should be cached ideally in the middle tier to avoid frequent network round-trips.

Entity bean containers typically support container-managed persistence (CMP) for relational data sources. But what if some of the persistent data (e.g., Flights) comes from nonrelational data sources (e.g., a hierarchical database)? Oracle offers Persistence Service Interface (PSI), a Java API that allows CMP providers such as Oracle Business Components for Java to manage O/R mappings from entity beans to database entities. Using such an interface makes it possible to map entity beans to any data source including other relational databases and even nonrelational databases. Thus these legacy sources can still benefit from the look-up, transaction, and caching simplicity provided by the container.

A common criticism against entity beans is the processing overhead imposed by the container, especially for fine-granularity beans. The Oracle9*i*AS container resides in the same shared memory and address space as the middle-tier application cache. This minimizes the container-imposed and network overheads since the relevant persistent data can be cached in the middle tier using the Oracle9*i*AS Database Cache.

Deployment and Runtime Architecture

Figure 2 depicts all the components residing within a single instance of the application server. This is one possible deployment scenario, but is usually not the case in a production environment, where it's best to distribute the components across many nodes to avoid scalability bottlenecks, and cater to fault tolerance.

Factors to consider when distributing components should be the proximity of components to the underlying data (i.e., avoid network round-trips) and how to distribute any resource bottlenecks (i.e., CPU, memory, I/O) across the different nodes. Ideally, a developer would balance the trade-off between data transfer/network overheads and fault tolerance/scalability requirements by distributing the components accordingly. For example, the servlet components may be CPU intensive, while the session bean and entity bean may be typically memory and I/O intensive. Should these components reside on the same node or be distributed across different ones? Since the balance is sometimes hard to find, the application server must be flexible enough to seamlessly distribute components to different nodes as necessary.

Figure 3 shows an *extreme* example of deploying each component on a separate tier to show the granularity to which the application server scales and provides fault tolerance. We've

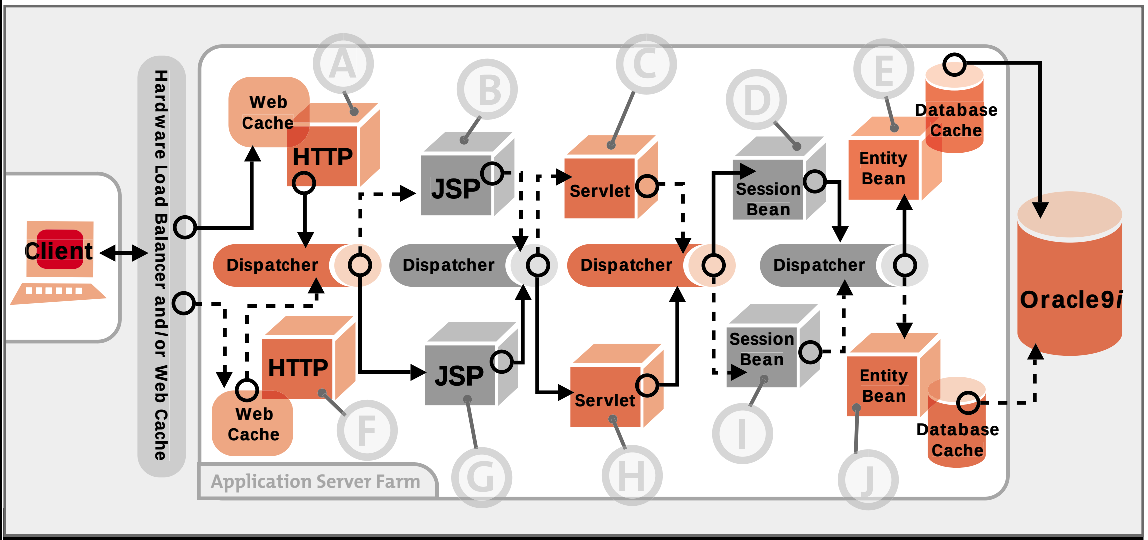


FIGURE 3 A potential Oracle9*i*AS deployment architecture

deployed the components into five tiers, across 10 physical nodes in the application server farm:

- Two nodes for HTTP listeners
- Two for JSP components
- Two for servlet components
- Two for session bean components
- Two for entity bean components

In this example the dispatcher on Machine C will register itself with the listeners on Machines B and G. The dispatcher on Machine I registers itself with the listeners on Machines C and H, and so on. An instance of the Web cache is placed in front of this "farm" to serve as a reverse-proxy, load balancer, and request router. Thus each tier can tolerate a machine failure and continue to service incoming application requests.

When a user accesses the flight reservations Web site, http://flights.oracle.com/, to make a travel reservation, the request first hits the Web cache listening on that host/port combination. If the request matches a recently cached item (based on predefined cachability rules), it's served directly from the Web cache and doesn't even hit any of the application server nodes.

If it were a "new" request, then the Oracle9*i*AS Web Cache, based on its predefined load-balancing criteria, would hand the request to a particular HTTP instance (i.e., Machine A or F). The HTTP Server then needs to invoke the JSP to render the user interface. Assuming the request hits Machine A, the HTTP listener on that machine will try to route it to any of the dispatchers serving JSPs (i.e., Machine B or Machine G). If Machine B is not available, the request will be sent to Machine G and vice versa.

The JSP then invokes the servlet to start the Flight Reservations application and keep track of the users' itinerary (i.e., HTTP session) as they build it. This request will be routed either to Machine C or H. Once the request has been routed to a Machine (say, H), and an HTTP session is established, all future requests from the same client will be routed automatically to the servlet instance on Machine H (using sess_iiop protocol).

The dispatcher architecture will similarly route this request to Machine D for business processing and to Machine E to query data from the entity beans. As the application queries persistent data (e.g., through the FlightController entity beans), the Database Cache Management Engine intelligently resolves the query by fetching data sets from the cache.

Similarly, changes to the underlying persistent data in the database tier (e.g., flight schedules) trigger an XML invalidation message to the Oracle9*i*AS Web Cache so that a subsequent request goes to the origin server to avoid serving stale data.

Conclusion

This article discussed the essential capabilities of application servers. To understand application design principles and ways in which J2EE applications can leverage the underlying application server infrastructure, we walked you through a simple J2EE flight reservations system. Using Oracle9*i*AS as an example, you've seen how to design, develop, assemble, and deploy J2EE enterprise applications.

Author Bios

Christopher G. Chelliah is a principal solutions architect at Oracle Corporation. He has been a project leader and active member on many projects involving distributed object-oriented systems based on a CORBA/Java/EJB/Internet platform.

Sudhakar Ramakrishnan is a principal member of the technical staff at Oracle Corporation. He is a member of the Internet Platform group, and has more than six years of development experience in areas of user interface, protocols, distributed computing, and component frameworks.

sudhakar.ramakrishnan@oracle.com

chris.chelliah@oracle.com

2001: The Year of Web Services

History, trends, players, predictions...

WRITTEN BY
FLOYD MARINESCU

The year 2000 saw J2EE compliancy move out of the realm of marketing and into nine shipping products. The threat of commoditization of J2EE application servers forced vendors to switch gears in 2001 toward leveraging J2EE servers as platforms for Web services, wireless, and EAI development.

The dominant players in the game surfaced, but a variety of strategic takeovers may reshape the J2EE app server market in 2001. This article walks you through the events, trends, and players of last year and describes the changes in the application server market landscape today.

2000: Year of the J2EE-Compliant Arms Race

For the app server market 2000 will be remembered as the year of the J2EE-compliant arms race. The first step toward compliance was offering a complete suite of J2EE products. The second: these products had to pass Sun's J2EE compliancy test suite, which consists of about 5,000 tests designed to certify that the compliant application server can run any J2EE application.

Allaire, owner of the popular servlet engine product JRun, purchased an EJB application server vendor and announced the new complete JRun J2EE server in June 2000. Oracle Corporation became a J2EE licensee at the end of May 2000 and announced full JSP, servlet, and EJB support in July 2000.

Also in July Bluestone, a traditional middleware powerhouse, announced servlet and JSP support to complement its popular Sapphire/Web EJB server with its new Total-E-Server product launch. Unify corporation began shipping the full suite of J2EE products around the same time.

By the end of summer, vendors that hadn't yet integrated complete J2EE offerings were not even on the radar of most analysts and project managers, and at this point catching up was very difficult.

The race was on. Now that J2EE vendors had complete and integrated J2EE product offerings, the focus switched to J2EE compliancy. Leveraging its competitive advantage as the J2EE specification provider, Sun was the first to ship a J2EE-compliant server with its IPlanet Application Server in June 2000. Other vendors engaged in an amusing battle of press releases, during which they all fought to be remembered as the first independent company outside of Sun to pass the J2EE compliancy test suite.

The first such announcement came from BEA on July 31, 2000. On September 8 ATG made a similar claim regarding its own application server. On October 19 Sybase announced it too had passed the tests, with Iona issuing a similar press release on November 15. Silverstream followed on November 17, Borland on December 15, and Bluestone on December 19.

Commoditization of the Servers: The Great Debate

In August 2000 a new debate flared up on TheServerSide.com: How do you choose between vendors once all the vendors are J2EE compliant?

Once a J2EE application has been written, what would differentiate the $1,500 Orion Application Server from the $30,000 Bluestone Total-e-Server? The unpopular opinion was that, essentially, shopping for J2EE servers would become like buying apples from a grocery store. That is, small vendors who couldn't afford to produce the best quality or who had small marketing budgets would disappear in a matter of time, leaving two or three industry giants as the sole providers of J2EE servers.

Another theory was that if J2EE servers commoditize, open-source servers like jBoss or low-cost commercial application servers like Orion and Unify eWave would be in a position to bring down companies like BEA. From a purely J2EE API perspective, jBoss, Orion, and BEA WebLogic are identical, so why would you pay extra for WebLogic when they can use jBoss?

Scott Dietzen, CTO of BEA, believes that "It is hard for a complex software stack to commoditize." Just because two servers implement the same specifications doesn't mean that the two servers will be considered identical. Dietsen points to SQL servers as an example. "SQL was a standard, but SQL servers did not become commodities." The emergence of the SQL specification did not allow cheap SQL databases to steal market share from the giants. Rather, it was the underlying quality and stability of implementation that won, resulting in Oracle's becoming the leader in the database industry.

2001: The Year of Web Services, Wireless, EAI, and Commerce Servers

"The application server days are over. The future is integrated offerings. J2EE compliancy will just be one of the many check boxes on the list," says Dave Brown, technical director of GemStone Systems.

In 2001 the focus of vendors is moving away from the J2EE server and into complementary products that will be built as J2EE applications, running on top of their existing J2EE servers. These new applications will leverage built-in support for transaction handling and scalability provided by J2EE servers, providing a fully integrated platform for development. ColdFusion is one such exciting example. Macromedia/Allaire plans to port the ColdFusion development platform to a generic J2EE application that can run on top of any J2EE-compliant application server.

The trend toward supporting complementary offerings built on top of J2EE began in 2000 with the emergence of the commerce server. Commerce servers, such as those provided by BEA and SilverStream, are out-of-the-box B2B and B2C frameworks combined with simplified rules-based development environments targeted at business analysts and script-level programmers. These products allow rapid visual development with a focus on personalization and customer relationship management features (CRM).

Given recent predictions that the B2B market could reach anywhere from $1.5 trillion to $8 trillion in revenue by 2004, B2B-enabling technologies such as Web services and enterprise application integration (EAI) will be a key area for vendors to support.

In a nutshell, Web services are a way for businesses to expose their services programmatically over the Internet, leveraging standards-based protocols and document interchange formats to execute any kind of online transaction. Web services initiatives such as ebXML and UDDI are currently the realm of early development, but should serve as viable deployment options by the end of this year. Vendors with investments in XML and Web-based protocols stand to ride that wave when it mushrooms at year's end.

Unlike Web services, EAI is a fundamental complement to J2EE products, that is, a requirement for enterprise-scale applications. Essentially, EAI products are message-oriented middleware (MOM) products (IBM MQSeries, MS BizTalk) that tie together heterogeneous systems within an enterprise by supporting custom protocols and document exchange, often based on XML.

The final major trend to capitalize on for 2001 is the wireless explosion. Most vendors have added the word *wireless* to their Web sites, with WAP gateways and component frameworks for supporting wireless devices.

App Server Market Share at the End of 2000

With no one clear survey on application server market share, it's difficult to properly gauge the market share of J2EE application server vendors. In August 1999 the Giga Information Group published a report that predicted that BEA and IBM would share the market lead at 24% market share each, with iPlanet third at 9%.

Recent vendor presentations, however, have pegged the market at WebLogic 32%, WebSphere 16%, and Sybase 15%. The most recent statistics came from a Web-based survey run by Giga in December 2000. The survey gave BEA a 56% market share, 33% for IBM, 3% for IPlanet, and 7% for all others combined. However, since the survey was Web-based and not controlled, these figures are probably not reflective of real market share.

All of the surveys are consistent in placing BEA as the market leader in 2000, followed by IBM. A clear third place isn't apparent at this time, but it's likely a close tie between Sybase, iPlanet, SilverStream, Iona, and HP/Bluestone.

Three Major J2EE Vendor Acquisitions Level the Playing Field for 2001

Three acquisitions that began in 2000 added three new giants to the J2EE arena: Brokat Infosystems/Gemstone, HP/Bluestone, and Macromedia/Allaire.

Brokat is a European software giant, a powerhouse in the European financial, integration, and wireless markets. Last July Brokat strategically expanded its offerings by acquiring Blaze Software, maker of rules engines for enterprise applications, and GemStone Systems. The Gemstone/J application server product was bleeding market share loss. However, the product itself stood in a league of its own, recognized by many as one of the most advanced application servers available due to its powerful object database-based background and multi-VM clustering features. This year Brokat plans to focus on being the wireless champion, utilizing its new software platform to build industry-specific wireless applications, with an initial focus on software for Brokat's strong financial services client base.

Last October technology giant Hewlett-Packard entered the middleware market by acquiring Bluestone software, a longtime veteran in the application server market. HP and Bluestone were the ultimate marriage. HP brought trusted and established hardware and software deployment platforms combined with an extensive sales and marketing force, while Bluestone brought a mature and powerful J2EE platform to the table. Leveraging this new complete offering, HP announced the Net Action campaign (a competitor of similar offerings from Microsoft, Sun, and IBM) in February. Net Action is an integrated platform for enterprise and Web services development.

Macromedia, the undisputed leader in Internet presentation layer tools (Dreamweaver, which has 70% market share in visual HTML tools) and rich media (Flash, which has 96% market share), announced its acquisition agreement with Allaire in January of this year. Allaire owns the JRun application server and ColdFusion, a popular server-side development platform that is a predecessor of J2EE. The acquisition of Allaire combines a client base of approximately 2 million users comprising client-side Macromedia developers, ColdFusion developers, and JRun developers. If Macromedia can properly integrate its offerings into a single platform for client- and server-side development, combined with a strategy for migrating its existing client base to this new platform, it can become an overnight market-share shark. Unfortunately, the merger won't be complete until mid-2001, placing Macromedia at a time-to-market disadvantage with its competitors.

The Application Server Market Outlook for 2001

The market share winners for 2001 will be composed of companies that provide J2EE-compliant servers with support for the latest standards, such as EJB 2.0 and JCA 1.0, but also earn recognition for powerful, complementary, and integrated offerings above and beyond J2EE – EAI and commerce server offerings, Web services support, or wireless support.

BEA WebLogic was the market share leader for 2000. I believe this position will be maintained in 2001. Its server was bundled with comprehensive documentation and running examples. Its application server was the first to be bundled on a CD with popular Java magazines back when high-speed, home-based Internet was not readily available. Being the first to implement many J2EE APIs has made the WebLogic application server a favorite among thousands of independent software vendors (ISVs).

All other factors aside, the ISV client base alone could keep BEA on top in terms of market share. Not only has WebLogic achieved a critical mass of acceptance, BEA has also invested heavily in the technologies that will differentiate vendors from one another in 2001: XML integration (eventually leading to Web services support) and commerce server offerings, personalization, process/workflow automation and wireless support. Only a company with very deep pockets and strong developer incentives will succeed in driving ISV support away from BEA.

I believe that IBM WebSphere will retain its position as the No. 2 enterprise development platform. Contrary to popular opinion, I don't think IBM will trail as far behind BEA as many think. On June 29, 2000, IBM announced that it would invest $1 billion in the WebSphere platform, in technology as well as the WebSphere consulting, sales, and marketing arms. The effects of this will be felt in this year, since IBM will hopefully invest significant resources into finally keeping up with the latest specs.

IBM has traditionally been and will likely remain the champion in EAI and process/workflow automation space. What is required is for IBM to invest in J2EE hooks to tie its existing proven products into J2EE. IBM recently announced an all-Java version of its own commerce server offering. IBM also was one of the first companies to provide wireless and XML support through its alphaWorks initiatives and the Wireless Transcoding Server.

Finally, IBM will likely become the first-to-market in Web services product offerings, as the company is intimately involved in the specification process of Web services initiatives, including UDDI, ebXML, XAML, and more.

The No. 3 position in the Enterprise Java space is controversial. Last year, depending on whom you asked, another company had the honor. Whoever gets it this year – it could be Brokat, SilverStream, HP/Bluestone, IONA, Sybase, Macromedia, ATG, Borland, or iPlanet – is unlikely to lead the competition by much of a margin.

Of all the products listed, I believe HP/Bluestone has the greatest chance of taking over the No. 3 spot in server-side market share. Bluestone has learned from WebLogic and GemStone by providing comprehensive J2EE trail maps and reference implementations to help get developers up and running. Running on top of a mature app server with powerful success stories (such as American Airlines' Sabre System), Bluestone has invested in all of the trends that became apparent in 2000, with product offerings for XML, wireless, commerce, personalization, and EAI.

The HP acquisition of Bluestone will place this new company in a unique position to take on the giants. HP will likely mobilize its massive sales/marketing force around its new middleware offering, as well as invest significant amounts of money in Bluestone products. If HP/Bluestone can execute properly, it has a chance at gaining market share in 2002 and perhaps take on the giants in 2003.

Conclusion: The Future of J2EE Application Servers

In 2001 J2EE will be used as the underlying platform for developing vertical-niche applications and frameworks for specialized applications such as Web services, wireless, CRM, process/workflow automation, and EAI.

The job of the ISV and component developer will be to write these niche frameworks to be as portable as possible. An explosion in the J2EE component market is thus likely this year. The job of the project manager will turn toward deciding which of these applications should be combined to solve business problems faster. In the best-case scenario, project managers will still be able to choose the underlying application server on which these portable frameworks will be deployed. However, many companies working on enterprise-scale projects will turn toward proprietary vendor-integrated solutions.

By the end of the year, the application server market will be dominated by vendors who can provide one-stop integration of tools, frameworks, and applications.

AUTHOR BIO
Floyd Marinescu is an Enterprise Java educator at The Middleware Company, a J2EE training company. Floyd designed and implemented TheServerSide.com J2EE community, and is currently working on an EJB design patterns book.

floyd@middleware-company.com

PRODUCT REVIEW

WebSphere Advanced Edition 3.5

by IBM

REVIEWED BY JIM MILBERY

AUTHOR BIO

Jim Milbery, a software consultant with Kuromaku Partners LLC (www.kuromaku.com), based in Easton, Pennsylvania, has over 17 years of experience in application development and relational databases. He is the applications editor for Wireless Business & Technology *and the product review editor for* Java Developer's Journal.

jmilbery@kuromaku.com

JAVA DEVELOPER'S JOURNAL JDJ WORLD CLASS AWARD

IBM Software Solutions
IBM Corporation
Route 100
Somers, NY 10589

Web: www.ibm.com/websphere

Test Environment:
Dell 4800 1GB RAM, 30GB disk, NT 4.0

Pricing:
Standard: $795/server
Advanced: $7,500/CPU
Enterprise: $35,000/CPU

Application servers are the one category of software product that seems to be on everyone's mind these days. No longer is there any doubt in my mind that *n*-tier applications are the future. Certain applications will benefit from a heavy-client architecture, such as desktop publishing, but most will have at least some portion of their logic running on the middle tier.

While data-intensive tasks will still run more efficiently in the database tier, key business rules and HTML-client generation will undoubtedly be handled by the application server. Every major software vendor has thrown the proverbial hat into the app server ring. Leaders are starting to emerge, but it's important to note that the market penetration for application servers is still less than 10%.

We learned a valuable lesson counting votes in Florida: it's not smart to declare any winners until more precincts have turned in their totals. However, IBM is aggressively targeting the application server market with their WebSphere line of products, which is how I came to spend the past several weeks working with their WebSphere Advanced Server and WebSphere Studio.

IBM WebSphere Editions

IBM sells a fairly large number of technologies under the WebSphere marketing umbrella, which are bundled into three versions of the application server:

- ***Standard Edition:*** Supports static HTML pages, servlets, JavaServer Pages, and XML
- ***Advanced Edition:*** Standard Edition features plus Enterprise JavaBeans
- ***Enterprise Edition:*** Advanced Edition features plus CORBA

WebSphere Standard Edition competes directly with other low-end dynamic Web server technology such as iPlanet's recently announced Web Server Enterprise Edition and Application Server 6.0 Standard Edition. WebSphere Standard Edition and WebSphere Advanced Edition are based on the same code line and are written in Java. While the Enterprise Edition shares many of the same features, it's constructed from a different code line that's based on IBM's Component Broker technology. The WebSphere Advanced Edition version 3.5 is likely to be the most popular of the three packages, and this is the software I took a look at.

IBM WebSphere Advanced Edition

Although IBM offers an enterprise-class Java development environment in the form of VisualAge for Java, JSP developers and Web site designers may find VisualAge a little difficult to learn. To address this market requirement, IBM provides a simpler development environment for Web site development called WebSphere Studio.

The reviewer's guide for WebSphere discusses both products in detail, and I recommend that you use WebSphere Studio if you plan on working with the WebSphere Server. WebSphere Advanced Edition and WebSphere Studio are packaged as separate installation kits on individual CDs. WebSphere's installation program allows you to accept a default installation process, or you can choose custom installation. The former overwrote my existing Apache Server installation on port 80 – which is my biggest gripe with every application server I've ever worked with.

Despite this small hiccup, the installation was painless and simple. The only problem I encountered was that the installer failed to create the administration user ID and password for the server. I was able to correct the problem right away by reviewing the installation manual. In fact, the most obvious improvement in this release of WebSphere is the enhanced installation and administration tools. The IBM HTTP server that comes with WebSphere can be managed through a browser interface that's crisp and well organized, as shown in Figure 1.

Both IBM and Oracle are embracing the Apache server as the HTTP server of choice for their application server products. While the choice is excellent, configuration isn't always easy. The browser-based administration tools make the management task relatively simple, compared to using Apache's arcane configuration files and syntax.

IBM also supports other popular HTTP servers, including iPlanet Enterprise Web Server, Microsoft Internet Information Server, and Lotus's Domino Application Server. Once I had the Apache server configured, I installed WebSphere Studio, which is tightly integrated into the WebSphere Server. I highly recommend that you use these products in tandem, as it's much easier to work with WebSphere through the Studio interface.

The entire application server market has rallied around the J2EE platform, and IBM WebSphere is no exception. Despite IBM and Sun's much publicized spat over J2EE licensing, I fully expect IBM to tackle J2EE certification for the WebSphere Application Server. The J2EE is chockful of required programming interfaces, but most developers will be concentrating on the big three: JavaServer Pages, Java servlets, and Enterprise JavaBeans.

The WebSphere Advanced Edition is all about J2EE programming. If you haven't had much experience working with J2EE, you may find it easier to start with JSP and servlets. Through them you get access to application data within a relational database using a simple programming paradigm. JSPs and servlets generate standard HTML pages from this data, so the resulting applications are compatible with even the lowest-common-denominator Web browsers. (You can even use servlet technology to generate browser pages for wireless devices using WML.)

IBM organizes logical groups of similar content into WebSphere "applications," which can consist of servlets, JSPs, or Enterprise JavaBeans. According to the J2EE specification, servlets and JSPs are used as the tools for generating the user interface layer of the application.

Core business rules and transactions are implemented by using Enterprise JavaBeans – session beans and container beans. The former handle the business rules and transaction logic; the latter handle the data persistence (e.g., storing records in a database).

While Sun's J2EE specification describes the various Java APIs that a vendor is required to support, actual implementation is left to the individual application server vendor. For example, vendors are required to support "stateful" session beans to implement transactions, but the actual transaction layer is up to the vendor. The speed and robustness of a vendor's EJB implementation is completely dependent on the transaction layer that the application server is based on.

WebSphere 3.5 offers a strong transaction layer on which its J2EE foundation is based. IBM supports both bean-managed persistence for EJBs – which leaves the detailed programming in the hands of developers – and container-managed persistence for DB2, Oracle, and Sybase. WebSphere 3.5 leverages JDK 1.2 and transaction enhancements such as failover and load balancing.

WebSphere supports additional standards in this release, such as LDAP (Lightweight Directory Access Protocol), and a copy of IBM's SecureWay Directory Server is bundled into the Advanced Edition to support LDAP. IBM's goal is to provide a complete, integrated stack of products from the database to the application server all the way to application development tools.

In order to manage application servers more carefully, WebSphere 3.5 includes Tivoli Ready modules that allow the server to be managed by IBM's Tivoli tools. IBM has made improvements in other areas of the server in this release as well, including integration with Domino, VisualAge for Java, and WebSphere Commerce Suite.

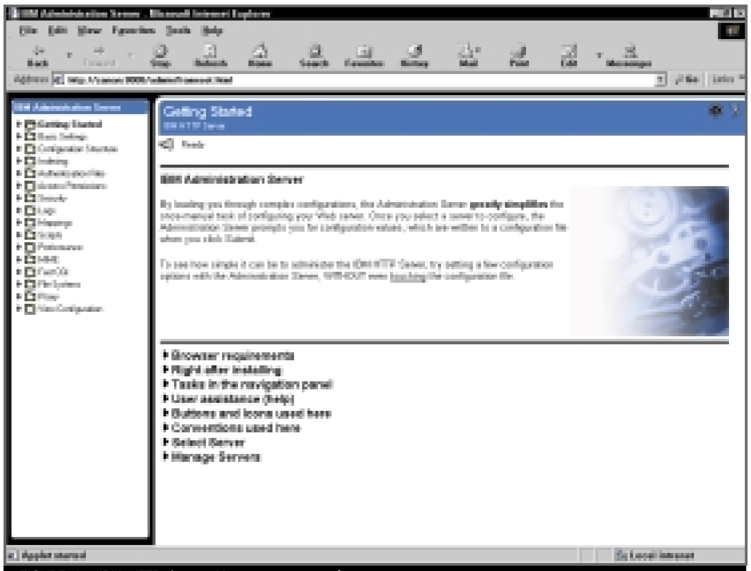
Choosing an Application Server Platform

Overall, WebSphere's biggest strength is probably its integration with the DB2 database and the rest of the IBM product stack. While the ultimate goal of the J2EE is standardization and interoperability, the reality of this vision as far as commercial products is concerned tends to vary. Vendors such as IBM that market a database, application server, and development tools will tend to have greater integration within their own stack. It's the classic "best of breed" versus "integrated suite of products" problem.

That's why I recommend that any organization looking to standardize on an application server take a very close look at the various products on the market. While they all tend to have the same set of features, implementation of those features tends to vary widely, even within the realm of the J2EE. For example, IBM provides the WebSphere Studio product that features a simplified interface for developing servlet-based Web applications. Developers who are looking for a simple platform for managing servlet-based applications will likely find the combination of WebSphere Studio and WebSphere Advanced Edition to be compelling.

Conversely, programmers who are building cross-database applications may find BEA's WebLogic more compelling, and corporations that want to leverage their PL/SQL programming skills will lean toward Oracle's 9*i* Application Server.

I don't mean to suggest that these are the only reasons to select one particular product over another, and there are lots of reasons for choosing IBM's WebSphere product line. However, I highly recommend working with these products for a while before you make a selection. The devil is definitely in the details.



IBM Administration Server

By leading you through complex configurations, the Administration Server **greatly simplifies** the once-manual task of configuring your Web server. Once you select a server to configure, the Administration Server prompts you for configuration values, which are written to a configuration file when you click Submit.

To see how simple it can be to administer the IBM HTTP Server, try setting a few configuration options with the Administration Server, WITHOUT ever touching the configuration file.

- Browser requirements
- Right after installing
- Tasks in the navigation panel
- User assistance (help)
- Buttons and icons used here
- Conventions used here
- Select Server
- Manage Servers

FIGURE 1 IBM HTTP Server Management console

Summary

IBM offers the Advanced Edition for most of the popular server platforms, including AIX, Solaris, Windows NT/2000, and HP-UX. If you're looking for an enterprise-class application server, IBM has put together a comprehensive package worth considering under the WebSphere umbrella. However, if you're interested in building JSP/servlet applications and don't need directory services integration, EJBs, mainframe access, and advanced failover capabilities, WebSphere Advanced may be more than you're looking for.

Advanced Source Control in VisualAge for Java

More source control tips

Part 2 of 2

WRITTEN BY
TIM DEBOER,
STEVE FRANCISCO &
ELLEN MCKAY

Using a software configuration management (SCM) system is an integral part of any development project. Source code is your most valuable resource and must be protected. However, with the large number of products available from many different vendors, it's essential that you choose an SCM system that will work with your favorite development tools.

The External Version Control tool in VisualAge for Java, version 3.5, allows you to interact with several external SCM systems from within the integrated development environment (IDE). Once you've used the tool to associate a project in VisualAge for Java with a group of files in the SCM system, you can add and remove classes from the SCM system, as well as check classes in and out. In Part 1 of this article (***JDJ***, Vol. 6, issue 3) we discussed several aspects of this tool, including:

- Using the team server instead of the External Version Control tool
- Automatic versioning of your SCM files
- Merging changes
- Working with class groups and resource files

In Part 2 we expand on this topic by discussing two additional aspects of the External Version Control tool:

- Refreshing your SCM files in the IDE
- Manipulating path prefixes

Refreshing Your SCM Files in the IDE

To refresh a project that's been revised, version control will compare your workspace contents with the contents of the SCM server and allow you to synchronize your project in a customized way. Files will be refreshed only if their contents have changed since you created the project or since they were last refreshed. Many different types of inconsistencies are detected during a refresh operation, such as file deletions, new versions of files created by your teammates, and potential conflicts in which changes have occurred both inside the IDE and on the SCM server. You control how these inconsistencies are dealt with and choose the course of action in each case.

Depending on which SCM system you're working with, you may be able to refresh your files directly from your SCM system, or you may first have to ensure that your working directory is synchronized with the server, then refresh the files in the IDE. This process varies depending on the capabilities and versions of the particular SCM system you're using.

If you were prompted to import your SCM files from your working directory when you added your project to version control, you might need to extract the latest copies of your files from your SCM system and place them in your working directory before you try to refresh your project. Otherwise, you won't be able to refresh your project properly because your workspace will appear to be synchronized with the working directory.

It's usually preferable to have the refresh operation use the SCM server as its source of file information rather than the working directory. Unfortunately, this is not always possible, as it depends on the SCM system's capabilities. In most cases, the EVC tool will be able to detect the best way to refresh files and you won't typically need to adjust it.

Manipulating Path Prefixes

When you import Java source or compiled class files into the IDE, they're loaded into the appropriate package in the workspace. Resources, however, are handled differently. They're loaded into the associated project_resources/<project_name> directory, which is automatically included in the project's classpath at execution time. The directory structure within the project's resource directory will be preserved when importing resources. This becomes a problem if the original file in your SCM server has any extra subdirectory levels, as those levels will be mirrored and may leave your resources inaccessible to your programs without modifying the classpath. For example, consider a file in your SCM system that's stored as:

```
data/translatable/com/mystuff/labels.prop-
erties
```

When you import this file, you'll also import by default the data/translatable directories. When your code attempts to load the com.mystuff.labels resource bundle, it won't be able to find it.

To help avoid this situation, External Version Control lets you specify a path prefix that's removed during import and replaced during export. In the above example, setting the path prefix string to data/translatable will cause your resources to strip off these directories when importing. That means labels.properties will appear on disk as:

```
ide/project_resources/<project_name>/co
m/mystuff/labels.properties
```

This lets you use the files within the VisualAge for Java IDE without abandoning the organizational structure you've created on the SCM system.

The path prefix you specify in this window will be saved and listed in the Details page of the Version Control Properties window and in the Refresh Items page of the Refresh window. The prefix will automatically be removed from any files you import using the Refresh window when they're placed in the project_resources directory.

When you import a non-Java resource file into VisualAge for Java from your SCM system, you can elect to remove some or all of its path before it's copied into the project_resources subdirectory. Enter the name of the path prefix as it's displayed in the list of available files. For example, if you import a file that's stored on the SCM server as \Source\Resources\mypackage\mine.properties, and type \Source\Resources\ into this field, the file will be imported to the following directory (which will be created, if necessary): project_resources\projectname\mypackage.properties. If you're using the SCCI handler, you must also type the name of the drive on which the file is residing.

You might remove part of or the entire path prefix for the following reasons:

- To avoid creating extra, unnecessary subdirectories
- To control the directory structure of the resource files that will later be loaded from the SCM server

When you export a file to your SCM system from VisualAge for Java you can select to add a path prefix to it before it's copied into your SCM system. For example, if you're adding a file called mypackage/myfile.properties to the SCM server, and you enter resources into the prefix field, the file will be exported as:

```
resources/mypackage/myfile.properties
```

You may want to add a path prefix for the following reasons:

- To create a special subdirectory for resource files to reside in
- To control the directory structure of resource files that are added to the SCM server or loaded from it later

All .java and .class files are automatically imported directly into your project and placed into packages. The path prefix doesn't affect the names of these file types except when adding new types to the SCM server, in which case the path prefix will prepend the file name.

Conclusion

Source code control is an integral part of the code development cycle. There are numerous software configuration management tools available to help developers track and store their source code. This availability means that developers must determine not only which SCM tool is best for their use, but also how to make that tool work with other development products.

The External Version Control tool in VisualAge for Java, version 3.5, allows you to interact with several external software configuration management systems, enabling you to take advantage of VisualAge for Java's powerful development environment while enjoying the benefits provided by your SCM tool.

AUTHOR BIOS

Tim deBoer is developing tools to build applications that run on the WebSphere Application Server. He previously worked with the VisualAge for Java technical support group, providing support to enterprise developers.

Steve Francisco is a software developer with IBM Canada and is currently working on the architecture and development of VisualAge for Java.

Ellen McKay is an information developer at IBM. Currently, she writes online help for various VisualAge for Java components, including the IDE, team programming, and external version control.

deboer@ca.ibm.com

cisco@ca.ibm.com

ecmckay@ca.ibm.com

Interfaces vs Abstract Classes in Java

How to decide which to use

WRITTEN BY
ANTHONY MEYER

Have you ever wondered why you should use interfaces instead of abstract classes, or vice versa? More specifically, when dealing with generalization, have you struggled with using one or the other? I'll shed some light on what can be a very confusing issue.

To start, I'll identify two pieces of the development puzzle: the behavior of an object and the object's implementation.

When designing an entity that can have more than one implementation, the goal is to describe the entity's behavior in such a way that it can be used without knowing exactly how the entity's behavior is implemented. In essence, you're separating the behavior of an object from its implementation. But is this separation best achieved by way of an interface or by way of an abstract class? Both can define methods without saying how they work. So which one do you use?

Modeling Behavior in an Abstract Class

As a rule, pure behavior is always modeled by interfaces and not in abstract classes. This example will model behavior in an abstract class to illustrate why.

Pretend you're designing a "motor" entity for an application that your sales department will use to sell motors. You're not modeling every aspect and nuance of a motor, but instead modeling what's important for the company and the process you're automating. (You find out what's important by talking to the users of a system. In this case it's your sales department. Good luck!)

Your sales department says that every motor has a horsepower rating, and this feature is the only attribute they're concerned with.

"You can avoid **accidentally implying an implementation pattern if** **you model behavior using interfaces**"

Based on this statement, you describe the following behavior of a motor:

Behavior: Someone can ask the motor for its horsepower rating, and the motor will return its rating as an integer.

At this point you don't know where the horsepower comes from, but you do know that this behavior must exist.

Translated into a method signature this behavior becomes:

```
public int getHorsepower()
```

Your company has several different types of motors, but given our particular application, this behavior is the only rule that applies to all of them. You look at both interfaces and abstract classes, but for purposes of illustration the motors will be modeled as an abstract class.

```
public abstract Motor{
  abstract public int getHorsepower();
}
```

You make a handful of concrete implementations of this class, and version 1.0 of the application enters production.

Time passes and you're called to create version 2.0. While reviewing the requirements for the second version, you find that a small subset of motors is battery-powered, and that these batteries take time to recharge. The sales department wants to be able to view the time to recharge from the computer screen. From their statement, you derive a behavior:

Behavior: Someone can ask a battery-powered motor for its time to recharge and the motor will return its time as an integer.

```
 public int getTimeToRecharge();
```

Translated into a method signature this behavior becomes:

```
public abstract BatteryPoweredMotor
extends Motor{
 abstract public int getTimeToRecharge();
}
```

The new battery-powered motors are implemented inside the application as concrete classes. It's important to note that these classes extend Battery-PoweredMotor as opposed to Motor. The changes are released as version 2 and the sales department is happy once again.

But business is changing, and soon solar-powered motors are introduced. The sales department tells you that solar-powered motors require a minimum amount of light energy to operate. This light energy is measured in lumens. The customers want to know this information. There's a fear that on cloudy days some solar-powered motors won't operate. The sales department requests that the application be changed to support the new solar-powered motors. From listening to their plight, a behavior is derived.

Behavior: Someone can ask a solar-powered battery for its lumens required to operate and the motor will return an integer.

```
public int getLumensToOperate();
```

In an Abstract class

```
public abstract SolarPoweredMotor extends
Motor{
 abstract public int getLumensToOperate();
}
```

Both SolarPoweredMotor and BatteryPoweredMotor extend the abstract class Motor (see Figure 1).

Throughout your application, motors are treated the same in 90% of the code. When you're checking if you have a solar- or battery-powered motor, use instanceof.

```
if (instanceof SolarPoweredMotor){ }
if (instanceof BatteryPoweredMotor){ }
```

You find out that horsepower is calculated for each type of motor so the getHorsepower() method is overloaded in each of the derived abstract classes. So far, this design looks good...

That is, until you find out that the sales department wants to sell a new type of motor that has both battery and solar power! The behaviors associated with solar- and battery-powered motors haven't changed. The only difference is you have a handful of motors that exhibit both behaviors.

The Problem with Modeling Behavior in an Abstract Class

Here's where the difference between an interface and an abstract class becomes apparent.

The goal is to add these motors with as little rework as possible. After all, code related to battery- and solar-pow-

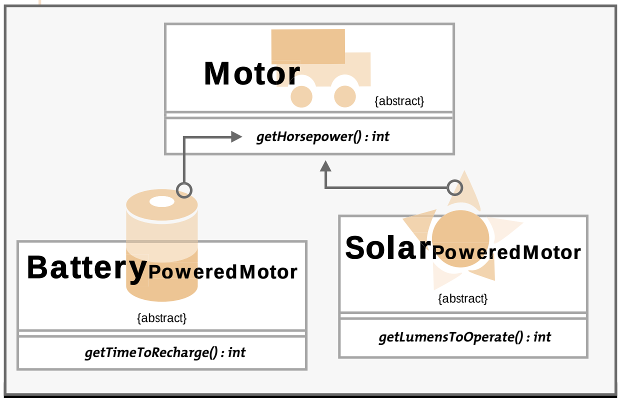


FIGURE 1 A UML diagram showing both behavior and implementation inheritance

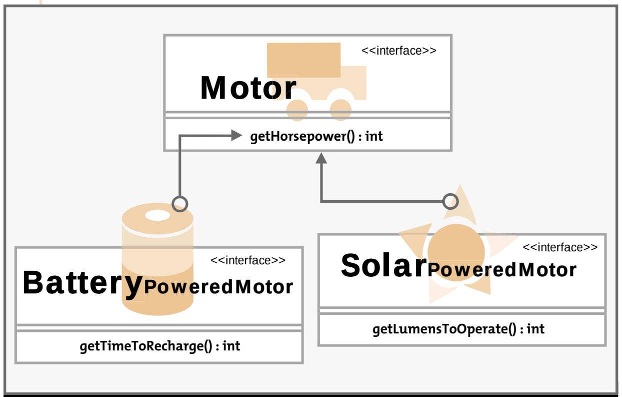


FIGURE 2 A UML diagram showing only behavior inheritance

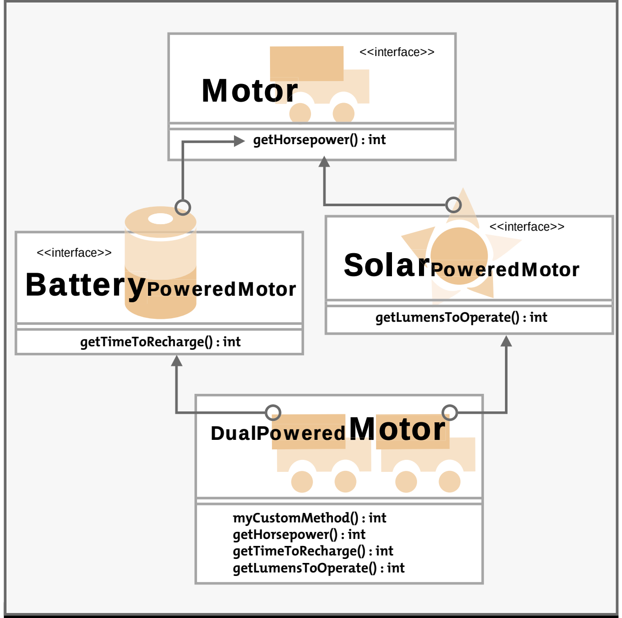


FIGURE 3 A UML diagram showing behavior inheritance from two different sources

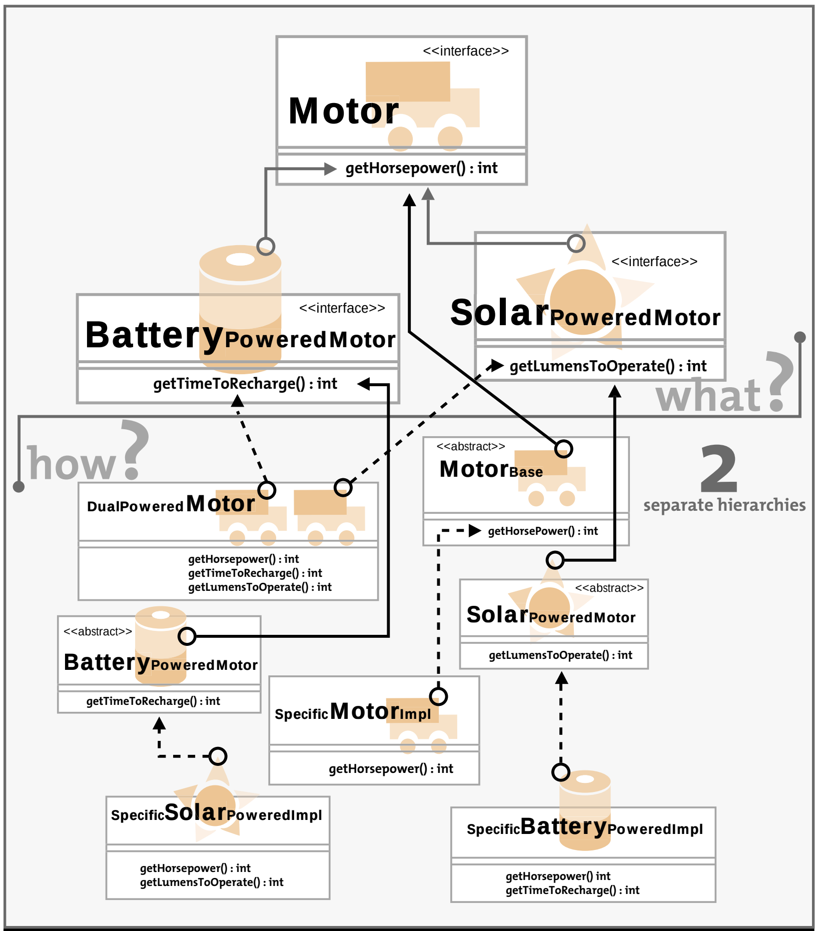


FIGURE 4 A UML diagram showing behavior inheritance and implementation inheritance derived from separate hierarchies

“Using abstract classes you can **enforce an implementation** hierarchy and avoid duplicate code”

AUTHOR BIO

Anthony Meyer is a technical director and a Java developer at Flashline.com. His experience includes the design, development and implementation of large-scale, Java-based, Internet applications in the corporate Web development environment. He has also created and implemented corporate-focused reuse strategies in the financial industry.

ered motors is well tested and has no known bugs.

You can make a new abstract class that's SolarBatteryPowered but then your motor won't trigger your instanceof when you check for solar- and battery-powered motors. The other option is to make the new motor extend either the SolarPowered or BatteryPowered abstract class. But if you do that, the new motor will lose the functionality of the abstract class it didn't extend. Technically your new motor needs to extend both abstract classes, but you painted yourself into a corner that can be solved only with a lot of special-case coding.

The reason you're having problems is that by using abstract classes you implied not only a behavior hierarchy but a pattern of implementation as well! You modeled how the motors receive their behavior instead of just saying the motors have a specific behavior.

While the phrase “Someone can ask the motor for its horsepower rating, and the motor will return the rating as an integer” implies something about the behavior of an object, it doesn't deny any behavior. Nevertheless, when you modeled with abstract classes, you created an implementation pattern that later was found to be incorrect, even though the behavior in the hierarchy was accurate.

Modeling Behavior in an Interface

You can avoid accidentally implying an implementation pattern if you model behavior using interfaces. Let's review the behavior:

Behavior: Someone can ask the motor for its horsepower rating, and the motor will return its rating as an integer.

```
public interface Motor(){
 public int getHorsepower();
}
```

Behavior: Somone can ask a battery-powered motor for its time to recharge and the motor will return its time as an integer.

```
public interface BatteryPoweredMotor
extends Motor(){
 public int getTimeToRecharge();
}
```

Behavior: Somone can ask a solar-powered motor for its lumens required to operate, and the motor will return its lumens as an integer.

```
public interface SolarPoweredMotor extends
```

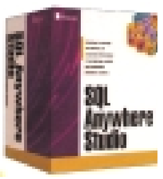
SHOP ONLINE AT **JDJSTORE.COM** FOR BEST PRICES OR CALL YOUR ORDER IN AT **1-888-303-JAVA**

BUY THOUSANDS OF PRODUCTS AT GUARANTEED LOWEST PRICES!

JDJStore.COM

GUARANTEED BEST PRICES FOR ALL YOUR JAVA-RELATED SOFTWARE NEEDS

SYBASE
SQL Anywhere Studio v7.0



Sybase® SQL Anywhere Studio is a comprehensive package that provides data management and enterprise synchronization to enable the rapid development and deployment of distributed e-business solutions. Optimized for workgroups, laptops, handheld devices and intelligent appliances, SQL Anywhere extends the reach of a corporation's e-business information anywhere business transactions occur.

SQL Anywhere Studio (base w/ 1 user) v7.0 . . . . . **$348^99**

SITRAKA
JProbe Profiler Developer v2.8.1

JProbe Profiler helps you quickly eliminate performance bottlenecks caused by inefficient algorithms in your Java code. JProbe Profiler combines a visual call graph interface, a unique Garbage Monitor, and advanced data collection technology to provide you with highly accurate performance diagnostics, including line-by-line results.

JProbe Profiler w/ Standard Support (includes JProbe Memory Debugger) . . . . **$498^99**

VISUALSOFT
JSmartGrid 1.0

JSmartGrid 1.0 is the smallest Swing grid component (162 KB) specifically designed for Java 2 Platform. v1.3.0.JSmartGrid 1.0 is a 100% Java Swing Component, offering developers an extensive API with scalable presentation abilities that can be easily integrated into applications built for the Web, databases, and application servers. It allows developers to build powerful tables, grids, or spreadsheet-like user interfaces, and can be embedded in any eBusiness and B2B application.

JSmartGrid 1.0 . . . . **$179^99**

W W W . J D J S T O R E . C O M

```
Motor{
 abstract public int getLumensToOperate();
}
```

In this way, only behavior is modeled (see Figure 2).

Now, I'll describe the solar-battery–powered motor in question:

```
public DualMotor implements
SolarPoweredMotor, BatteryPoweredMotor{
}
```

The dual-powered motor inherits behavior, not implementation (see Figure 3).

You can use abstract classes just as before, except in this case the abstract classes implement behaviors instead of defining them (see Figure 4).

Notice the two separate hierarchies. The interface defines behavior in a very pure way while the abstract class defines a pattern for implementation – including the origin of a given behavior. Notice how the bottom half of the diagram can be totally redesigned and yet the behavioral hierarchy remains intact. As long as the implementing class relies on the interfaces for behavior, the implementing class can change its parent abstract class without changing how other pieces of the code interact with it.

When to Use Abstract Classes

Now that I've fully discussed interfaces, abstract classes may seem like an evil half brother – something to be avoided. This is not the case! When you have a common implementation, abstract classes shine. Using abstract classes you can enforce an implementation hierarchy and avoid duplicate code. Using abstract classes, however, should not affect your decision to use interfaces to define your behavior.

Both parent and child abstract classes should implement interfaces that define the expected behavior if you think the implementation will change. In practice, relying on abstract classes to define behavior leads to an inheritance nightmare, while coding behavior with interfaces provides a cleaner separation of behavior and implementation. Thus it makes your code more resistant to change. If you want to modify your existing code to improve your design, I recommend reading Martin Fowler's book, *Refactoring: Improving the Design of Existing Code* (Addison-Wesley, 1999). He devotes an entire chapter to refactorings dealing with generalization.

anthony.meyer@flashline.com

Borland AppServer 4.5

REVIEWED BY NEAL FORD

AUTHOR BIO

Neal Ford is vice president of technology at the DSW Group. He also designs and develops applications, instructional materials, magazine articles and video presentations, and has authored two books: Developing with Delphi: Object-Oriented Techniques *and* JBuilder 3 Unleashed.

nford@thedswgroup.com

JAVA DEVELOPER'S JOURNAL JDJ WORLD CLASS AWARD

Borland Software Corporation
Worldwide Headquarters
100 Enterprise Way
Scotts Valley, CA 95066
Web: www.borland.com
Phone: 831 431-1000
E-mail: customer-service@borland.com

Test Environment:
OS: Windows 2000 SP1
Machine: Dell Inspiron 7500 laptop
Processor: 700MHz Pentium III
Memory: 256MB
JDK: 1.3

Let's start by getting the naming business out of the way! First there was a company named Borland. Then, for apparently no good reason, they changed their name to Inprise. The Inprise name was supposed to encompass the Enterprise products (such as VisiBroker, Entera, etc.) and the Borland name was kept for the tools (JBuilder, Delphi, C++Builder, etc.). Well, all the name change did was confuse everyone.

No, they weren't bought out. Yes, it's the same company, with the same developers. No, I don't know why they changed their name. Late last year the name experiment ended, and the name Borland now reigns supreme, referring to all the products made by the company Borland (formerly Inprise, formerly Borland). Now, is that straightened out?

The reason for the immediate digression into naming concerns the product I'm reviewing. This product started life many years ago as an application server for CORBA development. It featured excellent tools that made creating CORBA applications a breeze, writing much of the plumbing code for you. That was the Inprise Application Server, and it predated Java 2 Enterprise Edition. When J2EE became a reality, the product kept the same name but completely changed over to an EJB container (along with many other application server services). The Inprise app server existed until version 4.1. Now we have the Borland Application Server version 4.5, which I am about to officially review. The point of this history lesson is to indicate that this product is not a Johnny-come-lately to the application server market. It's based on solid technology and has been around awhile.

Installation is straightforward, running a standard InstallShield installer. Borland AppServer (BAS) installs not only the classes necessary for J2EE compliance, but VisiBroker as well. It was smart to build BAS around the VisiBroker core because VisiBroker is a well-established, rock-solid ORB. Why reinvent the wheel – and create a buggy wheel – when you already have proven code lying around? BAS installs the app server itself, which runs as a console application. It also installs the Borland AppServer console, which is the graphical management console. The BAS console is a Java Swing–based client that's very impressive.

It features a tree view on the left that shows all the pertinent running services, including different nodes for different app servers. This means you can manage multiple app servers from the same console. It also shows a great deal of information on the right-hand side about the artifact selected on the left (see Figure 1). This shows the status of a container-managed entity bean that's currently installed and available.

The console also serves as the deployment tool for EJBs. It offers wizards (real, functional wizards, not simple little apprentices) to deploy EJBs, generate JAR files for clients from existing EJBs (what a great idea), merge JARs together (another great idea), and migrate from an EJB 1.0 bean to an EJB 1.1 bean. This console is by far the best I've seen for an app server.

The third installed tool is the Borland J2EE Deployment Descriptor Editor. This tool allows you to create deployment descriptors for just about any conceivable J2EE resource including EARs, WARs, EJBs, and JNDI, and is the tool used to create BAS deployment descriptors for EJBs. Figure 2 shows part of the dialog used to create a deployment descriptor for an EJB.

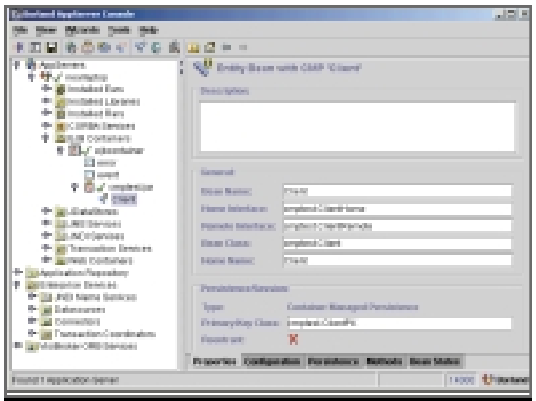


FIGURE 1 AppServer component information (right)

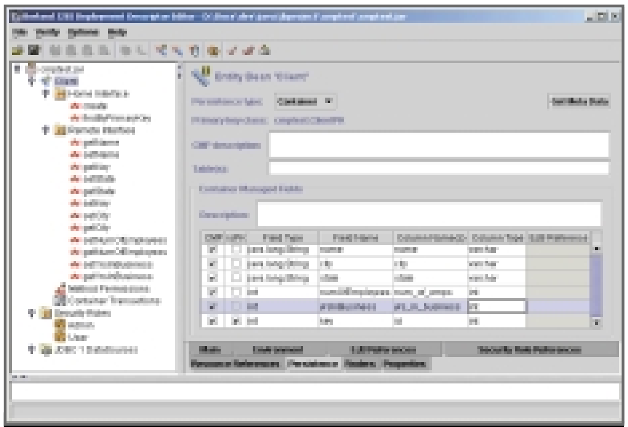
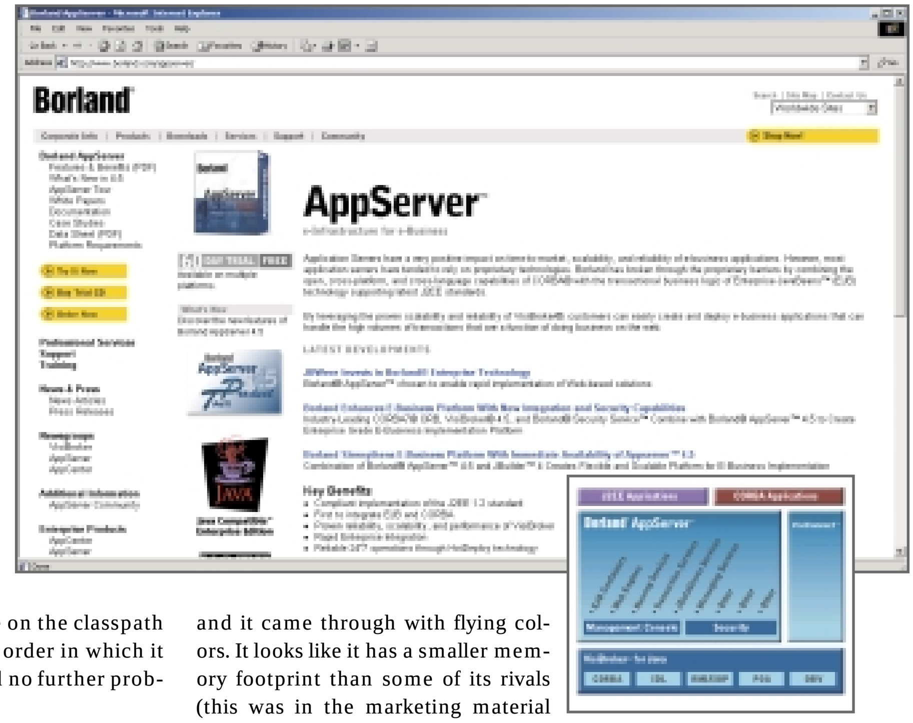


FIGURE 2 Part of dialog for EJB deployment descriptor



If you're familiar with JBuilder 4's EJB deployment tools, you'll recognize the ones from BAS – the code looks the same. Again, if you have a working code base, why rewrite it? Of course, this means that JBuilder and BAS are well integrated (as you'd expect). As with most application servers, command-line versions of all these tools exist, enabling the development process to be automated.

BAS is a very capable application server. I created and installed all different types of EJBs, including the dreaded container-managed persistent entity beans, without too much difficulty. BAS doesn't seem to care for JDBC 1.0 drivers (i.e., drivers that don't have support for javax .sql.DataSources). It's a little cumbersome to set up connection pools with these types of drivers, but this is understandable. No one uses JDBC 1.0 drivers unless they're forced to.

I did run into a few unusual naming service problems until I did some research and discovered that BAS doesn't really like to have the J2EE JAR file on the classpath that starts it. I assume that this interferes with the order in which it needs to load these classes. Once I removed it, I had no further problems.

I set up a cluster of two machines to test its ability, and it couldn't have been easier. I suspect this is because of the VisiBroker heritage, which makes it easy for machines to discover one another and use each other's services.

In summary, BAS seems to be a well-developed application server. I installed some EJBs and other services and informally stress-tested it, and it came through with flying colors. It looks like it has a smaller memory footprint than some of its rivals (this was in the marketing material and I confirmed it). Because it uses memory very efficiently, it should be able to support a larger pool of beans than most of its competitors on a single machine or cluster. The application server market is crowded and very competitive. BAS deserves to be evaluated against its competitors, where it should excel. It combines effective tools with high performance, proven technology, and ease of use.

CocoBase Enterprise O/R Mapping

BY JIM MILBERY

THOUGHT, Inc.
657 Mission Street
San Francisco, CA 94105
Web: www.thoughtinc.com

One of the more complicated issues that J2EE application developers face is the process of mapping relational data to EJBs. The J2EE specification provides EJBs as the mechanism to persist objects into a database. They certainly solve lots of problems for developers, especially in the areas of transaction management and distributed computing.

EJBs come in two basic flavors: session and entity beans. Conceptually speaking, session beans are simpler to understand as they map fairly easily to logical business transactions. Entity beans provide the glue that allows a session bean to interact with the relational database tables that manage the actual data. Database tables are relational and entity beans are objects; combining these two technologies may look simple – but it's a complex process under the covers.

Fundamentally speaking, working with entity beans and a relational database is an exercise in data mapping. RDBMS engines such as Oracle store data in tables and columns. Thus customer data is stored in a "customer" table and information relevant to the customer is stored in columns. All the data for a single customer within the customer table is equivalent to a "record." From the EJB perspective, customer data is represented by a customer "class" and the data elements are represented by "attributes." Conceptually, the mapping process is a simple one. Each database table is an EJB class and each and every column in the table becomes an attribute. Individual customer records are instantiated as EJB objects as necessary. Although it sounds simple, it can be complex to implement this type of mapping.

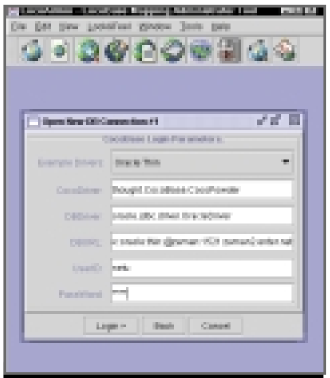
Entity beans come in two flavors, bean-managed persistence (BMP) and component-managed persistence (CMP). If you have connected EJBs to your database, you're probably familiar with BMP entity beans. With them you connect your EJB object with a JDBC driver that works with your target database. You're responsible for writing all the SQL code so you're able to optimize your application for the target database. Many first-generation EJB applications were constructed this way. The downside to this approach is twofold. First, you're locked into a single database vendor and the process of porting between databases can be a complex task. I'd argue that the second issue, productivity, is the more serious problem. Handcrafting SQL statements for a database is a huge loss in productivity. All the leading relational databases have high-speed optimizers and fast transaction processing engines. Ideally, then, EJB developers should be able to reverse-engineer preexisting relational databases and generate the persistence layer automatically.

A solution to this problem comes in the form of dynamic data mapping. A number of third-party vendors offer highly sophisticated data mapping tools for many of the popular J2EE application servers. One company garnering a lot of press and attention in this market is San Francisco-based THOUGHT, Inc. The developers at THOUGHT, Inc., have built an enterprise-class object-to-relational mapping product called CocoBase. CocoBase is packaged as an administration utility and a code-generation layer that works with most of the leading application server engines.

CocoBase plugs into your favorite database via JDBC drivers. Once you've connected to your database, CocoBase creates both BMP and CMP beans for your relational database tables. You're free to map multiple objects to a single database table and/or map multiple tables to a single object. For example, you might create different EJB objects to represent "good customers" and "bad customers." CocoBase generates a mapping file so you can change the mapping layer without having to modify the code or recompile the application. Developers are then free to concentrate on the business logic instead of wasting valuable programming time writing endless SQL statements. CocoBase uses a three-step process for building EJBS:

1. Using the CocoAdmin GUI tool, the developer connects to the database and visually maps tables to objects (and vice versa).
2. CocoAdmin generates CMP/BMP bean code for each object for the specified J2EE application server.
3. EJBs are loaded into your EJB server and are ready to use.

Such an approach offers great potential. First and foremost, it can improve the productivity of the development team. Database access and deployment code is not rocket science, but it can require lots of programming resources. An O/R mapping solution such as CocoBase can eliminate this task from your development list and also dramatically improve overall application performance.



CocoBase administration interface

Since the developers at THOUGHT, Inc., concentrate all their attention on the persistence layer, they're able to optimize the performance of this portion of the application. As an individual application developer you may not have the time or resources to build optimizations in your persistence layer. CocoBase generates some sophisticated performance optimizations directly into the EJB code that's generated. For example, it uses a data caching mechanism to reduce the number of interactions between the application layer and the database. It adds this performance improvement at the application server level without significantly impacting the performance of the database itself.

It's important to keep in mind that the J2EE specification provides a road map for development, not the implementation details. Application server vendors offering products in the J2EE space will find it more difficult to incorporate best-of-breed technologies inside their server suites as the J2EE specification expands. This provides an excellent opportunity for vendors such as THOUGHT, Inc., to provide optimized solutions for specific parts of the J2EE specification. As both the specification and the vendor implementations mature, we'll begin to see the benefits of this type of plug-and-play at the application server layer. After all, this is exactly what Sun would have us believe J2EE is all about – true interoperability.

Summary

Object-to-relational mapping is a complex process and may prove to be one of the single biggest hurdles that must be overcome when building an enterprise-class EJB-based system. Sophisticated mapping tools such as CocoBase are worth considering as a solution for this problem. I'd encourage you to place O/R mapping solutions on your short list of technologies for speeding up J2EE development within your organization.

jmilbery@kuromaku.com

JDJ ADVERTISER INDEX

ADVERTISER	URL	PHONE	PAGE
Allaire	www.allaire.com/download	888-939-2545	35
Amazon.com	www.amazon.com		47
Appeal Virtual Machines	www.jrockit.com	46 (0) 8 50630900	19
Borland	www.borland.com	800-632-2864	13
Cape Clear	www.capeclear.com/download	866-CAPE226	57
Career Opportunities Section		201-802-3028	110-113
ComponentSource	www.componentsource.com/java	888-850-9911	17
Compoze Software	www.compoze.com	866-COMPOZE	55
Corda Technologies	www.corda.com	801-805-9400	69
Dynamic Buyer Incorporated	www.dynamicbuyer.com	845-620-9800	101
Elixir Technology	www.elixirtech.com/download	65 532-4300	87
Esmertec	www.esmertec.com	877-751-3420	37
Fiorano	www.fiorano.com	800-663-3621	79
Flashline, Inc.	www.flashline.com	800-259-1961	67
Fuegotech	www.fuegotech.com	800-355-7602	58-59
Generic Logic	www.genlogic.com	413-253-7491	80
IAM Consulting	www.iamx.com	212-580-2700	73
ILOG	www.ilog.com/jdj	800 FOR ILOG	43
Informix	www.cloudscape.com/jdj	888.59.JAVA1	29
Infragistics, Inc.	www.infragistics.com	800-231-8588	14-15
INT	www.int.com	713-975-7434	76
IONA	www.iona.com	781-902-8000	25
JavaEdge Call For Papers	www.sys-con.com/javaedge	201-802-3069	97
JavaEdge Conference & Expo	www.sys-con.com/javaedge	201-802-3069	98-99
JavaOne 2001	http://java.sun.com/javaone/		91
JDJStore	www.jdjstore.com	888-303-JAVA	102
Jinfonet Software	www.jinfonet.com	301-990-6330	45
JustComputerJobs	www.justjavajobs.com	877-905-NERD	16, 85
Leapnet	www.leapnet.com	(312) 528-2400	38
LOOX Software, Inc.	www.loox.com	800-684-LOOX	61
Netaphor	http://cyberons.com	877 NETA4SW	62
New Atlanta Communications	www.servletexec.com	678-366-3211	89
No Magic	www.magicdraw.com	303-914-8074	7
Northwoods Software Corporation	www.nwoods.com	800-226-4662	82
O'Reilly	www.java.oreilly.com	800-998-9938	31
Parasoft	www.parasoft.com/jdj4	888-305-0041	39
Pingtel	www.pingtel.com/javachallenge		21
PointBase	www.jdj.pointbase.com	877-238-8798	27
Pramati	www.pramati.com	877-PRAMATI	71
PreEmptive Solutions	www.preemptive.com	800-996-4556	75
Programix	www.jthreadkit.com		24
Progress Software	www.sonicmq.com/jdj401.htm	800-989-3773	2
ProSyst	www.prosyst.com	866-PROSYST	63
Quadbase	www.quadbase.com	408-982-0835	96
QuickStream Software	www.quickstream.com	888-769-9898	90
Rational Software	www.rational.com/jdj2	800-728-1212	115
SD 2001	www.sdexpo.com	800-441-8826	93
Sitraka Software	www.sitraka.com/greatapp/jdj	888-361-3264	4
Sitraka Software	www.sitraka.com/j2ee-jdj	888-361-3264	53
Sitraka Software	www.sitraka.com/serverchart/jdj	888-361-3264	83
Sitraka Software	www.sitraka.com/promise/jdj	888-361-3264	116
Softwired	www.softwired-inc.com	41-14452370	65, 81
Sun Microsystems	www.sun.com/service/suned/training	800-422-8020	49
Sybase	www.sybase.com	800-8-SYBASE	33
SYS-CON Media	www.sys-con.com	925-244-9109	107
SYS-CON Media Reprints	www.sys-con.com	201-802-3024	107
Talarian	www.talarian.com/jms	650-965-8050	11
The Breeze Factor	www.breezefactor.com	435-658-3433	41
Tidestone Technologies	www.tidestone.com	800-884-8665	51
Togethersoft Corporation	www.togethersoft.com	919-833-5550	6
Unify Corporation	www.unifywave.com	800-GO UNIFY	77
Valtech	www.valtech.com	888 240 6028	23
Verge Technologies Group	www.verge.com		95
WebServicesEdge Conference & Expo	www.webservicesedge2002.com	201-802-3069	103
Wireless Business & Technology	www.wbt2.com	201-802-3020	86
XMLEdge Conference & Expo	www.sys-con.com/xmledge.com	201-802-3069	104-105
Zero G	www.ZeroG.com/installs	415-512-7771	3

Web Services: The Next Big Thing

Accelerate innovation in the world of distributed computing

WRITTEN BY
JACK MARTIN

If you search under Web services in Yahoo! the results include religious supplies and services, translation services, adult entertainment, and Internet services; however, that's all about to change. Web services are going to be the next great thing.

Web services are a new model for creating dynamic distributed applications with common interfaces for efficient communication across the Internet. They're built around ubiquitous, open standards such as TCP/IP, HTTP, Java, HTML, and XML, as well as newer standard technologies such as Simple Object Access Protocol (SOAP), Web Services Description Language (WSDL), and Universal Description, Discovery and Integration (UDDI).

UDDI is a specification of Web services' information registries. Web services are discovered through a UDDI-based registry, which is distributed across the Web. In that distributed registry, businesses and services are described in a common XML format. The structured data in these XML documents is easily searched, analyzed, and manipulated.

Web services are self-contained, self-describing modular applications that can be published, located, and invoked from anywhere across the Web. Since they communicate with HTTP and XML, any device that supports these technologies can host and access Web services.

Think of Web services as Legos with logic – interlocking blocks that create open distributed systems that allow developers to quickly and cheaply make their products available to businesses and consumers worldwide. They are evolving from an object-oriented paradigm to a service-oriented one.

Web services will move Internet applications toward a service-oriented architecture that can perform functions, from simple requests to complicated business processes. Once a Web service is deployed, other Web services can discover and invoke the deployed service. Web services might process insurance claims or be a purchasing service that orders and replenishes gasoline supplies for service stations when a given inventory level is reached. This frees gas station operators from having to schedule deliveries and, more important, enables oil refiners to accurately match production with demand.

Software developers will be able to publish functions to the Internet that are packaged on a pay-as-you-go plan for a one-time or limited use, or for unlimited use by other programs or as stand-alone offerings. ASPs will aggregate Web services to provide a higher-level set of features. Applications of the future will be built from Web services that are dynamically selected at runtime based on their cost, quality, and availability.

Each Web service component in a service-oriented architecture can play one (or more) of three roles:

- *Service providers* publishing the availability of their services
- *Service brokers* registering and categorizing published services and providing search services
- *Service requesters* using broker services to find a needed service and then employing that service

Service providers can describe themselves; service requesters can describe what they're looking for, determining which requester–provider pairs are a good match.

The interaction between a service provider and a service requester is designed to be completely platform- and language-independent. It requires a WSDL document to define the interface and describe the service, along with a network protocol (usually HTTP).

A WSDL 1.0 document as defined by Ariba, IBM, and Microsoft defines services as collections of network endpoints, and uses the following elements in the definition of network services:

- ***Types:*** A container for data-type definitions using some type system (such as XSD)
- ***Message:*** An abstract-typed definition of the data being communicated
- ***Operation:*** An abstract description of an action supported by the service
- ***Port Type:*** An abstract set of operations supported by one or more endpoints
- ***Binding:*** A concrete protocol and data format specification for a particular port type
- ***Port:*** A single endpoint defined as a combination of a binding and a network address
- ***Service:*** A collection of related endpoints

The power and simplicity of Web services will accelerate innovation in the world of distributed computing.

jack@skc.com

AUTHOR BIO
Jack Martin is president of Simplex Knowledge Company, an Internet software boutique. Simplex developed the first remote video transmission system designed specifically for child care centers, which received worldwide media attention. Simplex is currently designing B2B and B2C Web-based communities.

Corda Technologies Releases ProChart Version 3.8

(Orem, UT) – CORDA Technologies Inc. has announced the 3.8 version of PopChart. New features provide backward compatibility with legacy image types, such as GIF, in their Web-based applications.

Version 3.8 also provides modules to tie into existing Web server infrastructures. The modules include ISAPI for Microsoft IIS, an Apache module, and a servlet wrapper for Java-based Web servers.
www.corda.com

Borland Expands E-Business Development Platform

(Scotts Valley, CA) – Borland Software Corporation has released Borland Enterprise Studio Java Edition, which accelerates time-to-market for Java multitier, Web, and e-commerce applications by providing a complete design, development, and process management solution.

The Studio includes application development life-cycle products from Rational Software Corporation and Macromedia.
www.borland.com

Unify Unveils Affordable Java-Based Web App Server

(Sacramento, CA) – Unify Corporation has announced the availability of Unify eWave Engine 4.0, which allows organizations to take advantage of the latest industry standards in

the J2EE platform including JSPs, servlets, and EJBs at $595 per CPU.

Usability enhancements in installation and administration, and seamless migration from Java servlets to EJB component-based development are included.
www.unify.com

Flashline, Sun Join Forces to Speed Java Development

(Cleveland, OH) – Flashline was selected to create a customized marketplace within the Forte for Java portal, a virtual neighborhood in which Java technology developers can share add-on products, components, and ideas. Developers can access, evaluate, and purchase modules and add-ons to extend the Forte for Java development environment, as well as commercial Java technology-based components that can be plugged into applications to accelerate development.

The customized marketplace is available through the Forte for Java Products and Services Marketplace at www.sun.com.

BEA Relaunches Developer Center

(San Francisco, CA) – BEA has announced the relaunch of the BEA Developer Center. Running entirely on WebLogic Server using JSPs, key improvements include easier navigation of the

content and a more scalable architecture. The site continues to leverage the single-login mechanism used for downloads and ParterNet.
www.bea.com

TogetherSoft Secures $20 Million in Financing

(Raleigh, NC) – TogetherSoft Corporation has secured a $20 million round of financing from TA Associates, a private equity investment firm based in Boston. The financing will help fuel TogetherSoft's strategic growth, sales, and marketing, R&D, and major product development initiatives.
www.togethersoft.com
www.ta.com

QA-Systems Delivers QStudio Java 1.2

(The Netherlands) – QA-Systems has announced QStudio Java 1.2., which offers the possibility of automating quality analysis and control for applications written in Java.

QStudio Java checks the Java source code and evaluates the application on essential topics, such as reliability, testability, and maintainability. A trial version

of QStudio Java 1.2 can be downloaded from the Web site, www.qa-systems.com.

Fiorano Ships FioranoMQ 5

(Los Gatos, CA) – Fiorano Software, Inc., is shipping version 5.0 of FioranoMQ 5, a fully JMS 1.0.2-compliant Java messaging server. New features include a pluggable, scalable connection management (SCM) module, Java REALMS security support, and message routing across geographically distributed servers using the FioranoMQ Repeater, which allows multiple servers to be connected locally or across geographic regions.

A complete FioranoMQ 5 evaluation kit, including the JMS test suite, is available from www.fiorano.com.

Rational Offers Power to Build Content for Unified Process

(Lexington, MA) – Rational Software has announced that members of the Rational Unified Partner Program can now create additional content for the Rational Unified Process, Rational's Web-enabled software engineering process.

Rational partners will now be able to capture and distribute knowledge specific to their areas of expertise, such as application mining or specific B2B Web architectures, via the software development industry's de facto standard engineering process.

Customers of the Rational Unified Process will also be able to build quality software faster through additional guidance from Rational partners.
www.rational.com/partners

World's First Twain Scanning with Java

(Newton, MA) – Snowbound Software is delivering the world's first proven method for providing Twain scanning from within the Java programming environment. By integrating the Twain compatibility features of their Windows Imaging SDK with their Java Imaging SDK, Snowbound has developed a way to capture a scanned image and bring it into a Java application.
www.snowbnd.com

So You Want to Be a Senior Engineer

How to gain the skills and experience you need

WRITTEN BY
BILL BALOGLU &
BILLY PALMIERI

In recent columns we've discussed the current job market, rates and salaries, and different kinds of Java engineers: those who know Java and those who understand Java.

What separates a good, solid engineer from a senior engineer who's on top of the industry and in the highest demand? It's the experience and skills in the hottest technologies.

To become a senior engineer you need to have solid working experience with J2EE, XML, EJBs, and EJB-based application servers such as WebLogic, WebSphere, iPlanet, or Enhydra, as well as experience with servlets.

This doesn't mean simply reading a book or taking a course or working in a peripheral way with these technologies. You need to spend at least six months to a year working directly with them.

J2EE has a very high learning curve, so a lot of experience is required of an engineer who claims to be at the senior level. It's still a brand new technology with a brand new infrastructure. There are still a lot of truths and half-truths around J2EE and immature applications to work around. A senior engineer with more than six months to a year of experience knows where those problem areas are and how to get around them.

If you name-drop skills such as J2EE on your resume and don't really know the ins and outs of them, you'll be quickly found out at the interview. You may fool the recruiter, but you won't fool the hiring manager. Worst of all, you stand the chance of risking your credibility and reputation with everyone.

At an interview for a senior engineer position you'll need to know the difference between container- and bean-managed persistence. You'll be expected to discuss the known shortcomings of CMP versus BMP and why one tool is better than the other.

Most important, you should be able to talk about specific instances in which you've used these different versions on different projects, as well as problems you've encountered and how you solved them.

As current technologies move from browser-based applications to wireless applications, a senior engineer should be able to describe the difference between writing an application for a cell phone with six lines of text versus a Web browser.

XML is becoming an increasingly critical skill as applications move from browsers to independent devices such as cell phones, palm devices, and thin clients (those devices that give you directions from inside your car and will enable your refrigerator to tell you when you're low on milk).

"XML connects application servers to independent devices like thin clients," explains David Young, chief evangelist for Lutris Technologies. Lutris has partnered with Nokia and Motorola to develop applications and enterprise solutions for next-generation wireless phones.

"XML is powerful because it's an agreement by the world of how to format raw data," says Young. "XML does for portable data what Java does for portable applications."

Young also stresses the importance of J2ME, a tool that will be very important for writing lean, mean applications for both smaller and wireless devices. Senior engineers also need XSL, XSLT, and wireless technology skills such as WML (wireless markup language).

If you don't have experience with these technologies, you need to get it. But where do you get it? There are many good books and courses available on J2EE, XML, EJB, and wireless technologies that are, of course, the best place to start.

Working as a contract consultant or with a consulting company is still the fastest, most effective way to pick up real-world experience with a variety of new technologies. B2B companies or those doing business over the Internet can provide that experience, but perhaps not as quickly or broadly as contract consulting gigs.

Hands-on experience with application servers may be hard to come by for the engineer-in-training who doesn't have thousands of dollars to buy a proprietary app server such as WebLogic.

Enhydra, the open-source Java/XML application server, can be downloaded for free at www.enhydra.org. This free site provides engineers with the tools to build Java, EJB, and wireless applications (the alpha version of Enhydra 4 includes J2EE). You also have free access to the experience of some 3,000 developers in the international Enhydra community.

One final note: in the engineering world there's often confusion regarding the difference between a senior engineer and an architect. An architect is someone who meets with management/clients, does white-board meetings and design, and then draws up the models that are needed. The architect needs high-level vision, diverse industry knowledge, and great people skills.

The architect's work is then delivered to senior engineers who are responsible for taking the architecture and making it work. The senior engineer must solve the hard problems and ultimately be responsible for the project.

If you're building your skills toward the expertise of a senior engineer, be careful not to oversell yourself in the short term, and take time to gain hands-on experience with these cutting-edge technologies.

billb@objectfocus.com

billp@objectfocus.com

AUTHOR BIOS

Bill Baloglu is a principal at Object Focus (www.ObjectFocus.com), a Java staffing firm in Silicon Valley. Prior to that, he was a software engineer for 16 years. Bill has extensive OO experience and has held software development and senior technical management positions at several Silicon Valley firms.

Billy Palmieri is a seasoned staffing industry executive and a principal of ObjectFocus. Prior to that, he was at Renaissance Worldwide, a multimillion-dollar, global IT consulting firm, where he held several senior management positions in the firm's Silicon Valley operations.

WRITTEN BY STEVE BENFIELD

Web Services: XML's Killer App

Here a service, there a service, everywhere a service service

My hype meter has been revved up lately, and what has pegged it is Web services. Who is hyping up Web services? Hmm…Microsoft, Sun, IBM, HP, BEA, SilverStream, Ariba, BowStreet, webMethods…my aunt Judy. I'm expecting to see this e-mail soon: "Quit your job and make $100,000 a year writing Web services in this groundbreaking business opportunity." Oh…that one might be true <G>.

Okay, so what's behind all this hype? Is all this real?

My take: absolutely real – or at least it will be very soon. This is my fourth "sea change" in software development. I can recognize a good thing when I see it.

In the '80s, the PC computing revolution took off. *Snicker, snicker, PCs will never be serious.* This revolution networked individual PCs but didn't change how corporate IT was implemented. That evolved from file system–based databases (dBase anyone?) to client/server applications using relational databases – that did change how IT was implemented. *Snicker, snicker, client/server is for departmental apps.* The Internet revolution has networked practically every system in the world and has led to some changes in corporate IT. *Snicker, snicker, great for brochureware and shopping.* We're now evolving to allowing applications to be easily built through networked businesses. Do I hear a snicker from the back row?

Web services is a subset of a service-oriented architecture. Each piece of business functionality is encapsulated into some sort of object/component that is then made available as a stand-alone service. Pretty basic stuff, really. What turns a service into a Web service is that it uses XML-in and XML-out for parameter and return value passing and that the service is accessible through a standard wrapper. It also has some easy-to-use protocol such as HTTP or SMTP. So you send XML to a URL. Magic happens. XML gets returned. Pretty simple.

What differentiates this from a servlet wrapper to a session bean? The servlet parses the XML, calls the bean, gets return data, remarshals the return XML, and sends the string back. Well, at the core, not much. However, a proper Web service has a SOAP wrapper along with a description written in WSDL. If the service is public, it will be registered with some registry, most likely using UDDI. The registry might be private to your organization, public to the world, or available in an industry-specific registry. The services you create will be made available to others who need them.

The cool thing about having a standard such as SOAP and WSDL that supersedes any specific technology standard (EJB, DCOM) is that an application can be built without regard to the underlying technology implementation. Service-based architectures allow you to separate the business use of a function from its actual technical implementation. Creating a business process that includes DCOM, EJB, CICS, and AS/400 functionality becomes a snap if each function is packaged as a Web service.

This alone is useful, but if you're a pure-Java shop, what's the real benefit? You can wrapper all of your functionality with EJBs and just call them. Of course, we know things aren't that simple because of the proliferation of technology in many companies. But let's take this to the next level: assume each business process you want to call exists in different businesses. This means you're building an application that spans your internal systems and interacts with your business partners. As a developer, you'd look up the various services in the appropriate registries. From the registry you'll get a location and documentation you need to call the service. (A phone call might still be required for authorization, etc. Don't be fooled into thinking everyone will just use complex services without humans getting involved.) Your application would call each service (remember, a service is basically a URL with XML-in and XML-out). Your job is to integrate the pieces and spindle, fold, and mutilate the XML passed between the services. You're creating workflow and processing rules – a business process. Odds are you'll then publish the new object you've created as its own service. This isn't too different from normal OO except services are much more coarse-grained and don't require the same language to interoperate.

Service-oriented architectures and Web services look to fulfill the promise of application assembly. Discrete business functionality deployed as services is assembled into applications. To me this implies heavier reliance on workflow, rules engines, and messaging systems such as JMS. If we're manipulating XML at each step of the process, then a higher-level, more productive way of building applications can be achieved. However, at the core, each service still has to be written, and "real" work must be accomplished. This is where J2EE comes in and why I think J2EE is the perfect platform for a robust Web services architecture.

So keep your eyes open. Begin reading about Web services. It *is* the next big thing and it's already happening. It's a steamroller that will change how we write software. But at the core it's not a revolution – it's an *evolution*. And, luckily, J2EE puts us in the perfect position to take advantage of it.

steve@stevebenfield.com

AUTHOR BIO

Steve Benfield is director of e-business strategy at SilverStream Software. SilverStream provides a complete and easy-to-use service-oriented development, assembly, and deployment environment running on standard J2EE servers.