CROSS-DATABASE PORTABILITY WITH JDBC

JAVA™ DEVELOPER'S JOURNAL

The World's Leading Java Resource

Volume: 5 Issue: 1, January 2000

SYS-CON PUBLICATIONS

JAVA DEVELOPER'S JOURNAL

EDITORIAL ADVISORY BOARD
TED COOMBS, BILL DUNLAP, DAVID GEE, MICHEL GERIN,
ARTHUR VAN HOFF, JOHN OLSON, GEORGE PAOLINI,
KIM POLESE, SEAN RHODY, RICK ROSS,
AJIT SAGAR, RICHARD SOLEY, ALAN WILLIAMSON

EDITOR-IN-CHIEF: SEAN RHODY
EXECUTIVE EDITOR: M'LOU PINKHAM
ART DIRECTOR: ALEX BOTERO
PRODUCTION EDITOR: CHERYL VAN SISE
ASSOCIATE EDITOR: NANCY VALENTINE
EDITORIAL CONSULTANT: SCOTT DAVISON
TECHNICAL EDITOR: BAHADIR KARUV
PRODUCT REVIEW EDITOR: ED ZEBROWSKI
INDUSTRY NEWS EDITOR: ALAN WILLIAMSON
E-COMMERCE EDITOR: AJIT SAGAR

WRITERS IN THIS ISSUE
RUSLAN BELKIN, GENE CALLAHAN, ROB DODSON, S. ALAN EZUST,
SCOTT GRANT, JIM MILBERY, TIM MOYLE, PATRICK SEAN NEVILLE,
JOHN OLSON, GEORGE PAOLINI, SIMON PHIPPS,
VISWANATH RAMACHANDRAN, SEAN RHODY, AJIT SAGAR,
TODD SCALLAN, SESH VENUGOPAL, JASON WESTRA, ALAN WILLIAMSON

SUBSCRIPTIONS
FOR SUBSCRIPTIONS AND REQUESTS FOR BULK ORDERS,
PLEASE SEND YOUR LETTERS TO SUBSCRIPTION DEPARTMENT
SUBSCRIPTION HOTLINE: 800 513-7111
COVER PRICE: $4.99/ISSUE
DOMESTIC: $49/YR. (12 ISSUES) CANADA/MEXICO: $69/YR.
OVERSEAS: BASIC SUBSCRIPTION PRICE PLUS AIRMAIL POSTAGE
(U.S. BANKS OR MONEY ORDERS). BACK ISSUES: $12 EACH

PUBLISHER, PRESIDENT AND CEO: FUAT A. KIRCAALI
VICE PRESIDENT, PRODUCTION: JIM MORGAN
VICE PRESIDENT, MARKETING: CARMEN GONZALEZ
CHIEF FINANCIAL OFFICER: IGNACIO ARELLANO
ACCOUNTING MANAGER: ELI HOROWITZ
CIRCULATION MANAGER: MARY ANN MCBRIDE
ADVERTISING ACCOUNT MANAGERS: ROBYN FORMA
MEGAN RING
JDJSTORE.COM: JACLYN REDMOND
ADVERTISING ASSISTANT: CHRISTINE RUSSELL
GRAPHIC DESIGN INTERN: AARATHI VENKATARAMAN
WEBMASTER: ROBERT DIAMOND
WEB EDITOR: BARD DEMA
WEB SERVICES CONSULTANT: BRUNO Y. DECAUDIN
WEB SERVICES INTERN: DIGANT B. DAVE
CUSTOMER SERVICE: SIAN O'GORMAN
ANN MARIE MILILLO
ONLINE CUSTOMER SERVICE: AMANDA MOSKOWITZ

EDITORIAL OFFICES
SYS-CON PUBLICATIONS, INC.
39 E. CENTRAL AVE., PEARL RIVER, NY 10965
TELEPHONE: 914 735-7300 FAX: 914 735-6547
SUBSCRIBE@SYS-CON.COM

JAVA DEVELOPER'S JOURNAL (ISSN#1087-6944)
is published monthly (12 times a year) for $49.00 by
SYS-CON Publications, Inc., 39 E. Central Ave., Pearl River, NY 10965-2306.
Periodicals Postage rates are paid at
Pearl River, NY 10965 and additional mailing offices.
POSTMASTER: Send address changes to:
JAVA DEVELOPER'S JOURNAL, SYS-CON Publications, Inc.,
39 E. Central Ave., Pearl River, NY 10965-2306.

WORLDWIDE DISTRIBUTION BY
CURTIS CIRCULATION COMPANY
739 RIVER ROAD, NEW MILFORD NJ 07646-3048 PHONE: 201 634-7400

BPA INTERNATIONAL
SYS-CON PUBLICATIONS

SEAN RHODY, EDITOR-IN-CHIEF

Jack, Be Nimble

Are you nimble enough? That seems to be the new buzzword in the Internet world. Nimble. Nimbleness. Nimbler. My development team is nimbler than yours. Being nimble is the name of the game today. It's not enough to be good developers, we've got to be quick developers.

What's driving this? What's wrong with the way we've been doing business? I can name that tune in two words – the Internet. The sensationalism surrounding each new dot.com IPO has created a market in which the business plan is to be first at any cost. Many of the companies going public nowadays have a business plan that calls for them to lose money for several years in order to gain market share.

I'm not a strategist, but some of my colleagues who are assure me that there are precedents for this that are too important to ignore. It's almost always the first person in a market that controls the market share. Sometimes there's a significant financial barrier to switching. But often it may simply be inertia – people tend to stay with what they know. Better the devil you know than the devil you don't.

Another contributor is the knowledge that in six months someone can completely rearrange a market with an innovative site. It was pointed out to me that 18 months ago, if you'd walked in to a hotel, or the office of an airline, and told them that today they would be selling their surplus to customers at a price the customer set, they'd have laughed in your face, if not tossed you out into the street and called the guys with the white coats. Today priceline.com is quietly laughing. All right, maybe gloating is a better word. "I told you priceline would be big, really big!" Beam me up, Scotty.

So what we have is an environment in which market share matters more than profitability, and one in which six months is an eternity. Web time. We're all now on Web time.

And that leads us to the "nimble" question. It's one thing to go in and do business analysis for several months prior to starting development with a company that already has a going business and looks at the Net as a way to either grow that business or fend off online competitors. Companies like Merrill Lynch could afford to hold back and get their sites down pat. They had the business, although the DLJs of the world were hurting them.

But to companies that have no business, every day is an eternity. You can add up the opportunity cost, then toss in the development costs, and begin to understand the pained look in the eyes of every prerelease CEO. The Internet has removed a number of barriers to entry in the marketplace, and anyone can build a site in their garage that can impact the world. Time is not of the essence. It is the essence.

And that makes development really interesting. Just like that old Chinese curse, "May you live in interesting times." Welcome to "interesting," population unknown.

So how do we accomplish "nimbleness"…or is it "nimbility"? I've got a set of answers that I think works, but it comes down to the same old answer to the question "How do you get to Carnegie Hall?" – practice, practice, practice.

One of the first keys to nimbledom, to my mind at least, is having a team that's worked together before. Not necessarily everybody; there's room for newcomers on every project. But a core team that's jelled as a unit is a real jump start.

Another key to being nimble is familiarity with the tools, languages and paradigms used to do development. Use what you know; know what you use. When developers know the tools, they can focus on the project rather than on the environment.

Add tools that are complementary, not competitive. Look at your current toolset when selecting a new product. For example, several commercial personalization engines are available, some Java-based, some not. It's a lot more difficult to integrate the ones that aren't – sometimes speed-to-market should drive the technology decision, not marketing hype.

Develop, or buy, frameworks to increase productivity – then learn them. My team has a framework that we spent a great deal of time perfecting, but it saves us months of development time every time we hit a new project. That's because we know how we're going to build it, and at least a quarter of the project code is already built because of the framework.

The value we can bring to the business world by achieving nimbleness is almost incalculable, but the true essence of nimbledom is speed. Web time waits for no one. In the time it took me to write this article and get it to press, someone nimble developed and built a new e-commerce site. And someone else is still in the planning stages.

Good luck, and stay nimble.

sean@sys-con.com

AUTHOR BIO
Sean Rhody is the editor-in-chief of Java Developer's Journal. *He is also a principal consultant with Computer Sciences Corporation where he specilaizes in application architecture – particularly distributed systems.*

JOHN OLSON, PRESIDENT OF DEVELOPOWER, INC.

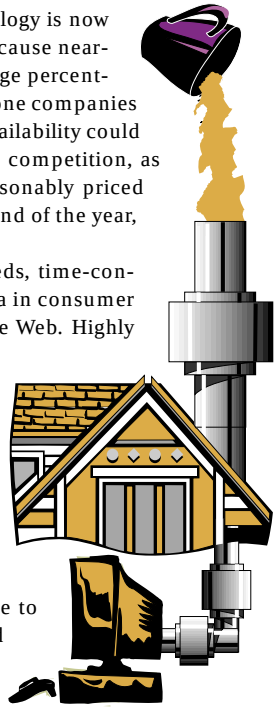
Fat Pipes Boost Java Flow

At work, Lisa takes advantage of her employer's dedicated T-3 to quickly access live NASDAQ quotes via a Java applet stock ticker. She spends each day alternating between development work and day trading. Soon to be rich, she dreams of early retirement and a life filled with leisure activities and day trading at home. Back at home her lack of patience and 56K connection make day trading a painfully slow torture. Java and 56K aren't meant to be together.

Unfortunately, most of the 50 million regular Internet users worldwide are ecstatic if they can even achieve 56K. High-speed connectivity is common to businesses but is out of financial reach of the average home-based Internet user. To be successful, consumer-targeted Web sites have to provide content quickly, because consumers won't wait. So while businesses enjoy IP-delivered, feature-rich Java applications and applets, the general public is relegated to HTML and JavaScript Web sites. For Internet development, narrow throughput has resulted in Java's being used only for specialty applets and Web sites that target businesses.

That is soon to change.

The final quarter of 1999 was filled with initial public offerings for Internet infrastructure companies frantically trying to raise huge quantities of money. These companies are scrambling to bring high-speed Internet services to home-based consumers before their competitors do. Tens of billions of dollars are being raised and spent to take advantage of and enhance existing cabling as well as solve "local loop" bottlenecks by laying down high-speed connections from national data backbones to local switches and even to the end users. Medium and small-sized businesses, and even homes, will have fiber optic connections. The frenzy is spreading to Europe, Asia and Latin America. While AT&T and several competing cable companies intend to deliver cable modem service to more than 10 million Americans by the end of 2000, numerous other companies are using wireless technologies to win the business of those same consumers. Though a latecomer, digital subscriber line (DSL) technology is now growing at nearly the same subscription rate as cable modems because nearly every home in North America is wired for telephone while a large percentage aren't wired for cable. A recent federal law forcing regional phone companies to lease existing phone lines to other DSL providers means DSL availability could soon exceed that of cable modem services. The fierceness of the competition, as well as the vast size of the yet untapped market, mean that reasonably priced high-speed data access should be available to your home by the end of the year, if it isn't already.



With the fatter pipes and much higher consumer access speeds, time-consuming downloads will no longer be a limiting factor to using Java in consumer Web sites. I believe we'll see an incredible transformation of the Web. Highly interactive, rich and full-featured sites will become the norm. Java use will explode.

Sure, the problem remains that not all Internet users will have high-speed Internet access. As with many Web sites that exist today, Web developers will provide both high-featured Java and low-feature HTML/JavaScript versions of their sites. However, affordable ISP costs will result in high-speed access becoming the standard in North America, even for home-based users.

A few years ago it looked as though Java was going to take over the world – and fast. But performance problems, particularly due to download times for IP access, made Java use infeasible for broad Internet use. Now it looks like that barrier is being removed and Java will finally become the standard language for Internet development.

john@sys-con.com

AUTHOR BIO

John Olson is president of Developower, Inc., a software consulting firm specializing in multitier solutions. A member of JDJ's Editorial Board, John is also editor-in-chief of PowerBuilder Developer's Journal.

A fascinating, powerful technology for creating portable messaging code within your applications

USING THE JAVA MESSAGE SERVICE WITH BEA/WEBLOGIC

WRITTEN BY SCOTT GRANT

In the last couple of years Sun has introduced a number of APIs targeted toward enterprise application development. One of the most exciting of these is the Java Message Service, or JMS. The JMS API is designed to do for messaging in the enterprise what JNDI does for naming and directory services and JDBC does for database access. JMS is an API that's designed to provide a common facility for enterprise messaging, leaving the underlying implementation of the messaging to whatever application server or other enterprise messaging software technology you wish to use. This is an exciting advance for those involved with the creation or use of message-oriented middleware (MOM) – and especially for Java developers who need to utilize such facilities within their own products. With JMS you should be able to write one set of code for messaging against the JMS API and then use it across any messaging system provider that offers JMS support.

In this article I'll show you how to create a series of message producers and consumers that utilize most of the functions of the JMS API – using persistent topics and queues, creating temporary destinations, filtering via message selectors, and so on. I'll implement these examples against one of the first vendor implementations of JMS, the one provided with the WebLogic application server from BEA (details on this server and configuring JMS are available online in the file "install.txt" at JavaDevelopersJournal.com). In this process I'll take you through the necessary steps to implement these applications, including the modifications to the application server to support the example code you'll create.

Before you continue with the example code here, I recommend that you first obtain and install the latest release of the BEA/WebLogic application server. You can download a free 30-day trial copy from BEA at www.weblogic.com.

The Code

Included online are four source files that demonstrate the use of the two types of messaging in JMS – publish/subscribe and point-to-point. Sender.java and Receiver.java demonstrate the use of Queues within JMS for point-to-point communication, and Publisher.java and Subscriber.java demonstrate the use of Topics for publish/subscribe communication. A "readme.txt" file, also online, contains the instructions for running the various applications and any last-minute information about the code. Before you attempt to run these applications, make sure you've read the "install.txt" file and made the necessary modifications to the WebLogic server as indicated. As the sample code also contains comments and further JMS details that can't be completely covered in this article, the reader is encouraged to examine the source code and run the applications.

The Java Message Service

The Java Message Service was designed as a set of interfaces that would be implemented by MOM providers and various other vendors that wish to support messaging (application server providers, database server providers, etc.). These interfaces provide a common API for client applications that wish to implement code that uses messaging against potentially any given number of underlying messaging systems.

JMS was also designed to be flexible enough to provide this wide support for existing messaging systems while remaining portable. As a result, it doesn't support every possible messaging option from every messaging system. While you may be familiar with any given MOM product, it's not automatic that JMS will support every aspect of that product.

The primary concept in JMS is that of Destinations. A *Destination* is nothing more than an association for message producers and message consumers. Destinations break down into two types, Topics and Queues. Separate interfaces defined for each Destination type allow a provider to support one or both, depending on their messaging facility. A Topic is designed for publish/subscribe messaging, and a Queue is designed for client-to-client (point-to-point) messaging. Both Destination types support persistence. JMS also provides support for transactions. Transactions enable you to support commits or rollbacks on your messaging operations. Thus, if you have a failure within the boundaries of your transaction, you can roll back all of the messaging operations that took place within the boundaries of that transaction. Likewise, if everything completes successfully, you can perform a commit on your messaging operations to make them durable. As JMS Transactions are beyond the scope of this article, I leave you to investigate that aspect of JMS on your own.

Initializing JMS

CONNECTION FACTORIES

To initialize JMS you use a ConnectionFactory – a tagging interface, one with no methods, that is extended via either TopicConnectionFactory or QueueConnectionFactory. A provider implements one or both of these factory interfaces to provide access to their specific messaging service implementation. WebLogic provides generic implementations of each of the Factory types. The Factories, as well as Topics and Queues, are considered "administered" objects and as such are created by the WebLogic server upon start-up (in our case we're using WebLogic's JMS implementation). These administered objects are then retrievable via JNDI. You use a JNDI naming context to retrieve the administered objects. Don't worry if this seems a little confusing – the code examples will make it easy to understand. You may also create your own Factory objects in WebLogic instead of using the default Factories; this is done within the weblogic.properties file. User-defined Factories are discussed later, in the section on Topics, while the details of defining your own Factories are contained in the "install.txt" file included online with the article source code.

The following code from within the Sender.java example shows how you initialize JMS and obtain the default ConnectionFactory for Queues from JNDI:

```
public final String JMS_FACTORY = "javax.jms.QueueConnectionFactory";

...

queueFx = (QueueConnectionFactory) initCtx.lookup(JMS_FACTORY);
```

The name of the default ConnectionFactory for Queues is passed in to the lookup() method of the initial JNDI naming context. A reference to an implementation of QueueConnectionFactory is then returned to us from the WebLogic application server (the getInitialContext() method in the Sender.java sample code that shows the details of initializing JNDI and obtaining the initial naming context from WebLogic – you can obtain details on the API from Sun's Web site at: http://java.sun.com/products/jndi/1.2/javadoc/index.html).

Because ConnectionFactories are administered objects, the WebLogic application server takes care of creating default ConnectionFactories for Queues and Topics, as well as any user-defined Factories, and binding them to names in the WebLogic JNDI implementation for lookup by client code. Thus the name in JNDI for the default ConnectionFactory for Queues is "javax.jms.QueueConnectionFactory" and a lookup of this name through JNDI will return a reference to a default implementation of this interface, provided by WebLogic. There is a default ConnectionFactory for Topics as well, and it follows the same format; "javax.jms.TopicConnectionFactory". A lookup through JNDI will return you a reference to a default implementation of TopicConnectionFactory, also provided by WebLogic.

CONNECTIONS

After successful creation of the appropriate ConnectionFactory, the next step is to create a *Connection*, an interface defined by JMS that represents a client connection to the underlying messaging provider. The ConnectionFactory is responsible for returning a Connection implementation that can communicate with the underlying messaging system. Clients will typically use a single connection. According to the JMS documentation, the purpose of a Connection is to "encapsulate an open connection with a JMS provider" and to represent "an open TCP/IP socket between a client and a provider service daemon." The documentation also states that the Connection is where client authentication should take place, and that clients can specify unique identifiers, among other things. Like ConnectionFactories, Connections come in two flavors, depending on the Destination type you're working with: QueueConnection and TopicConnection both extend the base Connection interface and you'll typically work with only one or the other, depending on the kind of messaging your client is using. The creation of a Connection is done via the ConnectionFactory. Connection contains two important methods, *start* and *stop*, that start and stop the sending and receiving of messages on the connection. See the full code block in Listing 1.

SESSIONS

Once you've obtained a Connection from the ConnectionFactory, you must create one or more sessions from the Connection. Session is the base interface, and again, there are two Destination-specific interfaces that extend it: QueueSession and TopicSession, which provide Queue- or Topic-specific Session methods. A Session is a type of factory that produces message consumers or producers for its Destination type (if it's a QueueSession, it creates Queue-oriented consumers and producers; if it's a TopicSession, it creates Topic-oriented producers and consumers). A Session may be transacted or not, and you typically set this by passing a boolean parameter to the appropriate creation method on the Connection. You also typically pass a parameter to the Connection's Session creation method that sets the message acknowledgment mode of the Session being created – this mode specifies whether the client or the consumer will acknowledge any messages it receives (this parameter will be ignored if transactioning is being used). Other methods provided by Session are various message creation methods that let you create JMS messages of specific types containing text, bytes, properties or even serialized Java objects (more on the Message interface below). See Figure 1 for a diagram of the actual JMS interface inheritance and creation relationships – see also the code in Listing 1.

DESTINATIONS

Before you can create any message producers or consumers, you must have a specific Destination with which to associate any producers or consumers. Remember, Destinations are administered objects, like ConnectionFactories. This means that a Destination is maintained by WebLogic and you must retrieve it through a JNDI lookup. This also means that in this case the Destination must be defined in advance. This is not to say that Destinations must always be created in advance. The JMS API provides the ability to create temporary Destinations that are available only for the life of the Session that created it as well as permanent Destinations through the JMS API at runtime. However, in the current WebLogic implementation of

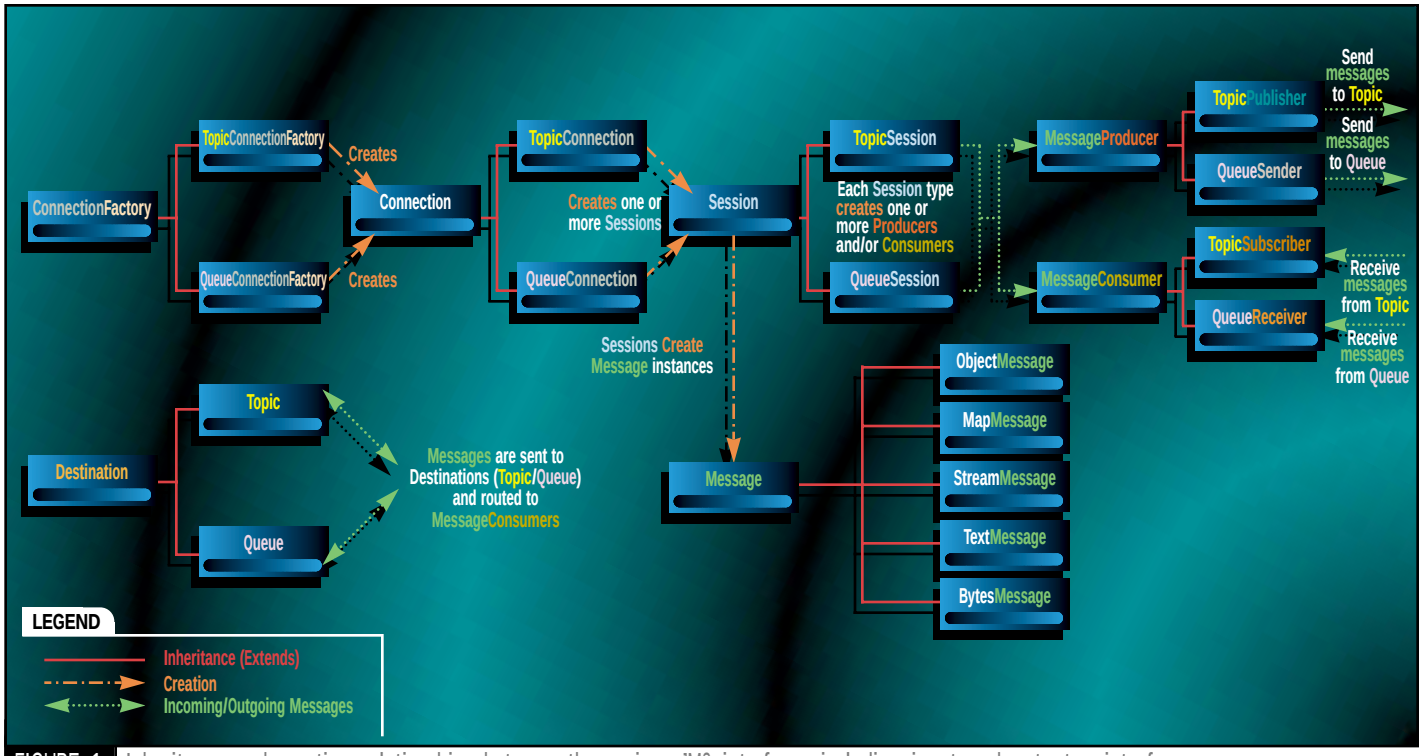


FIGURE 1 Inheritance and creation relationships between the various JMS interfaces, including input and output points for messages

JMS, it should be noted that if you create Destinations via a Session, Destination will exist only as long as the WebLogic server is running. If the server crashes or is brought down, the Destination will be gone. The only way to create a truly persistent Destination is to create it within the weblogic.properties file (see the "install.txt" file for details on how to do this). For the purposes of this article, our Destinations are created in advance, through the weblogic.properties file. Destinations defined in this file will be created by the WebLogic application server when it starts up, and will be made available to your client code through JNDI. Listing 1 shows the code from the Sender.java file, which demonstrates how to create a QueueConnection and a QueueSession, and retrieve our own Destination – a Queue we defined in weblogic.properties named "jdj.article.queue.sender" – the same as the package hierarchy for the Sender application.

Message Consumers and Message Producers

The final stage of the JMS initialization process is the creation of MessageConsumers and MessageProducers. Like ConnectionFactory, Connection and Session, these are base interfaces that have Destination-specific interfaces that extend them for use with either Topics or Queues (I use the terms *Consumer* and *Producer* interchangeably with *MessageProducer* and *MessageConsumer*). MessageConsumers are used to receive messages sent to a Destination, and MessageProducers are used to send messages to a Destination. Both are created by a Session instance. MessageProducer is extended by Destination-specific interfaces called *QueueSender* and *TopicPublisher*. MessageConsumer is extended by the interfaces *QueueReceiver* and *TopicSubscriber*.

Once you've created your MessageConsumer and/or MessageProducer, you have everything in place to begin receiving and/or sending messages. Since the creation of consumers and producers is specific to either Queues or Topics, I'll discuss the process for both types in the relevant sections below, specific to the Destination type.

Messages

Before I take the final step and begin to delve into the specifics of sending and receiving from Topics and Queues, we need to discuss one other interface, Message, which represents the JMS Message itself. This is the object that contains the information to be sent to a Destination and received by consumers that are listening on that Destination. Messages are created by a Session instance, and are composed of three sections: a header, properties and a body. The header is used to identify and route the message and the client developer typically doesn't see or deal with header information. Properties support application-specific values passed with a message. These property fields are predefined; a full description of them is available in the JMS Documentation. I use a few of these properties within the sample code. The body section of the message is the actual "payload" of the message and can be of five types represented by five specific interfaces: StreamMessage, MapMessage, ObjectMessage, TextMessage and BytesMessage (see Figure 1).

Persistence

JMS has an additional important aspect worth discussing: it supports persistent delivery of messages. Quite simply, this means that if a Consumer isn't running or is unavailable when a message is sent to a Destination, the message will be held until the next time that Consumer connects to the Destination. If you have five applications listening on a single Topic, for instance, and one of them should crash, the next time that application starts and connects to that specific Topic, any messages sent to that Topic while the application was down will be delivered to the application when it begins listening again. If this seems difficult to follow, it'll make more sense when you run the sample code.

Queues

A Queue is designed for "point-to-point" or "one-to-one" message delivery. This means that a Queue should have only one client sending messages to it, and only one application consuming messages from that Queue. In reality, you can have multiple clients sending to a Queue, but you should never have more than one application consuming messages from that Queue. If you have more than one Consumer on a Queue, there's no guarantee about which Consumer will receive the message, but it'll only be one. If you need multiple Consumers on a Destination and would like all of them to receive the same message, you should use a Topic (see the following page).

The Sender.java and Receiver.java files contain the code that shows how to use a Queue. The code demonstrates the initialization process for JMS and retrieval of our predefined Queue, and how to create both a MessageProducer and a MessageConsumer for sending and receiving messages on the Queue.

There are two specific interfaces for consuming and producing messages on a Queue. *QueueSender*, which is returned from a QueueSession via a call to one of the createQueueSender() methods of the QueueSession, is used to send messages to the Queue. *QueueReceiver*, which is returned from a QueueSession via a call to one of the createQueueReceiver() methods of the QueueSession, is used to receive messages from the Queue.

Listing 2 is the code from the sendMsg method of Sender.java that shows how you create a MessageProducer from your Session, then construct a TextMessage and send it. In this code we're creating a QueueSender, then creating a TextMessage with text retrieved from a TextField in the UI of the application. We then use the QueueSender method "send" to deliver the message.

There are two ways to handle the receipt of incoming messages on a MessageConsumer – synchronous and asynchronous. The first step is to create the MessageConsumer itself; the next is to decide whether you want synchronous or asynchronous message delivery on the Consumer. After you create a MessageConsumer from your Session, if you want to synchronously receive the next message produced for the Destination, you can call the receive method on your MessageConsumer – this method takes no parameters and will block until the next Message is received, which it will return to the caller. For asynchronous message receipt, you register an implementation of the MessageListener interface with your MessageConsumer. This goes for both Topics and Queues. MessageListener has one method that you must implement, onMessage, which receives one parameter, a Message. This method will be called on your implementation of onMessage for every message delivered to the Destination. This is demonstrated in the implementation of onMessage in both Sender.java and Receiver.java. In Receiver.java we have the following code in the initializeJMS method, which creates the MessageConsumer (a QueueReceiver) and sets the implementation of MessageListener:

```
// Create a Receiver for the Queue...
receiver = session.createReceiver(queue);

// Set the listener (this class)
receiver.setMessageListener(this);
```

Once the Connection's "start" method is called, messages will begin to come in to the Consumer as they arrive at the Destination.

REPLYTO — USING TEMPORARY QUEUES

You'll also notice that both Sender.java and Receiver.java implement MessageConsumers and MessageProducers. Each implements MessageListener. This demonstrates one of the interesting features of JMS, that is, the use of temporary Destinations. An application that wishes to receive responses to the messages it produces can create a temporary Queue or Topic and pass this Destination in the Message it sends. One of the properties of Message is the JMSReplyTo property. This property is intended to be used for this purpose. You can create a temporary Queue or Topic and place it into the JMSReplyTo property of the Message. Consumers who receive this message have a private, temporary Destination that they can use to respond to the sender. This is demonstrated by two methods, one in each of the files. In Sender.java we have the following sections of code, which create the temporary Queue and place it in the JMSReplyTo property of the TextMessage:

```
// Create a temporary queue for replies...
tempQueue = (Queue) session.createTemporaryQueue();
```

The line of code above is found in the initializeJMS method of Sender.java. This code creates a temporary Queue on application start-up that will exist for the lifetime of the application. The next line of code, found in the sendMsg method of Sender.java, shows how to set the JMSReplyTo property to contain the temporary Queue.

```
// Set ReplyTo to temporary queue...
msg.setJMSReplyTo(tempQueue);
```

When this message is received by the QueueReceiver of Receiver.java, the temporary Queue is extracted from the JMSReplyTo field, and a QueueSender is constructed by the application to send a response message back to Sender.java. This demonstrates how to use the properties of a JMS Message and shows the usefulness of a private, temporary Destination. It also demonstrates how clients can be both Producers and Consumers of messages. The following code from Receiver.java demonstrates how to handle the extraction of the temporary Queue from the JMS Message; this code is found in the onMessage method:

```
// Get the temporary queue from the JMSReplyTo
// property of the message...
tempQueue = (Queue) msg.getJMSReplyTo();
```

The following block from the sendReplyToMsg method shows how to create a QueueSender and send the reply:

```
// create a Sender for the temporary queue
if (sender == null)
    sender = session.createSender(tempQueue);

TextMessage msg = session.createTextMessage();
msg.setText(REPLYTO_TEXT);

...

// Send the message to the temporary queue...
sender.send(msg);
```

Topics

A Topic is designed to implement a "publish/subscribe" message delivery mechanism. Whereas Queues are designed to have one Producer and one Consumer, a Topic is designed to allow multiple Producers to send messages to it, and to have multiple Consumers receiving delivery of the same Message from the Topic. A Topic is a "many-to-many" model. The creation process for MessageProducers and MessageConsumers of a Topic is similar to that of a Queue. You use your Session to create the TopicPublishers and TopicSubscribers. These mirror the QueueSender and QueueReceiver in that each provides Topic-specific capabilities and implements the base MessageProducer and MessageConsumer interfaces.

The creation process for a TopicPublisher is nearly identical to that of the QueueSender. Following is the code from the sendMsg method of Publisher.java that shows the creation of a TopicPublisher and how to publish the message to the Topic:

```
// create a Publisher if there isn't one...
if (publisher == null)
   publisher = session.createPublisher(topic);

TextMessage msg = session.createTextMessage();
msg.setText(text);

...

// Publish it to the topic...
publisher.publish(msg);
```

DURABLE SUBSCRIBERS

One of the interesting aspects of TopicSubscribers is that it is a durable subscriber – a unique name provided by a client that identifies it within a Session. A Session, in turn, has a Client ID associated with it, defined either at runtime through a method call on the Connection or as part of the administered ConnectionFactory (in our case it's defined in this way within the weblogic.properties file). Thus a Connection provides a Session with a Client ID, and durable subscriber names are unique identifiers within Sessions, associated with a specific Client ID. The purpose of durable subscribers is to create unique, persistent Consumers for a given Topic.

An application that creates a TopicSubscriber via a call to the createDurableSubscriber method of a TopicSession must pass in a durable subscriber name (a string) as one of its parameters; for instance, you could set the durable subscriber name to the name of the user currently logged into the application, and so on. This name will uniquely identify a particular subscriber to a Topic (in conjunction with a unique Client ID for the Connection/Session). Once this durable subscriber has registered with the Topic, the Topic will persistently ensure delivery of messages to that subscriber. This means that if a particular durable subscriber is unavailable, messages will be held until the next time that durable subscriber (with the same unique, durable subscriber name and Client ID) registers as a Consumer. The Subscriber.java file demonstrates the creation of durable subscribers and allows you to use a default subscriber name or to set this identifier from the command line (see the "readme.txt" file for further details on running the application and demonstrating the message persistence with durable subscribers). The following code fragment from the initializeJMS method of Subscriber.java shows how to create a durable subscriber from your TopicSession:

```
subscriber = session.createDurableSubscriber(
                                              topic,
                                              subscriberID,
                                              SELECTOR,
                                              false);

// Set the listener (this class)
subscriber.setMessageListener(this);
```

TopicSubscribers that aren't created as durable subscribers won't receive messages persistently, and will receive them only while they're running. For further details on durable subscribers, and Topics in general, please see Sun's JMS documentation. One other item about the preceding code: note that the third parameter passed to the method is SELECTOR. This is where message selectors are associated with a Consumer (see below for details on message selectors).

FILTERING — USING MESSAGE SELECTORS

The final area of JMS I'll touch on is the message selector – a filter applied to MessageConsumers that can act on the properties and header sections of incoming messages (but not on the body of the message) and determine whether the message is actually to be consumed. Message selectors are strings, and are based on a syntax that is a subset of SQL-92, according to the JMS documentation. You can apply message selectors as part of the creation of a MessageConsumer. The way these selectors are applied to incoming messages differs slightly based on whether the MessageConsumer is a QueueReceiver or a TopicSubscriber. I encourage you to examine the syntax of message collectors, and how they can be applied, by examining the Subscriber.java file and running the Publisher and Subscriber applications. The following section of code defines our message selector in Subscriber.java, and the application itself lets you change the selector from a text field:

```
public final String SELECTOR = "JMSType = 'TOPIC_PUBLISHER'";
```

This selector examines the JMSType property of the incoming message from our Topic and determines whether the value is equal to TOPIC_PUBLISHER. If it is, the message is passed to our MessageListener implementation; if it isn't, the message is ignored. See the "readme.txt" file for details on running the applications and demonstrating this behavior. I also encourage you to check out Sun's JMS documentation.

Summary

JMS is a fascinating and powerful technology for creating portable messaging code within your applications. It allows for both "point-to-point" and "publish/subscribe" message delivery, and also supports transactions and persistence. The WebLogic application server from BEA offers a robust and complete implementation of JMS that works in conjunction with the other technology provided by the application server, such as EJBs and servlets. This creates interesting potential for persistent, asynchronous messaging with transactioning support between various enterprise objects and services. It is my hope that this article will encourage you to explore the online sample code in conjunction with the article text, and to examine the possibilities that WebLogic, and JMS in particular, offers.

AUTHOR BIO

Scott Grant, a director of development at Rapid Logic, Inc., and a Sun-certified Java developer, has 14 years of diversified engineering experience. He was the chief architect and lead developer of Rapid Logic's Rapid-Control for Applets product and currently leads the development of their enterprise technology products.

scottg2@home.com

Listing 1

```
public final String JMS_FACTORY =
"javax.jms.QueueConnectionFactory";
public final String QUEUE = "jdj.arti-
cle.queue.sender";
...
private void initializeJMS(boolean
transacted)
{
 try
   {
    if (initCtx != null)
    {
     // Look up the default QueueCon-
     // nectionFactory...
     queueFx = (QueueConnectionFactory)
            initCtx.lookup(JMS_FACTORY);

     // Create a QueueConnection from
     // the Connection Factory...
     conn = queueFx.createQueueConnec-
     tion();

     // Create a QueueSession from the
     // QueueConnection. The first
     // parameter is a boolean that
     // specifies transacted or not
     // transacted. The second param
     // specifies that our Session will
     // automatically Acknowledge a
     // client's receipt of a message.
     session = conn.createQueueSes-
     sion(transacted,
              Session.AUTO_ACKNOWLEDGE);

     // Look up the Destination we want
     // to use for our Consumers and
     // Producers for this session. In
     // this case a Queue called
     // "jdj.article.queue.sender".
     queue = (Queue) initCtx.lookup(QUEUE);

        ...
      }
   }
   ...
}
```

Listing 2

```
// create a Sender
if (sender == null)
    sender = session.createSender(queue);

TextMessage msg = session.createTextMessage();
msg.setText(text);

// Override default, to insure it's
// using persistent delivery...
msg.setJMSDeliveryMode(DeliveryMode.PER-
SISTENT);

// Set ReplyTo to temporary queue...
msg.setJMSReplyTo(tempQueue);

// Send the message...
sender.send(msg);
```

KL Group "Profiling w/ JProbe Profiles"

www.klgroup.com/performance

Money...The Root of All Evil

WRITTEN BY
ALAN WILLIAMSON

When all is said and done, I hope you're reading this column – the first in the new millennium – in familiar surroundings. With any luck, the prophets of doom around the Y2K problem have been proved wrong and the world didn't stop spinning suddenly in a haze of apocalyptic fireworks. If this is the case, I congratulate the human race for making it past another one of life's man-made hurdles.

That's not to say that all our systems have made it smoothly into the new age. Who knows what state the world's IT systems will be in? I'm thinking it's not going to be quite as bad as predicted. There'll be glitches, of course, some anomalies, but on the whole I think the risk to human life has been grossly exaggerated. That said, I'd like to take some time now for reflection.

Now that the world's clocks have rolled over to 2000, we've lost a species that flurried so well in the run-up, including 1999. I don't think it's a major loss, and I'm sure Mother Nature will make the balance sheets tally up. These parasites preyed on unsuspecting victims, latching onto their IT budgets and, like all good blood-sucking insects, bleeding their victim dry before moving onto the next one, with the previous body feeling dazed and dizzy trying to figure out exactly what happened.

Who am I talking about? The so-called Y2K companies that popped up, charging astronomical rates to tell the end client that they should prepare for the turn of the century. Not many of them could actually offer any more than the warning – a 40-page report detailing the joys the change from 1999 to 2000 would bring to your company. And boy, was that an expensive report to commission! These companies were generally one-man bands run very much like a cult (minus the sex, I assume). They roamed the land looking for unsuspecting, worried, stressed IT directors who, come January 1, 2000, would have to report to their board of directors on what steps they took to sidestep the Y2K issue. An easy kill, methinks.

I'm talking about some real cowboys here. I'm aware of many companies of this ilk – companies that don't have even one computing qualification among them. In fact, I know of one British company that made a fortune reselling the free computer disk from our government for £25. But, you say, they must have added extra value to the disk to warrant the price increase from £0 to £25. Well, yes, they did, so I must apologize for not telling you the whole story: they added the company's sticker – with logo prominently beaming. With printing costs like that, maybe we're all in the wrong game!

Nugget from Vienna

I've just come back from the Java Migration Conference in Vienna, where IBM was our host for four days. As with many of these conferences, the majority of it was the same old nonsense, but you don't go for that. You go for the odd golden nugget that pops its head up – and a number of nuggets did indeed show themselves. One such came from Big Blue themselves in the form of Oma Sewhdat. Oma is in charge of the Java certification program that IBM is spearheading. It was one of those talks that surprised me. I sat down, mentally preparing myself to shut down and sleep, with my eyes open, for the next hour. But what I heard caught my attention, and suddenly all my processes were up and running again.

Oma began by outlining the need for Java certification, and I must say I was impressed by the philosophy that went into this. IBM's biggest concern is the lack of skilled individuals that can actually do the job at hand. Oma talked of spiraling salaries and the problems employers face with retaining good staff. As regular readers of this column will recall, I've had a number of things to say on this issue myself, as we have to go through the process of hiring and are only too painfully aware of how hard it is to get good people.

But one of the things he said got me thinking. If the likes of IBM, Sun and Oracle are starting to think what we have known for years, then maybe it's time for the industry at large to wake up to themselves. We're heading for disaster if we continue on the trajectory we're on now. If salaries continue to rise out of control, with Java developers becoming elitist and unaffordable, where do you think we'll end up? That's an easy question to answer: Java simply won't be deployed. A much more flexible and affordable solution will evolve to take its place. I believe this is one of the reasons for the popularity of Visual Basic. It wasn't rocket science and not at all difficult. Thus, finding good skilled people for it wasn't a problem and, more important, wasn't expensive. Whatever we, as developers, may think of Visual Basic isn't the point of the argument. The point is that Visual Basic was a quick route of development for companies that needed solutions fast and cost effectively.

If we're not careful, Java will become too expensive for companies to consider seriously. We're seeing this at the moment. Anyone tried hiring EJB developers? Sign a blank check and pray that no one comes along and head-hunts him or her. This will hopefully settle as more and more developers come up to speed with the new technology, but as

far as I'm concerned, for the time being anyway, silly money is being paid for these developers.

I'm one of the ones who wants to see the quality of development go up. At the moment, we as an industry are producing second-rate code. The only industry that's producing cutting-edge development is the games industry. Why? Because in this world the end users simply won't tolerate slow gameplay. They're constantly asking for more and more, with the average gamer not in a position to upgrade his or her machine every time a new game hits the shelves. In the corporate world the same pressure simply doesn't exist. Now don't get me wrong. I'm not saying that everyone is producing second-rate code. Let's just sidestep that flood of hate e-mail. I'm talking as a whole, and the sad fact is that the number of poor developers outnumbers the real hot developers.

Why is this? Well, I'm sure there are many reasons. My personal view is that we have too many people thinking they're programmers when in actual fact they have no clue. We've had this debate before in this column, which spawned a wonderful flurry of traffic on the mailing list. Many of you shared this opinion, or at least some of it. But the reason I'm bringing it up again is the thoughts I gleaned from this conference in Vienna. In speaking to Oma and a number of other key speakers, I found that the general thinking was that the quality is poor and that to secure Java longevity something has to be done.

Maybe we need more education. Teach Java at university? On the face of it this sounds like a wonderful idea. In practice, however, it doesn't quite work. I don't rate "Academic Java" very high. I haven't seen one example of someone being taught Java at university and coming out being able to do a day's work. Alas, I can't take credit for the term *Academic Java*; it was used at the conference by a number of key speakers who were expressing their distrust of this latest training bandwagon. I think the problem is that in many instances we have the blind leading the blind. The lecturers don't have the slightest idea how Java is being used in industry, and therefore all the skills that should be taught are glossed over.

We have a number of developers here at n-ary who got their Java skills from academia. Even they say it was a waste of time, and the Java they did then comes nowhere near the real-world Java. So why the big discrepancy? I don't know the answer to that one. But I'm asking all around me about their thoughts and I'm learning lots about how others feel. Let me know what you feel.

Mailing List

Come and join our mailing list and discuss what you feel about the state of Java code as a whole. If you want to be part of the discussion, send an e-mail to listserv@listserv.n-ary.com with subscribe straight_talking-l in the body of the text. From there you'll get instructions on how to participate on the list. Thank you all for your continued posts, and I have to say that I thoroughly enjoy the variety of topics discussed.

Members of this mailing list were let in on something new we tried. You read the column, you even join in the discussions on the mailing list. Well, now you can listen to a daily radio show. Yes, the **Straight Talking** column has gone radio. I and my esteemed colleague and cohost, Keith Douglas, host a 10–15 minute show once a day based on the ramblings of the mailing list. It's a mix of Java talk, music and general banter. So if you're bored, come and listen to us. Check out http://radio.sys-con.com/ for more details. Again, let me know what you think. I'd love to hear your thoughts.

Salute of the Month

This month's salute goes to one of the main men at the heart of producing the Java Virtual Machine for the AS/400. This man was introduced to us at the Vienna conference and he bears a spooky resemblance to one Jim Driscoll from Sun Microsystems. (Jim, if you're reading this, I think I've found your double!) Blair Wyman, the gentleman in question, gave an absolutely excellent speech that not only entertained but also educated us in the ways of the JNI interface. Blair describes himself as one of the backroom boys who doesn't get out that often. I think this is truly a

JAVA MIGRATION CONFERENCE SHOW REPORT

BM Vienna played host to this year's Java Migration Conference. The conference was packed with wall-to-wall talks aimed at the developer looking toward Java or the manager looking to move his or her department or systems to Java.

Fortunately for the delegates, IBM did little product pushing and concentrated mostly on the overall benefits a Java solution has to offer. That's not to say there was no discussion on WebSphere and San Francisco, but on the whole this was limited to simple infomercials. It wasn't a trade show. You were there to listen to speakers talk on a vast array of topics ranging from basic Java language skills right through to deploying a general EJB application.

The conference focused primarily on selling Java for the server side. Thus technologies such as Servlets and Enterprise JavaBeans featured strongly in the majority of presentations. Presentations from some of the smaller companies and vendors came as one of the refreshing changes. Many of these representatives weren't afraid to tell it like it is, with lots of interesting hints and warnings regarding some of the more popular development tools.

The conference opened with a look at certification and the inroads the major players are making to raise the standard of Java. Oma Sewhdat, who heads up IBM's certification program, discussed the logistics of acquiring the necessary qualifications. He took us through some of the early backroom politics to the current stage in which a coherent and logical training program is now in place. According to Sewhdat, once everyone was around the table, discussions went smoothly and all the companies wanted to see developers with greater skills. However, things got ugly when it came to naming the beast and who's branding would go where. We were told that this intercompany squabbling is ongoing, with a name for the new wave of training still pending. However, it's exciting and extremely worthwhile for both employees and employers. So watch this space.

A number of video conferences took place. Rational's Grady Booch, for instance, took us through the joys of designing for a large-scale application system. It was a good overview – one of those presentations that left the delegates asking questions as opposed to answering. Grady managed to convey enough information to allow us to think that little bit further into the future when we design our Web-based applications. Another excellent

overview of the whole Java arena came from Rick Ross of Activated. Rick is also president of the Java Lobby, and as usual conveyed exactly what many of us feel about the politics of our Java world. One of the more controversial points he raised was the ownership of Java. Rick pointed to an alliance between IBM and Sun rather than just allowing Sun full control. This is an interesting view and one that I'm sure was discussed over many a dinner table the following evening.

Speaking of dinner tables, our conference organizer, Susan Brunner, had a nice surprise for the delegates one night. Susan had arranged for one of Austria's finest wine producers, Willi Opitz, to come and treat us to a wine-tasting ceremony. He brought a range of his products for us all to swirl and spit. It has to be said that not all of us spat it out! Willi took us through a number of hints and tips on what to look for in a good wine and it was a refreshing break from the world of Java. The night ended with each delegate picking up a free bottle of wine after enjoying a meal at the revolving restaurant in the Danube tower. Willi was kind enough to grant **SYS-CON Radio** an interview and we managed to get some of his tips for us all to enjoy.

As is true of all conferences, some of the speakers are really boring. I'm sure you've all sat through a presentation at some point where the very will to live slowly seeps away. The names of these people shall remain anonymous. A name that won't, however, is that of one Blair Wyman from IBM Rochester. Blair was one of the main developers behind porting the virtual machine to the AS/400. Blair had three sessions, each one more entertaining than the one before. His presentations were not only informative and insightful but bloody damn funny to boot. The virtual machine is one piece of software we all take for granted, but Blair took us through some of the headaches he's had to face in order for us to rest safely in our beds.

SYS-CON Radio was there in full force with complete audio coverage. I and my esteemed **Straight Talking Radio** half, Keith Douglas, interviewed many of the conference speakers, including a number for our own **Straight Talking** shows. For full details be sure to visit the Web site at http://radio.sys-con.com/.

The conference was held at IBM's main office in Vienna, which provided an excellent venue for networking and making new friends. There was a rich tapestry of backgrounds, and the opportunity to mingle and introduce yourself presented itself many times. This conference had much to offer, assuming you were bold enough to ask the questions you needed answering.

Big Blue excelled here, and definitely came across as the developer's friend. We look forward to the next one.

Author Bio

Alan Williamson is CEO of n-ary (consulting) Ltd, the first pure Java company in the United Kingdom. The firm, which specializes solely in Java at the server side, has offices in Scotland, England and Australia. Alan is the author of two Java servlet books, and contributed to the Servlet API. He has a Web site at www.n-ary.com.

shame, as I for one believe Blair is one of IBM's best-kept secrets. I'd never met a real live IBM'er – well, at least not without a press agent in attendance, censoring his or her every word, so it was a real breath of fresh air to meet Oma and Blair. One term Blair introduced me to was the notion of Big Iron, which is used to describe large servers. I thought this rather sweet.

The Cool Way to Travel

As you all know, I'm experiencing a country revival with all these Dixie Chicks and Dolly Parton CDs floating around. You'll be pleased to know that I think this is beginning to subside. I can't be sure yet, but I'll keep you posted on the status. I've been doing a bit of traveling lately and I have to say that MP3 has been keeping me company throughout my travels. It's kinda cool to travel with over 40 CDs on your portable. Anyone who's experienced this world knows only too well what I mean.

I'd better go now. I'm on a fitness drive, to get the body beautiful. Let me tell you, it's a long haul. Right. I'm off for a swim now....

alan@sys-con.com

Mastering Java Security Policies and Permissions

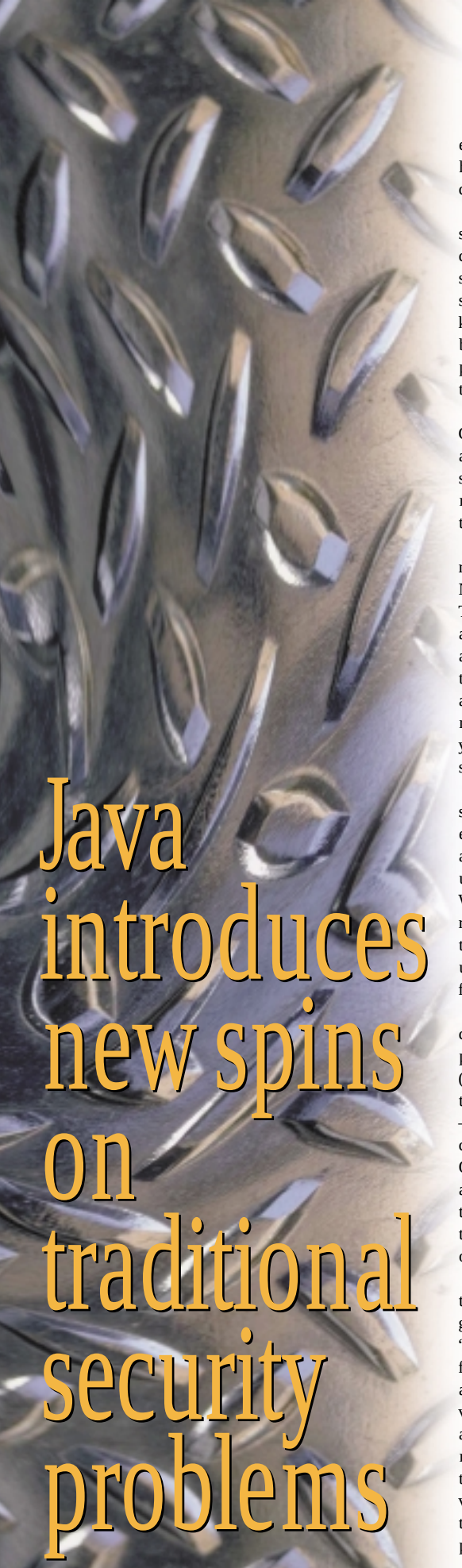
WRITTEN BY PATRICK SEAN NEVILLE

Despite recent high-profile attacks, application developers often consider security something to tack on at the end of a project, and history encourages this approach. It's unusual to see security considered a fundamental design element in a programming language.

But here's where the Java 2 platform deviates radically from the norm – security is in fact a crucial design goal, interwoven into the language's core bootstrap mechanisms. Even before a virtual machine loads its first class, it's aware of specific security policies and permissions.

Unfortunately, security policies are one of Java's more confusing elements. Many developers find them outright irritating. To make matters worse, a great deal of printed and online tutorial code – even from official sources – circumvents security features by granting all permissions to all code. Usually there's some sort of caveat that these sample policies shouldn't be used in real-world applications. This isn't particularly helpful; real-world policy examples remain scarce, and too many applications remain dangerous.

This article exposes Java's treatment of core security. I'll cover how the model operates throughout a class's lifecycle, its internal logic and how it can be extended for smarter code.



Java introduces new spins on traditional security problems

Background: Security Fundamentals

Computer science served as a catalyst in the evolution of burglary. All the traditional attacks hold true here but have mutated to fit a new context.

Secrecy attacks represent the most familiar security challenge. Examples include a thief who cracks a mail server and steals messages, and a snoop who records an online shopper's SSL session and eventually recovers the cryptographic key that can decrypt it. It could also include bribing or tricking someone into divulging a password (social engineering and "rubber hose" techniques).

Integrity attacks involve the altering of data. Culprits include an attacker who successfully alters records in a database or intercepts a message from a server and replaces it with a new message before a client receives it – a "man-in-the-middle" attack.

"Trojan Horse" or "Trojaned" applications represent a special case of integrity attacks. Named after the device that sealed the fate of Troy, these attacks occur when a person replaces an existing application with a new version or adds some malicious code to a familiar application. Imagine, for example, a scenario in which an attacker replaces your Web browser with a nearly identical browser that secretly sends all your credit card information to some hidden server.

Not all burglaries are committed artfully; some are brute-force muggings. The computer equivalent of a mugging is a "denial-of-service" attack, an attempt to render a server or network unusable. For example, an attacker might flood a Web server with a tremendous number of requests. While attempting to manage this load, the attacked machine or network might be unable to respond to any requests at all, possibly failing altogether and shutting down.

It's worth noting that people don't challenge computer security. Crackers (those who trespass on computer systems) and phreakers (eavesdroppers who monitor telecom network transmissions) get the press, but it's their tools – that is, code – that do the work and, as application developers, we're concerned with code. Of course, malicious code doesn't come from attackers only; it could amount to poorly written apps that inadvertently compromise system-level resources or hinder the performance of other apps.

Malicious code also includes viruses. Though true viruses are traditionally assembly-language, low-level programs, most modern "viruses" are actually mere macroscripts that function only in specific operating systems or applications. Depending on the effects of a virus, it may result in a secrecy attack, integrity attack or denial-of-service attack. The virus may be delivered through a Trojaned application or Trojaned e-mail message. The fact that viruses rarely victimize Java applications is due to the language's insistence on a firm policy and permission design.

Java-Centric Security Concerns

Just as Java introduces new approaches to computing, it also introduces new spins on traditional security problems. How do we ensure that an object marshaled across an RMI system isn't victimized by a man-in-the-middle attack? How do we defend a portable Jini-enabled device from a denial-of-service attack? Java's security APIs are designed to be extended to address such issues.

There are many pieces to the Java security puzzle spread across several API subsets. First and foremost is the core security package that consists of classes in the java.security and java.lang packages. All Java applications, including Enterprise JavaBeans, Swing clients and Web applets, are affected by this core architecture. Regardless of whether you delve into cryptography or authentication APIs, and regardless of the scale of your application, the core Policy and Permissions model will affect your Java 2 project even if you stubbornly insist on ignoring security.

The Policy and Permissions model addresses integrity attacks and, to a lesser degree, denial-of-service and secrecy attacks. The reason the model touches on all these attack categories is that it's built on a security truism: the only secure application is one that doesn't run at all. The core security model deals with which applications are capable of running and in what manner they're permitted to function.

Since application developers are personally required to deal with the security model, risky applications can rarely be developed unwittingly. If a method includes a known security risk, it simply won't execute unless some developer acknowledges the risk and specifically grants a permission.

Policy-Based, Permission-Driven Architecture

Generally speaking, a Permission is simply a statement that some "thing" can execute a particular "action" on a particular "target." This notion is common to most multiuser operating systems and platforms: a user (the thing) may have Permission to access a particular file (the target) and read it (the action). Java sticks fairly closely to this definition.

Here's the architecture in a nutshell: specific code (described by a CodeSource) can perform certain actions on certain targets (described by a Permission). Policy objects manage the relationships between CodeSource objects and Permission objects (see Figure 1).

Permissions concern classes, not objects. They deal with what a piece of code is allowed to do regardless of its instantiations and who's executing it. To make matters even simpler, Permissions are always positive – they never deny actions, only grant them.

The relationship between CodeSource and Permissions is encapsulated in a policy and available before any application logic is even considered by a virtual machine. Consider an object's lifecycle (see Figure 2).

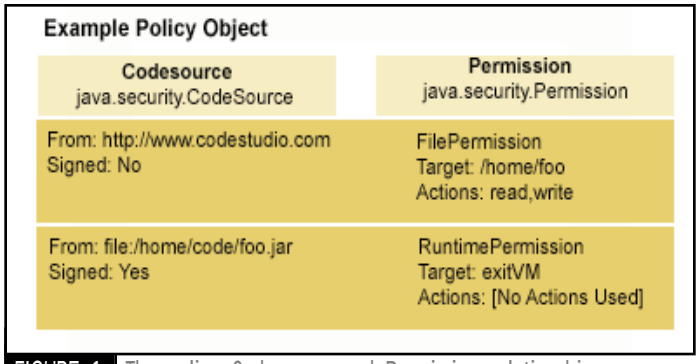


FIGURE 1 The policy, Codesource and Permission relationship

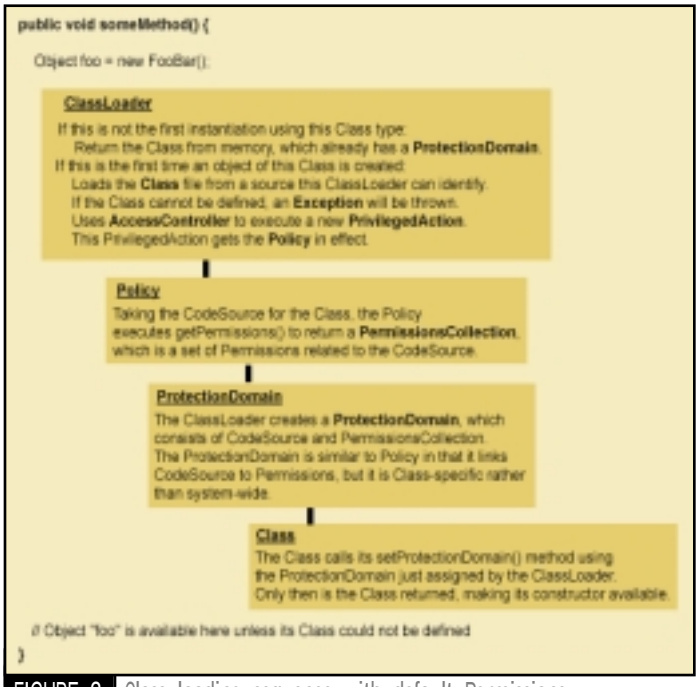


FIGURE 2 Class loading sequence with default Permissions

When an object is instantiated, its class must be defined first. Based on the class's CodeSource, it's then linked to a set of Permissions that a policy has prepared for it. The virtual machine is aware of how to construct this policy upon start-up (more on this later). Finally, an instance of the class is created.

When that instance is subsequently acted on through method invocations, a SecurityManager may check its Permissions at runtime, throwing AccessControlExceptions if any class in the stack of executions lacks the necessary Permissions.

This is simplifying things a little. Behind the scene, matters are a bit more complex. For instance, the ClassLoader will elegantly group Permissions in ProtectionDomains and associate a loaded class with a ProtectionDomain instead of directly to a PermissionsCollection. Furthermore, you probably recognize that ClassLoaders are themselves classes loaded by other ClassLoaders. In fact, there's a chain of ClassLoaders leading backwards up a hierarchy that begins at the virtual machine's special, primordial, native ClassLoader, which is completely inaccessible (it's returned as null). And we haven't even mentioned the complexities of checking every class in an execution stack for the proper Permissions at runtime.

As application developers, we usually don't need to concern ourselves with these details. Of more importance, everything in the model is extendable, so we can create custom Permissions, require security checks wherever we like in application code and craft custom policy objects. Moreover, we can easily create different policies for different users, altering end-user functionality without touching the source code itself and without tangling with the SecurityManager.

Cracking Open the Policy Matrix: CodeSource and Permissions

Policy is fairly simple. It links code to Permissions in a more or less key=value format. Its contents, however, are worthy of a closer look.

CodeSource, defined in java.security.CodeSource, identifies class files for security purposes. Don't confuse it with *codebase*, the property used to describe the root location of an applet or application.

CodeSource characterizes classes by location and by the identity of their creator or deployer. The first trait is straightforward – we refer to location using either file or HTTP protocol. The second characteristic is a little trickier. Java code collected in a JAR file can be cryptographically signed by a specific identity. This process allows a virtual machine to verify the application's integrity.

Here's an example: imagine that Alice has created an application and distributed it in the archive "code.jar". Bob downloads code.jar and wants to grant special Permissions to it. But he wants to be sure no one has tampered with or Trojaned the code.jar file, so he doesn't grant Permissions to any file named code.jar but only to code.jar archives that Alice has signed. The authentication mechanism runs behind the scenes, using KeyStores. A good resource for information on signing code and the nature of digital signatures is *Java Cryptography* by Jonathan Knudsen.

While CodeSource is defined by only two characteristics, Permission objects offer hierarchical variety. The abstract top-level permission object, java.security.Permission, introduces the notion of a target and a series of actions. Its subclasses are AllPermission, FilePermission, SocketPermission and BasicPermission. BasicPermission in turn has a number of subclasses, most of which have no "actions" associated with them.

When you're creating your own Permission objects, you're free to subclass either Permission or BasicPermission. Use of AllPermission should be avoided even for testing code. As its name suggests, AllPermission passes all security checks. Instead of using AllPermission, try assigning your code the specific Permissions it needs from the outset.

A Permission's type will relate to the general function it addresses – FilePermission objects relate to file I/O functions. The possible targets associated with a Permission should follow reasonably from the type – a FilePermission will have targets that are files. The actions will in turn follow reasonably from the target, so a FilePermission might provide actions of "read" or "write" on a particular file target.

A word about actions – not all Permissions have them. In fact, most of BasicPermission's subclasses don't use them. RuntimePermission, for example, lists a number of possible targets but no actions at all. The targets themselves, however, imply an action. For example, the RuntimePermission target "createClassLoader" implies a target of "ClassLoader" with an action of "create." Since there's only one possible action, it's included in the target as a sort of shorthand. This seemingly arbitrary inclusion and exclusion of targets and actions doesn't exactly make Permission usage intuitive. Until you become accustomed to the specific Permissions, refer to the APIs for guidance (a good reference is http://java.sun.com/products/jdk/1.2/docs/guide/security/Permissions.html).

Mastering Policy Files

In the default Java 2 implementation you should store policy information in the same text files in which the VM instantiates policy objects.

If you're interested in creating a new security provider, you could craft an implementation that stores the policy data in a database, a smart card or even an Enterprise JavaBean. For the purposes of this discussion I'll focus on the existing flat file implementation provided by Sun.

Out of the box, Java uses one default file to generate its core policy object. This policy must be available to every virtual machine so its data source will always be in the same place: ${JAVA_HOME}/jre/lib/security/java.policy. You can create additional policy files by following its syntax.

Phrased loosely, the file format simply says "grant" (some code from a specific location – this is optional – signed by some certificate – this is also optional) permission to access a "target" using a specific "action."

The codeBase and SignedBy attributes are optional. If you don't include them, the grant will apply to all code. You can also employ the wildcard "*" symbol, though wildcards succeed only when used in a range denoted by a period "." or when used by themselves. For example, "*" and "a.*" are both valid, but "a*" isn't and neither is "a*b" (see Listing 1).

Once you've written a policy file, you can integrate it in a number of ways. To dynamically link policy files, include the file as a runtime parameter and add the following flag to your java invocation: -Djava.security.policy=[fully-qualified filename]. For instance, the command "java –Djava.security.policy=/foo/home/myperms.policy Foo" will execute "Foo.class" with a policy instantiated using the file "myperms.policy".

You can also drop the policy file in a user's directory to make it available for all code run by that user. This is a handy tactic for systems administrators who need to assign different Permissions across a user base. This method is possible because the java.security file specifies user directories as appropriate locations for policy files. Of course, this line can also be removed to prevent users from creating such policies.

Finally, you can make your new policy file globally available to a virtual machine by creating an entry for it in the java.security file. This has the disadvantage of requiring a separate installation procedure when code is moved across platforms, but may be useful for granting privileges to applications that are seldom redeployed.

To install a policy file statically in this manner, take a look at the following lines of the java.security file:

```
policy.url.1=file:${java.home}/lib/security/java.policy
policy.url.2=file:${user.home}/.java.policy
```

You need to add a similar line. For example, if you have a policy file named myapp.policy located in the "/usr/local" directory, you'd add the following line to the entries in java.security:

```
policy.url.3=file:/usr/local/myapp.policy
```

This will inform the virtual machine of your policy object every time Java starts.

A simpler way to create and add policies is to use Java's policytool, a GUI interface to the process. Since policytool hides syntax and usage, it's important not to become overly reliant on it. Policytool is mostly helpful for administrators and nonprogrammers. It's located in the $JAVA-HOME/bin directory and can be executed from a command prompt with the command "policytool."

What Happens in a Security Check?

While application developers typically needn't bother with the actual routines involved in Permissions checking, understanding the low-level functions may aid in the security design process. The relevant objects are SecurityManager, AccessController and AccessControllerContext (see Figure 3). First, SecurityManager comes into the picture. When a method requires a Permission to properly execute, that method retrieves the active SecurityManager and calls its checkPermission() method. Who decided which core methods in the Java language are protected? That is, who determined that java.io.FileInputStream.read() should ask a SecurityManager before executing? The language's security designers made these decisions and the existing standard Permissions indicate their choices.

You can protect operations in your own application code by adding the same code block that the Java designers used:

```
SecurityManager sm = System.getSecurityManager();
If (sm != null) sm.checkPermission(perm);
// if the permission isn't granted, an AccessControlException
// will be thrown.
```

If you want core Java code to pass more security checks than the designers created, you must develop subclasses and require security checks in their methods. For example, if you decide there's a need to prevent the creation of large strings, subclass string and add the SecurityManager.checkPermission code block to any constructor or method in it you wish to protect. Theoretically, you could force a Permissions check before your application executed any method in the language.

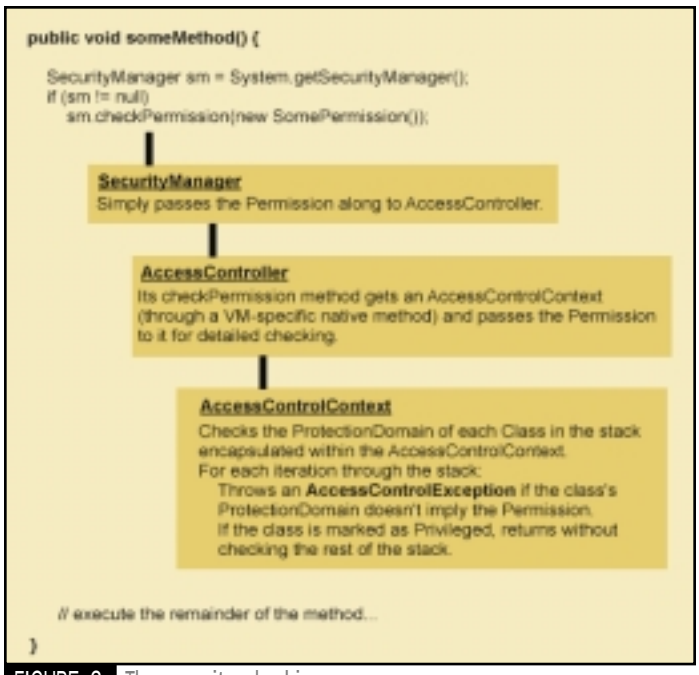


FIGURE 3 The security checking sequence

SecurityManager, however, runs interference only for your code. In the current implementation its checkPermission() method does just one thing: it passes the Permission off to AccessController.

AccessController is the true workhorse of the security core. It's a complicated beast, relying on a number of native calls, and it's difficult to do it justice in a few short sentences. Essentially, when checking a Permission, AccessController gets a snapshot of the current execution stack, which is encapsulated in an AccessControllerContext that uses native, VM-specific methods to gather an array of classes that represent the current calling Thread's stack. AccessController then checks the Permissions (in the form of ProtectionDomains) associated with each class in this stack. If any class – even one – lacks the Permission being tested, an AccessControlException is thrown. Otherwise the method returns silently.

Higher Powers: The doPrivileged() Block

You can see that eventually security checks equate to whether all classes in a stack analyzed by a particular AccessControlContext are linked to the necessary ProtectionDomain. But there's a way to request that the calling class be tested only for the necessary Permission, not the entire stack. The slightly unwieldy yet useful doPrivileged() block permits such functionality.

Say Admin.class has Permission to read a secret string. User.class does not – and should not – have this Permission. Yet in certain circumstances a User object needs to be able to execute Admin's sensitive method and get the value of that string. You can permit this without granting User.class access to the value's data source by using the doPrivileged() block, which looks like Listing 2.

Possible uses for such a code block might include changing a password or reading a KeyStore file. The doPrivileged block permits such actions in isolated code snips without requiring developers to grant a broad Permission to an entire application. Since all core Java classes are granted all Permissions, it can be used to execute sensitive methods even if a client application has no Permissions at all. *Note:* If the doPrivileged block returns a value, it'll always return an object, so it must be cast to the appropriate type. It's obviously dangerous – suitable only for short functions – but it's an extremely useful tool when used cleverly.

Determining What Permissions Are Needed

Judging by posts to various Java mailing lists, deciding what privileges your code needs is often a frustrating endeavor. This may be particularly true if you're porting jdk1.1 code to the Java 2 platform, which suddenly requires all sorts of Permissions to execute methods that previously worked fine.

There's currently no tool available to automatically generate a policy file for specific code. The best way to tackle the task is to plan with Permissions and policy in mind from the beginning and design accordingly. This approach has the benefit of putting a great deal of AccessControl checking at the designer's disposal so that developers can better customize applications for different use cases.

If a developer misses the security element at design time, he or she can solve the issue through runtime testing. Pay close attention to the thread dump that'll appear when an AccessControlException is thrown. This dump will explicitly report which Permission is missing. Try adding that Permission to the policy file and test the code again. If it succeeds, though, consider the implications of adding the Permission before you commit to it. Consider whether a doPrivileged block or some alternative might provide better security. At the very least, be aware of the risks of any Permission you grant and watch for exploitation of those risks during your program's operation.

Keep in mind that without using doPrivileged(), every class in a thread of execution must be granted the Permission for Java to be happy. If class A calls a method in class B that calls a method in class C that requires a Permission, all three classes must be granted that Permission.

Conclusion

It's impossible to craft completely safe applications. Short of not developing or executing code at all, there's simply no such thing as absolutely secure software. But by understanding the Java 2 security core and integrating it into the design process, you'll find your applications are highly resilient to many common attacks and ensure that your system isn't compromised by malicious Java code.

For further information on Java security, check the official documentation at www.javasoft.com/security. An excellent in-depth study of the architecture is the highly recommended *Inside Java 2 Platform Security* by Li Gong, the Sun scientist who led its development.

Author Bio

Patrick Sean Neville has produced Web-based applications and Java systems for television companies, advertising agencies and a variety of new media companies. Creator of The Code Studio (www.codestudio.com), he's also the chief Java engineer at Stockback.com.

neville@codestudio.com

Listing 1

```
grant codeBase "file://home/code"
SignedBy "Admin" {

 // example with no action
 permission java.lang.RuntimePermission
 "setFactory";

 // two examples with action fields, one
 // containing a wildcard
 permission java.net.SocketPermission
 "*:*", "connect,accept";
 permission java.util.PropertyPermis-
 sion "user.home", "read";

};
```

Listing 2

```
/*This is a method in Admin.java, which
any class may call.*/
dmin.java, which any class may call. */
public String getValue() {
 String result = (String) AccessCon-
troller.doPrivileged(new PrivilegedAc-
tion() {
              public Object run() {
 // Admin has permission to access this
 // method
 // even if its calling classes do not
    return readSecretValue();
     }
              });
    return result;
}
```

Download the Code!

The code listing for this article can also be located at www.JavaDevelopersJournal.com

Assuring Reliability of Enterprise JavaBean Applications

WRITTEN BY
TODD SCALLAN

The Enterprise JavaBean specification demonstrates the evolution of distributed objects from middleware to application components. In this article we'll discuss where EJB fits into the distributed object landscape.

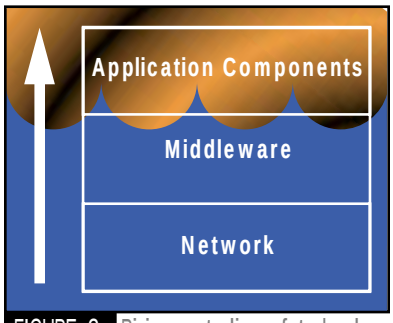


FIGURE 2 Rising waterline of technology

Middleware in the Middle Tier

The three-tier architecture is fundamental to the deployment of applications on an Internet-based infrastructure. A Web server fronts the middle tier (see Tier 2 in Figure 1), and there are many systems behind it that constitute the value-add for an e-business application. From a practical perspective, e-business applications are assembled from disparate components and must be accessible to Web clients (Tier 1) through the Internet and reach relevant data repositories (Tier 3), such as financial data or travel reservations.

Middleware is the glue that integrates systems within the middle tier. It plays a vital role in application reliability since integration points among the various systems are potential points of failure. As distributed object technology has become well accepted within mainstream software applications, standards such as CORBA and Java's RMI have emerged as preferred object-communication mechanisms. However, from an application perspective, these technologies present a rather low abstraction level, leaving the task of defining how components should interoperate to the programmer. This means inventing proprietary programming interfaces that must be adhered to by all parties involved with the creation of the application.

One fact of life concerning technology is that the "waterline" is always rising, making technology more accessible and easier to use (see Figure 2). In the Internet era the waterline corresponds to the higher level of abstraction presented by technologies for assembling Web-based applications. More to the point, software frameworks provide a simpler model for creating e-business applications, supporting the development of software components that can easily work together.

An application component implements some core business logic for an application. A framework is a software infrastructure that defines certain rules of engagement among application components. These rules are typically implemented as services with standardized interfaces that allow components to interoperate without prior knowledge of other components' existence. The ability to cut and paste information between different desktop tools is a classic example. A framework also provides an execution environment for end users and applications, as in the Microsoft Windows desktop.

Although frameworks and component models offer convenience for quick assembly of applications, only those based on proprietary technologies and captive install bases, such as Microsoft's OLE for desktop integration, have historically enjoyed widespread industry acceptance. But thanks to the Internet, the tide appears to be turning for an open standard framework as the industry embraces EJB.

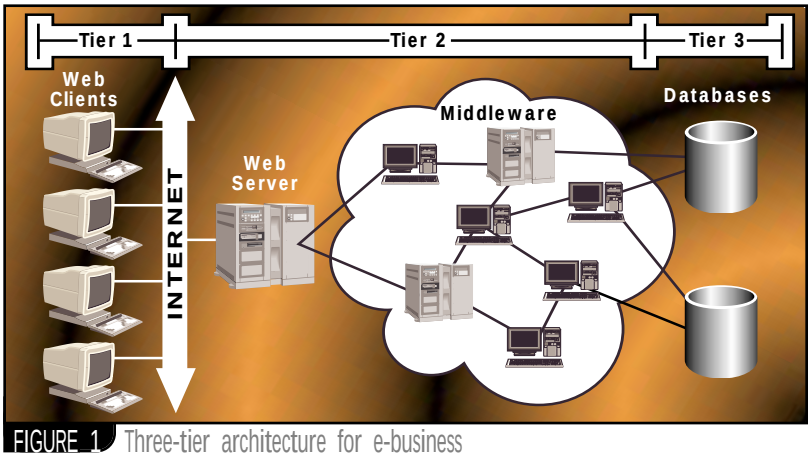


FIGURE 1 Three-tier architecture for e-business

Relationship of EJB to CORBA and RMI

CORBA is an open industry standard that facilitates remote object invocation via published interfaces. However, it doesn't yet specify a market-accepted component model for plug-and-play interoperability. (We'll touch on CORBA's new component model specification in a little while.) To illustrate this point, contrast CORBA with its parallel universe, Microsoft:

DCOM is to CORBA as OLE is to _____.

In other words, DCOM and CORBA provide the communication substrate in their respective universes. OLE is the framework that sits on top of DCOM; you'd experience it daily as a Windows user. But in the CORBA universe, what is analogous to OLE? What provides the higher abstraction level moving up the waterline? The answer isn't particularly clear yet, but EJB seems to be emerging as a viable solution for e-business applications.

The EJB specification defines an execution and services framework for server-side Java components. Application servers house EJB containers that together manage threading, transactions, object lifecycle and other EJB needs. EJB's growing acceptance as a standard component model for middle-tier applications is supported by commercial products from industry movers like IBM, BEA Systems and Oracle, and by increasing adoption by customers. For an EJB to be accessed from outside the framework, its container must expose a public interface through CORBA or RMI (see Figure 3). Since RMI is a Java-only communication mechanism, it can be used over IIOP to speak to non-Java applications.

It should be noted that the CORBA Component Model (CCM) – part of the recently adopted CORBA 3.0 specification – defines a framework for applica-

tion components implemented in any major programming language. CCM extends the EJB model that will allow Java and other programs to operate on the same CORBA platform.

Challenges for Objects in E-Business

With the rapidly growing need to integrate heterogeneous systems and enable applications for e-business, system components are being tied together as distributed objects. These can be new application modules, such as EJB components, or existing applications that are encapsulated in order to be integrated into the environment. Distributed objects exhibit some very useful characteristics:

- Objects can be located anywhere across a network, obviously a necessary attribute for distributed applications.
- Transparent access allows clients to interact with servers regardless of physical location. The underlying CORBA or RMI infrastructure takes care of locating a server resource. Transparency is also very important to developers as it allows them to build applications without having to worry about networking or communication details.
- By using CORBA's IDL for describing object interfaces, an application can be implemented in the programming language that's appropriate for the task at hand and can reside on any platform. For example, a Java object running on a Windows platform can invoke a C++ object on a UNIX platform. The Java client isn't aware that the server is implemented in C++ since it's communicating with the server using native Java constructs.
- Distributed objects provide the basis for building and deploying component-based applications capable of supporting e-business.

The sophisticated nature of three-tier e-business applications presents several challenges for successful deployment:

- ***Complexity:*** A typical e-business site consists of many different system components, including various platforms, application modules and network connections. Distributed object applications introduce a whole new development paradigm that requires deployment and maintenance issues to be considered as early in the development cycle as possible. Ask yourself: How will I diagnose system problems in the lab or the field? How will I control or log usage of my server objects?

- ***Coordination:*** In a system with this magnitude of diversity and distribution, there are many participants in the creation and deployment of the various components. It's no longer typical for an entire application life cycle to be managed by a single department; input must be marshaled from multiple sources. Questions of coordination include: Does the client application developer understand the semantics of the interface presented by a particular server? Is the correct version of the server implementation being used?

- ***Change:*** In the age of e-commerce, applications are updated more rapidly. Delivery cycles can be months, weeks or even days. Numerous e-business applications are built every day only to fail in production due to a lack of scalability, flexibility or reliability. Middleware products are changed from one vendor implementation to another or even between implementations from the same vendor – witness the existence of multiple application servers from leading vendors. Moreover, changes to e-business applications are introduced incrementally into live systems since the entire system can't be brought down to roll out a new release of a component.

Before going live it's imperative to test the system and validate that it'll perform correctly. Rapid change introduces uncertainty: Will the system function properly? Will it scale with anticipated loads in production? What happens if a server object fails? The ability to methodically capture test cases and performance metrics is essential to quantifying the success of change. Even after ensuring that the system will work in a test environment, the behavior of the live system must be monitored. This will uncover potential problems and allow iterative improvements to be made, thereby maximizing the system's reliability.

Application servers based on the EJB specification facilitate the creation of Web-based e-business applications. Such applications operate in the middle tier and rely on distributed object communication for interoperability. To make sure an EJB implemention will run reliably, one can test its behavior and scalability from outside the framework. This can be achieved by exercising functionality and simulating usage models through public interfaces via the CORBA or RMI protocols. Test automation can help streamline the process of assuring the reliability of an EJB application. Segue Software's reliability solutions address EJB functionality and scalability, and offer mechanisms for diagnostics and control in a distributed object environment.

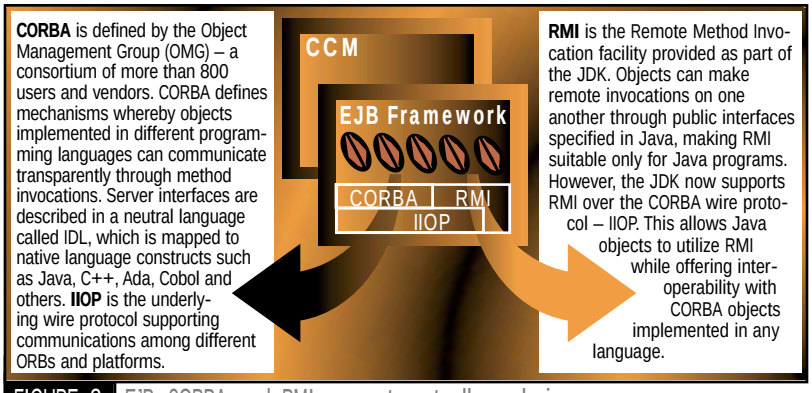


FIGURE 3 EJB, CORBA and RMI are not mutually exclusive

Conclusion

The power of distributed object technologies such as EJB lies in the support for object-oriented designs and distributed implementations. The payoff for a successful component-based application can be substantial, but the costs can be even more significant if the proper steps aren't taken to guarantee the quality of design and implementation. To ensure the reliability of e-business applications, it's necessary to cover the following essentials for successful deployment:

- ***Function:*** Validate that each component functionally behaves as expected.
- ***Scalability:*** Ensure that components perform well under various load conditions.
- ***Diagnostics:*** Monitor, troubleshoot and analyze behavior anywhere in the system.
- ***Control:*** Meter usage of and control access to objects.

As distributed objects evolve from middleware to application components, the need for advanced test automation grows. EJB is an early example of industry consensus around application components in the middle tier of e-business, and is therefore driving the technology waterline for test automation upward.

tscallan@segue.com

AUTHOR BIO
Todd Scallan is the director of product management for Segue Software's distributed computing products. He holds a BS in electrical engineering from Lehigh University and an MS in computer engineering from Syracuse University.

CORRECTION
In the November CORBA Corner column, an incorrect URL was supplied for a list of generic software project risks. It should be www.construx.com.

SYS-CON Radio Interview

with BILL ROTH PRODUCT LINE MANAGER, SUN MICROSYSTEMS

JDJ: What is your role in the Java 2 Enterprise Edition, and what do you see as its strength?

Roth: We're extending the Java platform from its traditional base on the desktop all the way into server-centric space. We want to take write once, run anywhere and extend it all the way to the server side so you can write your business applications once, especially Web-facing business applications, and have them run on any app server. The good news is that we're 90% of the way there. We want to make sure that people can deliver the same Java applications on a server anywhere. That includes applications with servlets, JavaServer Pages and Enterprise JavaBeans – its three major components. There are a lot of other APIs under the hood so you can do transactional systems. There are JNDI, JTA and JDBC. Basically, we want to make it easier to run business applications.

JDJ: Tell us about the whole Enterprise system. Would all the APIs within the Enterprise keep changing? How is it going to look to the end developer?

Roth: We're beginning the process of building a platform. Our draft specification is up on the Web at Java.Sun.com/j2ee. One of the things in the platform is a table that states the required elements of the platform, which include Java 2 Standard Edition, Enterprise JavaBeans 1.1, servlets, JavaServer Pages and so on. We're going to build a reference implementation of the platform to prove you can actually implement the stuff we have in our specs. We're also going to have a set of tests – so we can test not only our implementation but other peoples' – and a branding program. You pass the test – the Enterprise developer will be guaranteed. Folks can brand an application server, for example, and be guaranteed that the components will write once and they can run them anywhere.

Bill Roth (r.) being interviewed by Alan Williamson of SYS-CON Radio

JDJ: So whenever a developer develops that Enterprise solution, if they see a server that is Enterprise enabled, they can be sure that it will come with all the necessary components to support their Enterprise solution.

Roth: Exactly. When something is Java 2 Enterprise Edition branded, you can be assured that JavaServer Pages, servlets and Enterprise JavaBeans will all be there. They'll all be versions, but you can begin to see that it exists. Ultimately we're going to evolve the platform to add more functionality. One of the things we may add to the table, for example, in future releases, will be the Java Message Service. That's something that may find its way into the platform. It's a crucial piece of technology, but I think we want to focus on delivering something that supports Enterprise JavaBeans, JavaServer Pages and servlets.

JDJ: You're talking about the server engines here?

Roth: Yes. Compatibility testing is very important to us. We want to make sure that we don't have any fragmentation in the servlet space. What we're working on and what our engineers are working on is a new tool called JCheck. It's a way to guarantee that you've lived up to the requirements. It's not a compatibility kit, and we still preclude anyone from saying that they're servlet compatible, but it'll be a way to get a relatively good idea of whether developers have all the APIs and if they implement them in the way we say and only the way we say. That's how we ensure write once, run anywhere, and that they behave properly. JCheck is currently under development. We have some nuts and bolts to work on, as well as getting it through the lawyers and licensed properly. We intend to make it a tool, just like the JDBC tests that will be available soon in the public domain in binary form.

JDJ: You're now deploying real hard resources to work with the Apache team to come up with Tomcat, which is another open-source community. Are there other examples of Tomcat in other areas of the Enterprise world that are going to pop up?

Roth: I think that we responded to our community source licensing program as well as servlets and JavaServer Pages in response to a direct, concerted effort by the industry; we don't see that in any other place. I think Sun's announcement of its intent to license the source of servlets and JavaServer Pages to the Apache group is a boon to developers. It leverages off the innovation that's able to happen in the open-source environment. I want to make one point, however, and that is that we're licensing the source only. We'll continue to maintain the specification as part of our Java community process, sort of an auditable, verifiable way that we'll move our technologies forward. Apache will be a key member of the expert group, as will other folks in the industry who have a vested interest in this.

JDJ: You chose Apache as your reference, so there's going to be no other reference that's offered for servlets. Why not Netscape Enterprise, which is one of your alliance partners, or JWS, which is a Sun core Web server?

Roth: That's a really good point. In general, I don't think there are many instances of Sun actually licensing commercial products and making a commercial product in a reference implementation. We didn't do it with EJB. We didn't do it with a lot of our technologies. We certainly didn't do it with JNDI. What we're doing is basically taking our implementation and giving it to the community. If we were to take Netscape and their servlet engine, for example, that would pretty much irritate every other vendor. You could say that Apache is a kind of vendor, but it's sort of a new beast. There are really two factors. One, it was our technology going into Apache. Two, based on the market data I've seen, most notably from Netcraft, it's clear who the market leader is. Our goal is to get the stuff ubiquitous, and Netcraft tells me that Apache has – at least if you believe the June numbers – about a 57.5% market share, at least in terms of Web servers on the Internet. It was clear that this was the right process and the right thing to do.

Using the Java Media Framework with Objectivity/DB

JMF provides a platform-neutral framework for displaying time-based media

WRITTEN BY
S. ALAN EZUST

The Java Media Framework API allows developers to incorporate various media types into Java applets and applications, and supports the capture, transmission, playback and transcode of many types of audio and video. There's an implementation written in 100% Pure Java, so the code can be ported onto any supported Java platform.

Objectivity for Java is a cross-platform, scalable, industrial-strength, object-oriented database that runs on NT and most flavors of UNIX. One of its strengths is its ability to store, manage and retrieve large binary objects.

This article demonstrates that the two technologies work quite well together, and will show you how to:

- Represent multimedia objects as persistence-capable classes in Objectivity.
- Write adapters that permit these objects to be read as InputStreams.
- Send the InputStreams to the JMF MediaPlayer.

Motivation

There are a number of advantages to storing multimedia data in an OODBMS rather than leaving them in flat files; many of them are related to containment and management issues. It's easy to add metadata (such as authors, actors, animators and producers) to a multimedia database object by simply adding data members to the classes that represent them. This also leads to the possibility of querying the database based on such criteria.

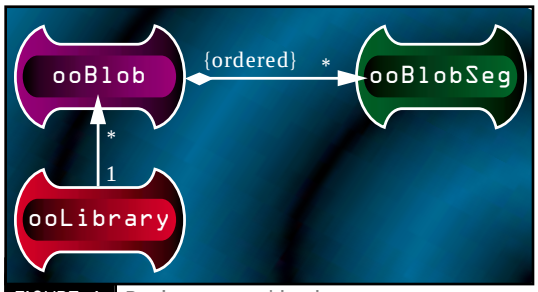


FIGURE 1 Persistence-capable classes

A single multimedia object may be broken up into smaller segments. This means that random access to a particular part of the video or audio clip is also possible. Modifications can be made and a history can be kept on who performed what changes and when. These types of activities would be difficult to do outside a database framework.

Because the data is represented as an object, it's also possible to take advantage of OO features of the Java language to manipulate a multimedia object the same way one would manipulate any other Java object. The different kinds of multimedia data can be represented as different subclasses with inherited and overridden methods (as we've done in this example).

Since Objectivity can easily manage large databases and store gigabyte-size multimedia objects in a single database file, it's an ideal storage medium for such data. Because it's a pure object-oriented database, there's no need to break up the data into relational tables and reassemble them into objects at retrieval time. Because it supports distributed databases, it's possible to distribute the multimedia data across multiple hosts, even replicate parts of the database at multiple sites, and still be able to back up, restore and install all of the stored multimedia data with a simple administrative tool. This allows Objectivity to manage the directory structure of the data so you don't have to.

Persistence-Capable Classes

THE SCHEMA

To store a large multimedia object, it makes sense to break it up into smaller chunks of data. This provides some flexibility in the way we might decide to stream the data out. The persistence-capable classes we use for this application are shown in Figure 1.

The ooLibrary provides a name lookup facility for a collection of ooBlob, so we can store and retrieve many multimedia objects with this application. The ooBlob is a collection of ooBlobSeg objects, and each ooBlobSeg contains an array of bytes that represents a chunk of the raw binary data.

Each class that needs to be stored in the database must be defined as a persistent-capable class. Objectivity provides a semitransparent interface to the Java language, but because there's no pre- or postprocessor, certain things need to be done to the class to make it persistent-capable. First, each class must implement a persistent interface. The easiest way to accomplish this is to extend from ooObj, which implements the persistent interface properly. The getters must then call fetch() and the setters must call markModified(). Finally, all access paths to the data members must be restricted to going through these methods. While this does require some preparatory work to reuse preexisting classes, it permits Objectivity to run on all standard JVMs and development environments without any special customizations or patches.

STORAGE HIERARCHY

Objectivity supports a distributed storage hierarchy with page-server architecture (see Figure 2). At the top level is a Federated Database that provides an entry point into the persistent

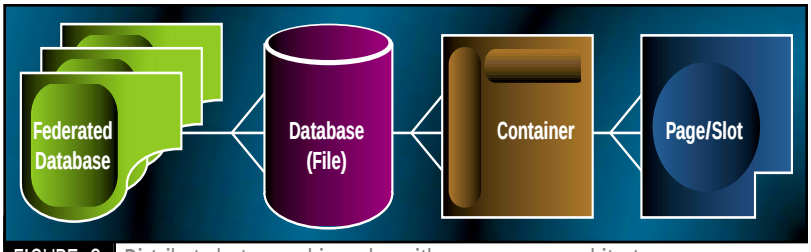


FIGURE 2 Distributed storage hierarchy with page-server architecture

storage. An FD is a collection of databases in the same sense that *Star Trek*'s Federation of Planets is a collection of planets. Each database corresponds to a physical file on disk, but need not be physically located on the same host or network. A lock server process provides the locking services for the entire federation, and each client can talk directly to the page server host after it has received proper authorization from the lock server.

Each database comprises a collection of containers that provides another level in our storage hierarchy. Each object must reside in a container, which in turn resides in a database, which resides in an FD. Containers and databases can be created and destroyed dynamically as easily as objects.

Persistent object references can refer to objects in other databases, so object navigation is still location-transparent across databases. Objectivity permits 65,535 databases per federation and 32,767 containers per database. The page limit per container is 65,535 logically addressable pages, but a large array that spans multiple pages requires only one logically addressable page and can span many more physical pages not subject to the page limit of a container. Since page size can be up to 64KB, this means that Objectivity can address and store petabytes of data without problems.

The locking granularity is at the container level. Since most multimedia objects can fit in a single container, storing each multimedia object in its own container makes perfect sense. This means that once a lock is obtained on the container, a client application can talk directly to the page server that holds the data and stream through the entire multimedia object without needing to stop in the middle to communicate with the lock server process. The programming example, which you can download from www.JavaDevelopersJournal.com, can be easily customized to support a larger segment size for larger multimedia objects, but some small modifications to the code would need to be made to allow for a multicontainer, multimedia object.

Streaming the Data

ooBlob has no streaming interface; it's just an aggregate of objects. However, it's fairly straightforward to write a wrapper for ooBlob, which does provide this interface. Class ooBlobStream derives from java.io.InputStream. Each instance holds onto an ooBlob reference and serves out data on demand to the user of the instance. ooBlobStream instances are normal transient Java objects so they're not part of the schema.

ooBlobStream doesn't do fancy double-buffering to ensure that there's always available data. The application has been tested on a Pentium II/233 and can play 2–5 minute MPEG files without any delays. It's possible that double-buffering may be necessary on some platforms and with some multimedia types, but it seems that JMF media player does all the buffering that's required to stream the data smoothly on data that's been tested so far.

What's really nice about the InputStream interface, however, is that you can either wrap a java.io.BufferedInputStream around it, which may provide better performance, or write your own multithreaded double-buffered input stream, ensuring that there's always data available to be read. Such a class could be quite general-purpose and need not be aware of the fact that the stream it's buffering is coming from Objectivity.

Thread Policies

Objectivity supports multithreaded clients, but it also supports multiple database sessions per process. It may be desirable to have a strict mapping of threads to sessions, where each thread belongs to a session. This is called the "restricted thread policy." If we use this, it means that each thread that accesses persistent data must be explicitly joined to a session before it can proceed, and each thread is joined to one session at a time at most. In our ooBlobSeg code most of the accessors do explicitly join to the session. The reason for this is that, in this particular multimedia application, we have no control over which threads access our data. Perhaps in another application with multiple concurrent open sessions, explicitly joining every incoming thread to a session is either inconvenient or impractical. Thus it may make more sense to use the "unrestricted thread policy," which would allow a thread to implicitly and automatically join the session that owns the persistent object being accessed.

Java Media Framework API

JMF provides a platform-neutral framework for displaying time-based media. The Java Media Player APIs are designed to support most standard media content types, such as MPEG-1, MPEG-2, QuickTime, AVI, WAV, AU and MIDI. The API offers a platform-neutral interface for writing multimedia applications.

JMF doesn't directly support reading from InputStream, so there's still a bit of work to be done before we can play the multimedia objects directly from persistent store. First we must create a class that implements PullSourceStream and, optionally, the seekable interface to provide random access to the multimedia object. The JMFSourceStream wraps an InputStream, a ContentDescriptor and a long that represents the size of the stream.

Next we must create a class that derives from PullDataSource and properly wraps the PullSourceStream. Once done, we can pass this on to the JMF MediaPlayer and play our multimedia object in real time.

Conclusion

Objectivity provides a convenient and high-performance storage mechanism for multimedia files because of its page server architecture. Streaming data in real time is handled quite well without any extra double-buffering.

It took approximately four person-days to write the code that stores and plays multimedia data from Objectivity. I had to learn the JMF API and its limitations during this time. The source code is readily available for download from www.JavaDevelopersJournal.com.

Acknowledgments

I'd like to thank Chad Adams from Payback Training Systems for his assistance in helping me understand the JMF API and Java Multimedia Players in general. I'd also like to thank Al Cherri at Objectivity for his comments.

AUTHOR BIO

S. Alan Ezust is a senior systems engineer at Objectivity Inc. (www.objectivity.com), a leading provider of scalable, mission-critical, object-oriented databases.

ezust@brainwashed.com

PointBase Mobile Edition / Server Edition 2.0

Synchronize data between client and server databases

REVIEWED BY JIM MILBERY

AUTHOR BIO

Jim Milbery is a software consultant with Kuromaku Partners LLC, based in Easton, Pennsylvania . He has over 15 years of experience in application development and relational databases. Jim can be reached via the company Web site at www.kuromaku.com.

jmilbery@kuromaku.com

Test Environment
Client/Server:
Gateway Solo, 256MB RAM, 14 Gigabyte disk drive, Windows NT 4.0 (Service Pack 4)

PointBase, Inc.
2121 South El Camino Real, Suite 1110
San Mateo, CA 94403
Phone: 877 238-8798
www.pointbase.com

PointBase, formerly known as DataBahn and DataBean, is legendary Oracle founder Bruce Scott's latest venture. PointBase, an embedded database that's written entirely in Java, is available for a wide variety of platforms. The main advantages of embedded databases are their ultra-small size, self-management capabilities and portability. The PointBase Server Edition is designed to run on a wide variety of hosting platforms and uses a very small footprint. In fact, the Mobile Edition can reportedly consume as little as 270K of memory on the client. These types of databases are ideal for embedded systems and applications that require the services of a full object-relational database without all the associated overhead. PointBase's Mobile Edition, the smaller of the two versions, has been designed to run as part of a lightweight client application. With the Mobile Edition you can store Web data at the client level without requiring complex client-side installation routines. Typically, developers have avoided using such a technique because the complexity in synchronizing the mobile database with a back-end database is more trouble than it's worth. PointBase's clever "unisync" capability has been designed to address this problem. Through this option you can synchronize data at both ends, making it much more practical to use a client-side mobile database in your applications.

PointBase allows you to manage the database directly using a JDBC-based API. Therefore, programs that interact with the data can also manage the database as part of their normal processing. Despite the fact that the product is meant to use a small amount of real estate, it still comes with an impressive number of database features. PointBase supports both SQL-92 and SQL-99 standards as well as Java stored procedures. With the Server Edition you can create a complete small-footprint Web server platform that includes both the database and an application by using server-side Java code with PointBase.

Installation and Configuration

PointBase offers restricted versions of the Mobile and Server editions that you can download free from their Web site. While they're fully functioning, they only let you create a scaled-down database limited to 5MB. The software is packaged in a zip archive format and the installation files are InstallShield Java Edition. I downloaded the PointBase Server Edition archive that contains the server installation, the client installation kit and the documentation in Adobe PDF format. The installation process itself is incredibly simple and requires little interaction to complete. I had the server software and the client tools installed in under 10 minutes and the server utility running shortly after that, as shown in Figure 1.

Working with PointBase

The PointBase Server includes a simple but useful interface for starting and stopping the database as well as monitoring any activity. You can set different levels of messaging that allow you to watch the activities of the server as it works. PointBase provides three separate interfaces for working with the database – PointBase Commander, PointBase Console and the JDBC APIs. Although I worked mostly with the console, the Commander utility provides a command-line–based interface that allows you to run scripts of commands against the database as necessary. This allows a developer to create batch scripts to build and populate databases without having to write a Java program or being forced to use a visual interface (which doesn't work well for batch-type processing). The PointBase Console provides an interactive interface for working with the database, as shown in Figure 2.

PointBase's marketing literature and documentation are geared toward presenting PointBase as an embedded solution, which is evident when you work with an interactive console. Although I was able to perform most of the standard types of database operations, such as creating tables, viewing the database catalog and importing data, the utility itself is very basic. There are no frills whatsoever, and the forms tend to be sluggish at times, even while running

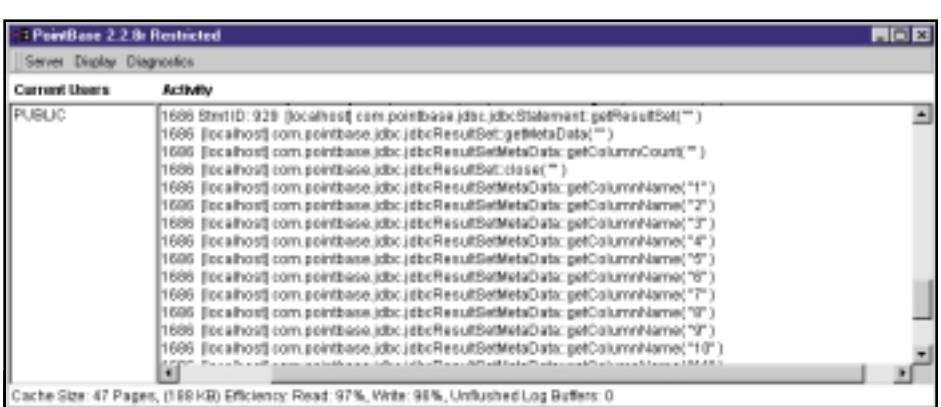


FIGURE 1 PointBase server manager

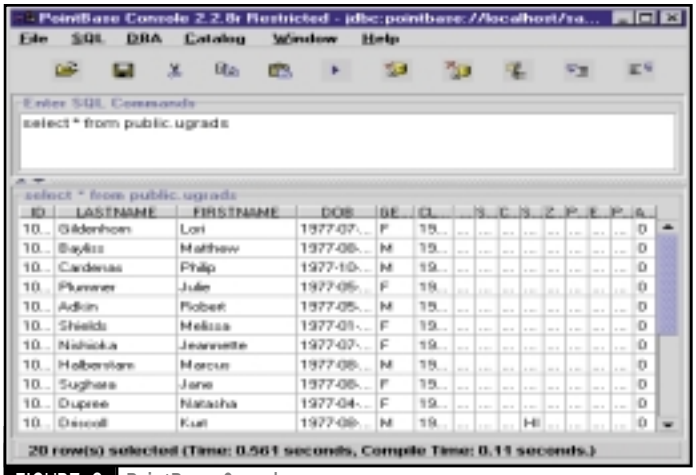


FIGURE 2 PointBase Console

on a fast desktop machine. End users typically wouldn't interact directly with PointBase except through programs that you provide them, so this isn't a real big issue. However, if you're somewhat new to Java and relational databases, you're going to have a tough time working with this interface. Experienced object and relational database developers will have no trouble working with the console; I easily imported some existing Oracle database scripts. The console interface provides a tool for importing data from flat files or from an existing database, and I successfully moved data between the two without difficulty.

The real power of PointBase is its ability to manage all aspects of the database at a low level. Although the database itself is easy to maintain, there are an incredible number of parameters you can set, such as database page size and even encryption algorithms for the database pages. PointBase supports row-level locking, repeatable reads and automatic lock-escalation – quite impressive when you consider the small footprint of the database. With the programming interface you can completely handle backup and recovery within your application code.

Data Synchronization

One of the main features of PointBase is its ability to synchronize data between client and server databases. PointBase uses a publish/subscribe model that allows data to be moved over a variety of protocols such as TCP/IP, HTTP, SNA, JINI, IIOP, e-mail and MAPI. While most database vendors provide similar capabilities for their mobile versions, PointBase supports heterogeneous database synchronization. For example, client databases can be stored in PointBase and then synchronized with a master database that's stored in Oracle. PointBase even provides interfaces for managing the inevitable data conflicts that occur with distributed databases, as well as a sophisticated API for managing the synchronization process. If you don't have a lot of experience working with databases and Java, you might find yourself getting a little lost, as the documentation and example code that comes with PointBase tends to be a bit skimpy.

Summary

Embedded databases will play an increasingly larger role as Web-based applications require more sophisticated client-side data storage options. Relational databases are well known and widely understood, so it makes sense to leverage this experience for lightweight clients and embedded systems. PointBase offers a very comprehensive package from an engineering team with vast experience in the development of database engines. I'd recommend evaluating PointBase if you're in the market for an embedded database.

Writing XML-Friendly Java Documentation

The millennium is here, and it's time to get thinking about documentation again

WRITTEN BY
TIM MOYLE

It's there in every how-to-learn-Java book you pick up, right in the first chapter: a brief section about documenting your Java code. Like any good programmer, you probably flipped through those pages pretty quickly and said, "I'll get back to this later." But you never did, did you? No. Instead you plunged right into writing cute little applets and then bigger applications and pretty soon you were a Java guru. Documentation was never part of the deal.

It's time for you to get back to thinking about Java documentation. Recent trends in the Java marketplace are making Java documentation more and more necessary. The rise in demand for application servers and third-party Enterprise JavaBeans means that there will be a lot of developers working with code that they didn't write themselves. If the primary benefit of using EJBs is a reduction in development time, then a lack of quality Java documentation practically eliminates that advantage.

Another reason that Java documentation is so important is the volatility of the current job market. The accelerated hiring and high turnover in high-tech companies means that the project you're working on now will probably pass through many hands – if it hasn't already – before it's released. How else will you communicate your intentions to future developers except through clear and concise documentation?

The engineers at Sun were quite clever when they first released Java. They incorporated the Javadoc standard, which generates class documentation from source code, into the Java Development Kit from the very beginning. For those of you who didn't even read the first chapter of your Java books, Javadoc parses through your source code and creates HTML files that document every method, property and constructor of your classes. Even if you don't go beyond the basics of writing a Java class, Javadoc will create a very simple HTML file that outlines each of these features. But the true strength of Javadoc is what the developer puts into it. Using special "tags," developers can specify the return types of methods, the exceptions that are thrown, references to other Javadoc files and even the version and author of the class itself.

A recent innovation makes Javadoc even more powerful than it already is. Leading vendors in the Java community are adopting Flashline's new JavaDox standard for Java documentation. The information that was once stored in HTML is stored in the XML (eXtensible Markup Language) format. The difference in the results is dramatic. I've included two different representations of the same method, Integer.parseInt-(Strings). The HTML format (Listing 1), while easily viewable in a Web browser, is practically unreadable in its source. The tags don't convey any information besides how the data is to be formatted. The XML (Listing 2) is concise and easy to understand even if you don't have knowledge of the Document Type Definition (DTD) that defines the rules of creating a JavaDox file. The XML file can be used for many different purposes (including translating it into HTML files with no noticeable difference to the end user), while the HTML files are useful only when viewed in a browser.

There are many other benefits to storing this information in XML. Developers can search across packages, even from different vendors, for the information they need. One of the strengths of XML is that it allows contextual searches. Instead of just searching for any instance of java.util.Vector, you can specify that you'd like to find all methods that return java.util.Vector.

JavaDox also reduces the plentiful number of HTML files that Javadoc creates into one XML file, which can be formatted in many different ways. XML separates the content from the format, which means that Java documentation can be displayed in any conceivable style, from HTML to Acrobat files to PowerPoint presentations to anything you can imagine.

Which leads us to the greatest strength of XML storage: it's nonproprietary. In these days of foul-calling by the Department of Justice, XML is truly the people's tool. Anyone can write an application that reads XML files since it's text-based and easily readable. And with the abundance of XML tools and class libraries, anyone can easily create an XML-ready application.

This is what makes JavaDox exciting. (Trust me on this.) Take me as an example. I do almost all of my coding in Borland's JBuilder. Whenever I type the name of a class followed by a period, a pop-up box appears with a list of all properties and methods that I have access to. JBuilder does this through a combination of introspection and simple code parsing. But imagine a tool that can pull up not only a list of available methods and properties, but also brief descriptions, sample usage and references to other resources. You'll never again be stumped by the arcane intricacies of Java I/O or JNDI; it'll all be at your fingertips.

With all the advantages that XML has over HTML, it's inevitable that Java documentation will continue to move toward XML. That's why it's important now to write XML-friendly Java documentation. Following are some key points to remember:

- *Document every possible feature that you can*. How many times have you been working with a class, and you find a method that needs a string as one of its parameters? How many times have you wondered what that string was supposed to be? Is it a filename, a system property or the name of your first pet? In Javadoc and JavaDox you can comment on the function of any feature of your class, including parameters, return types, overridden methods and deprecated methods. Take advantage of this, and make your fellow developers happy.

- *But don't comment the obvious.* Many times a brief sentence is all the explanation you need. If you've written good, self-documenting code, then, when a method's signature needs a string called fileName, you can assume that developers will know that what they need to pass is, in fact, a file name. (Of course, if it isn't, you've got bigger problems.) In these situations write comments that go beyond the obvious; explain what sort of file your method expects, or explain how this method will manipulate the file.
- *Follow Sun's Javadoc-style conventions.* The first sentence of any description should be brief, and should summarize the functionality of the feature that it's documenting as succinctly as possible. This is especially important in JavaDox, because all descriptions are broken into <briefDescription> and <fullDescription> tags. Write in the third person ("Gets the label") as opposed to the second person ("Get the label"). Begin method descriptions with a verb phrase as opposed to "This method...." For additional style conventions visit http://java.sun.com/products/jdk/javadoc/writingdoccomments.html.
- *Remember that your documentation may not always be displayed in a browser.* Javadoc allows you to embed HTML in your source code. To make your documentation truly XML-friendly, you should avoid this luxury. Since XML formats such as JavaDox can be easily repurposed, there's no telling where or in what format your Java documentation will be displayed. Your best bet is to use HTML sparingly. For example, if you wish to refer to an article published on a Web site, you may have no option but to create a hyperlink. But try to rethink how you structure the information in your comments and avoid using tables, lists and images except when unavoidable.
- *If you do use HTML, make sure that it's well formed.* While most applications of JavaDox don't parse through sections where they expect HTML to be embedded, it doesn't hurt to make sure that your HTML fits in with XML standards. In case you aren't familiar with XML, the most important thing about any XML document is that it be well formed. This means that each tag must be nested correctly, and that all tags must eventually close. In other words, <B><I>Hello</I></B> is well formed, while <B><I>Hey there</B></I> is not, because the tags are nested incorrectly. Closing all tags, however, can seem somewhat counterintuitive, especially with tags that you don't usually think of as needing to be closed. For example, instead of <HR>, you should use either <HR></HR> or <HR/> (the latter is preferred) if you wish to put a horizontal rule in your documentation. The scope of this article prevents me from going into this in detail, but an excellent place to start if you want to learn more about well-formed XML is Elliotte Rusty Harold's *XML Bible* (IDG Books).
- *Use descriptive HTML tags.* <code>...</code> is preferred over ... because it's closer to the concept of XML. Element tags should be self-describing and should reflect the meaning of the data that they contain, as opposed to the presentation.
- *Don't include non-Unicode text in your source code.* XML is only required to support the Unicode character set, and using any non-Unicode character may cause unintentional havoc. Not that you would anyway, right? You'd be hard-pressed to come up with a character that isn't available in Unicode, but programmers are a crafty bunch indeed.

This has been only a brief introduction to XML-friendly Java documentation. For more information on JavaDox, visit the JavaDox Web site at www.componentregistry.com/javadox. There you'll find the DTD that defines the structure of a valid JavaDox XML file, sample stylesheets, the JavaDox application that generates XML from your Java source code and the documentation for JDK 1.2 in JavaDox.

Author Bio

Tim Moyle is a software developer and the manager of developer relations at Flashline.com. He's developed Web-based e-commerce solutions, including the creation of the first auction-based outsourcing system for component development.

tim@flashline.com

Listing 1

```
<H3>parseInt</H3>
public static int <B>parseInt</B>
(<A HREF="String.html">String</A> s)
throws
<A HREF="NumberFormatException.html">
NumberFormatException</A>
<DL>
 <DD>
  Parses the string argument as a
  signed decimal integer. The characters
  in the string must all be decimal digits,
  except that the first character may be
  an ASCII minus sign <code>'-'</code>
  (<tt>'&#92;u002d'</tt>) to indicate a
  negative value. The resulting integer
  value is returned, exactly as if the
  argument and the radix 10 were given
  as arguments to the
  <A HREF="Integer.html#parseInt"><CODE>
  parseInt(java.lang.String, int)</CODE>
  </A> method.
  <DD>
   <DL></DL>
  </DD>
  <DD>
 <DL>
  <DT><B>Parameters:</B>
  <DD><CODE>s</CODE> - a string.
  <DT><B>Returns:</B>
  <DD>
   the integer represented by the
   argument in decimal.
   <DT><B>Throws:</B>
   <DD>
    <A HREF="NumberFormatException.html">
    NumberFormatException</A> - if the
    string does not contain a parsable
    integer.
    </DL>
  </DD>
</DL>
<HR>
```

Listing 2

```
<method accessSpecifier="public" stat-
ic="true">
  <methodName>parseInt</methodName>
  <signature>(String)</signature>
  <return>
    <returnType>int</returnType>
    <returnDescription>
     the integer represented by the
     argument in
     decimal.
    </returnDescription>
  </return>
  <description>
    <briefDescription>
     Parses the string argument as a
     signed decimal integer.
    </briefDescription>
    <fullDescription>
      ...
    </fullDescription>
  </description>
  <parameter>
    <parameterType>
      java.lang.String
    </parameterType>
    <parameterName>s</parameterName>
    <description>
      <briefDescription>
        s - a string.
      </briefDescription>
      <fullDescription>
        s - a string.
      </fullDescription>
    </description>
  </parameter>
  <exception>
    <exceptionType>
      NumberFormatException
    </exceptionType>
    <description>
      <briefDescription>
        NumberFormatException - if the
        string does
        not contain a parsable integer.
      </briefDescription>
      <fullDescription>
        NumberFormatException - if the
        string does
        not contain a parsable integer.
      </fullDescription>
    </description>
  </exception>
</method>
```

SilkPilot 1.2.1

from Segue Software

A tool for testing distributed application components

REVIEWED BY GABOR LIPTAK

AUTHOR BIO

Gabor Liptak is an independent consultant with 10+ years of industry experience. He is currently an architect of a Java e-commerce project.

http://gliptak.homepage.com/

Segue Software
201 Spring St.
Lexington, MA 02421
Phone: 800 287-1329
www.segue.com/ads/corba
E-mail: info@segue.com

Pricing (North America):
SilkPilot Standard Edition: $995 per seat
SilkPilot Professional Edition: $3995 per seat
(prices do not include annual maintenance)

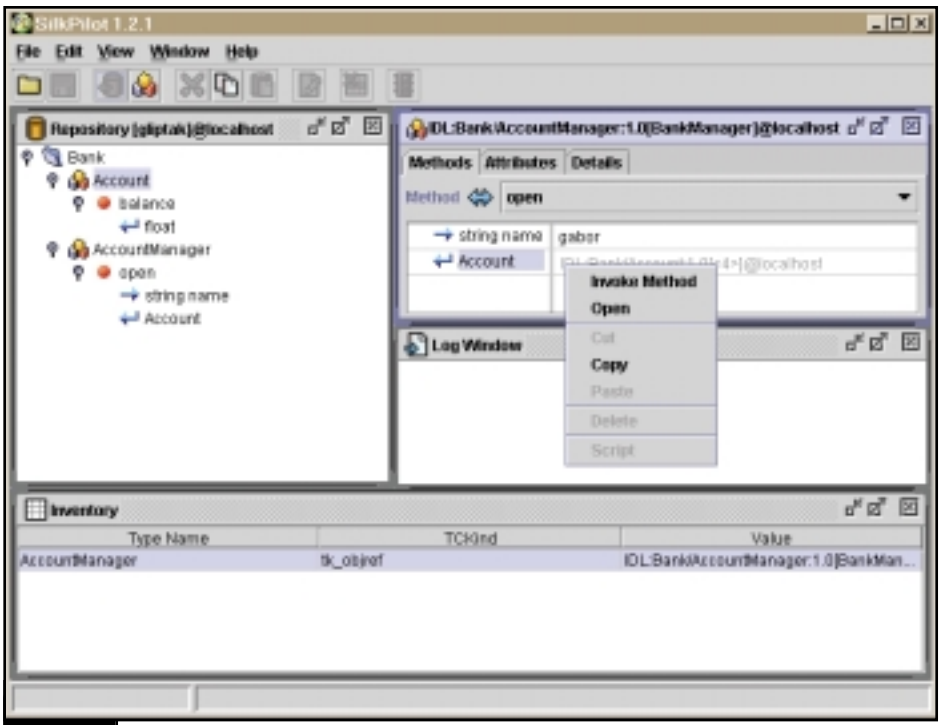


FIGURE 1 Main screen after opening IR and loading BankManager object

SilkPilot is a tool to test distributed application components. The maker of SilkPilot is Segue, a provider of testing tools for the enterprise. Their family of Silk products includes other testing tools for various areas of Web, Java and distributed development.

To test the functionality, I ran version 1.2.1 Professional. It utilizes a nonbundled ORB to provide the CORBA 2.2 distributed environment that's required. VisiBroker 3.4 (or higher) and OrbixWeb 3.2 (or higher) are supported by the product. ORBacus 3.1.3 (or higher) and BEA's WebLogic Enterprise are also supported, but it seems to be incomplete in certain places. SilkPilot is a Java 2 JFC application that's targeted toward Java 2, although with certain modifications it's possible to use it to script JDK1.x-based clients. It runs on Windows 95/98/NT, Solaris, HP and different UNIX operating systems.

SilkPilot is a well-done implementation of this simple idea: the way to test distributed objects is to call them in a distributed fashion and to save this interaction for later repeatability.

The operation of SilkPilot is as follows: it fills the detailed distributed object information for its UI from an IR (InterFace Repository, IFR in the SilkPilot documentation). The user then prepares test scenarios using the UI, and the resulting test objects ("scripts") are saved and can be used independently from the SilkPilot environment for (regression) testing. The ability to save "scripts" is the difference between the Standard and the Professional versions, making the former an expensive distributed "live" object browser and tester.

SilkPilot requires an IR to query detailed information on objects accessed, but it doesn't require access to IDL files. For every session only one IR can be open, but this isn't a major limitation as one can have sessions sequentially. If the vendor-supplied IR isn't available (or

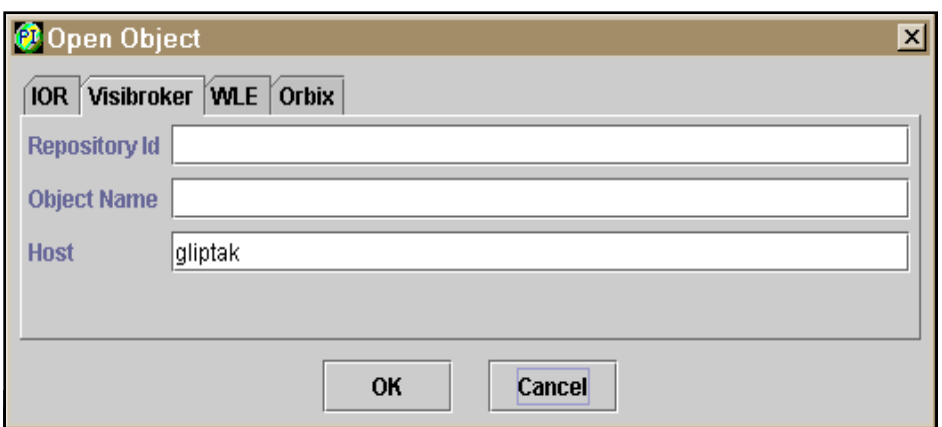


FIGURE 2 Objects can be opened using IOR- or ORB-specific methods

isn't compliant with the current CORBA standard, as I found is the case with Orbix), a "local" IR supplied with SilkPilot can be utilized. In this case the IDL files are required and need to be loaded into the "local" IR.

SilkPilot installed on my Windows NT 4.0 (SP5) machine without any problems, although to evaluate it I had to apply for a "node-locked" install key, required for permanent installations, which I found tedious. I would have preferred a readily running evaluation version, which doesn't have this restriction. During install it found the JDK but not the VisiBroker install on my machine.

The documentation included is detailed and provides hands-on examples on functionality. Context-sensitive help is also provided, using the same HTML documentation.

To demonstrate the available functionality, I prepared and loaded everybody's favorite Bank example, supplied with VisiBroker, into SilkPilot. After opening the IR and loading the BankManager object, the main screen looked like Figure 1.

The top left window lists all interfaces available in the currently opened IR. Using the information from the IR, objects can be created using the built-in "Data Composition" window (not shown) or opened using IOR or ORB specific methods, as shown Figure 2.

Methods can be tested interactively against the objects opened. The interface uses tabs nicely and also uses editable property sheets for displaying and editing various data elements. When the user decides that a method invocation is to be added to the current set, the invocation can be saved using "Script" (available only in the Professional Edition).

After collecting all testing step(s) desired, the results can be saved into a directory. That directory will contain all files necessary to compile and run the testing steps prepared from the UI using command line. In my case the directory contained the following files:

```
Client.java – generated testing code
Client.idl --  snapshot of IDL
Bank_AccountManager_BankManager@localhost_.ior – IOR saved
compile.bat – compile tests
run.bat – run tests
silkputil.jar – helper classes
```

The result of compiling and executing using the run.bat file is text output:

```
name = gliptak
INVOKE Bank.AccountManager.open
result = IDL:Bank/Account:1.0[<0>]@localhost
```

The compile/run scripts generated are OS dependent, containing Windows-specific scripting instructions and scripts generated on a Windows system that won't run on UNIX systems and vice versa. The Java file generated can be compiled and run on other host OSs (but the user needs to create their own scripts to do so). I think the product would benefit if allowed to run the generated script(s) (configuration?) on a different host system.

There are advanced features for preparing more complex test cases. Exceptions and all possible CORBA data types (including primitive and user-defined types) are supported. The UI is well integrated and drag-and-drop of simple and complex data types is supported. However, I found it limiting to be unable to transfer data between the host OS and Swing using the clipboard.

As indicated above, SilkPilot is oriented toward the module-level testing of CORBA distributed objects; considering the tie-ins into CORBA, it may not be a tool your average tester can use to maintain test scripts.

Segue provides an impressive list of customers and partners on their site and, based on my brief evaluation, this is no accident. Currently, SilkPilot is a unique product on the marketplace, and it's worth a closer look if you're in need of a component testing tool.

Java in the Middle Tier

With the advent of J2EE, Java has firmly ensconced itself in the middle tier

WRITTEN BY
AJIT SAGAR

This month the Java platform segues into the new millennium. These are very exciting times; 1999 was a crucial year in the acceptance of Java in the enterprise as one of the key drivers of e-business. It's ironic that applets – the components of Java that helped propel it into the mainstream of Internet applications – currently occupy a backseat in the vehicle that propels Java into the 21st century. During the year gone by, the focus was on server-side Java, as predicted last year by several industry pundits – more precisely, on Java in the middle tiers of distributed computing.

This month in **e-Java** we'll take a look at some of the developments in Java that have had an impact on Java-based e-business. Specifically, we'll examine the segregation of Java into three separate editions, the role of Java servlets and JavaServer Pages and the impact of J2EE in defining middle-tier business logic.

The Three Lives of Java

At JavaOne '99 Sun made an important announcement that defines the future direction of the Java platform. The architecture for the platform was redefined, aimed at making it simpler for software developers, service providers and device manufacturers to target specific markets. The revised structure defines three editions of the platform:

- *Java 2 Platform, Standard Edition (J2SE)*
- *Java 2 Platform, Enterprise Edition (J2EE)*
- *Java 2 Platform, Micro Edition (J2ME)*

Each of the new Java platform editions combines the Java Virtual Machine, the Java programming language, core packages and optional packages. The new editions share many core packages and the Java programming environment. Each comes with its own set of core APIs to enable development of business solutions in its respective domain. Applications on all three editions may be developed using a common application programming model. Those developed using this model should be able to scale upwardly from systems built with J2ME to systems built with J2SE. With the Java 2 platform Sun defines what constitutes the core Java technology platform for the enterprise. A core set of APIs forms the "least common denominator" of APIs offered by the platform. For programmers on virtually any Java platform – from smart cards to the mainframe – that means that 15 class libraries have been declared "core," and thus constitute the soul of all Java technology. These are applet, awt, beans, io, lang, math, net, rmi, security, sql, text, util, accessibility, swing and corba. The three editions of the platform are described below.

JAVA 2 PLATFORM, MICRO EDITION

J2ME is a new edition of the Java 2 platform targeted at consumer electronics and embedded devices. It consists of a virtual machine and a minimal layer of APIs targeted at providing only enough functionality to securely and safely download Java classes to a device and configure the Java environment. The rest of the Java functionality needed to provide a complete Java runtime environment for a particular kind of device is provided within the context of an industry-defined profile. J2ME comes in two flavors, called *configurations*, that are targeted at two broad categories of devices:

- *Devices with 128–512K of memory available for the Java environment and applications*
- *Devices with 512K+ of memory available for the Java environment and applications*

JAVA 2 PLATFORM, STANDARD EDITION

J2SE provides a complete, secure foundation for building and deploying network-centric enterprise applications ranging from the PC desktop computer to the workgroup server. It's implemented by the Java 2 Software Development Kit (SDK), Standard Edition, and the Java 2 Runtime Environment, Standard Edition. J2SE includes the following APIs:

- *Java Foundation Classes (JFC) and Swing*
- *Pluggable Look & Feel (PLAF)*
- *Accessibility*
- *Drag and drop*
- *Security*
- *Java Interface Definition Language (Java IDL)*
- *JDBC*
- *Remote Method Invocation (RMI)*
- *Java 2D*
- *The Collections Framework*

JAVA 2 PLATFORM, ENTERPRISE EDITION

J2EE enhances J2SE to server-side, middle-tier applications that address multitier enterprise solutions. It enables solutions for developing, deploying and managing multitier server-centric applications. J2EE is a unified platform for building, deploying and managing enterprise-class software applications that have the ability to run on a variety of computing environments. The primary technologies in J2EE (in addition to the J2SE technologies) are:

- *Enterprise JavaBeans (EJB)*
- *Java Servlets and JavaServer Pages (JSP)*
- *Java Naming and Directory Interface (JNDI)*
- *Java Transaction API (JTA)*
- *CORBA*
- *JDBC*

EJB technology, the basis of J2EE, provides a scalable architecture for executing business logic in a distributed computing environment. J2EE combines the EJB component architecture with other enterprise technologies to offer solutions on the Java platform for development and deployment of server-side applications.

• • •

If we cut through the entire gamut of Java technologies that hold the promise of enabling the development of enterprise-wide business solutions, Java servlets (and JSPs), RMI and Enterprise JavaBeans emerge at the core of these technologies.

Java Servlets and JavaServer Pages

Java servlets play the role of access mechanisms to middle-tier services. They are a specific enhancement to the world of enterprise computing, not a Java layer on top of an existing service. The popularity of Java servlets is based on the premise that most enterprises are taking advantage of the thin-client environment made possible by the ubiquitous acceptance of the worldwide Web and the Internet. Under this model the Web server becomes enterprise middleware and is responsible for running applications for clients. Servlets are

used primarily by Web servers as Java-based replacements for CGI scripts. They can access server-side APIs in a variety of ways – by sending and receiving e-mail, invoking methods on remote objects using RMI or CORBA, or by obtaining directory information via JNDI and using that information to invoke business services offered by EJBs.

JavaServer Pages are based on servlet technology and are currently poised as one of the most significant elements of dynamic content presentation in e-business applications. JSPs combine markup (HTML or XML) languages with blocks of Java code to produce dynamic Web pages. Each of these pages is compiled into a servlet by the JSP engine the first time it's requested. Subsequent requests for the page use the compiled servlet. JSPs provide a variety of ways to talk to Java classes, servlets, applets and the Web server. Using JSPs, an application developer can split the functionality of a Web application into components with well-defined public interfaces. These components are contained in a single page. JSPs also have mechanisms that allow the application to leverage JavaBean components so as to present different data views to the Web browser. They facilitate the creation of front-end business components such as shopping carts and user profile managers.

Enterprise JavaBeans

Enterprise Java Beans form the core of Java enterprise computing. EJB defines how server-side components are written and provides standard contracts between components and application servers that manage them. EJB promotes the development of a component marketplace in the enterprise, where vendors can sell reusable components that can be purchased to help solve business problems. Thus EJBs offer business services in the form of APIs that may be accessed via a distributed object framework. This framework is offered by Java in the form of RMI.

The idea behind EJBs is to enable the creation of business services in the middle tier. The business objects defined via EJBs are responsible for accessing data from the end-server tier, processing it and making the results available to the presentation tier via RMI. If the presentation tier is accessed via a Web browser, servlets and JSPs are used to make the results available in HTML (or XML) format via a Web server.

Trading Places

When Java first emerged as a popular technology, its scope was more or less limited to the client. Client-side technologies were used to access server-side services via different protocols, depending on the programming environment. With the advent of J2EE, Java has firmly ensconced itself in the middle tier. In the new millennium we can hope to see further penetration of Java in various tiers of the enterprise.

ajit@sys-con.com

Author Bio

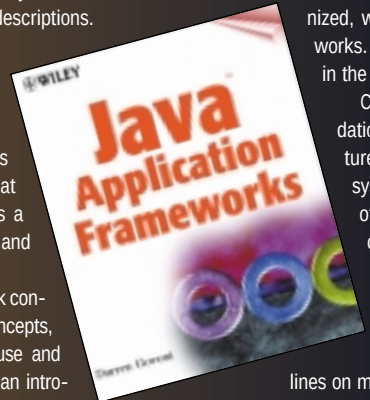
Ajit Sagar is a member of the technical staff of i2Technologies in Dallas, Texas, focusing on Web-based e-commerce applications and architectures. A Sun-certified Java programmer with nine years of programming experience, including three in Java, Ajit holds an MS in computer science and a BS in electrical engineering.

book review

E-Book
Java Application Frameworks

by Darren Govoni
John Wiley & Sons

This month's e-book covers an important topic that relates to industry-strength Java development. The author offers some insight into how to design and use frameworks in Java applications – basically, sound advice and guidelines on design reuse, design patterns and object frameworks. A fair amount of code is used to support the textual descriptions.



The two Java frameworks Govoni covers are Java Foundation Class (JFC/Swing) and the InfoBus. In my opinion, InfoBus is a good framework to study from an academic perspective. From a practical standpoint, however, it's one of the Java technologies that's being phased out (Lotus recently announced that eSuite is being discontinued). All in all, though, it's a good book for readers who are new to frameworks and design patterns in the world of Java.

Chapters 1 and 2 introduce the reader to framework concepts. Chapter 1 presents general framework concepts, guidelines for design and implementation, code reuse and object design – a useful chapter for folks who need an introduction to object frameworks. The concepts are well explained, design models are covered well and the chapter contains some simple examples in Java that supplement the author's discussion. Except for the examples, this chapter should be useful to both Java and non-Java readers. Chapter 2 focuses on Java-specific frameworks. I thought some of the Java concepts explained were too basic, but they may be useful to newcomers to Java. The author uses the Java Collections Framework as the main example in this chapter. Again, this was a basic introduction to the Collections Framework. I found the latter half of the chapter more useful. The author walks the reader through an exercise in framework development in a Java environment.

Eight design patterns – Factory, Abstract Factory, Adapter, Bridge, Builder, Observer, Model/View and Iterator – are discussed in detail in Chapter 3, with diagrams and examples. The author offers useful tips on when and where to apply design patterns. The chapter refers to Gamma et al.'s classic book to set the stage. Although Govoni doesn't offer an exhaustive list of prevalent design patterns, this is a decent chapter that introduces the reader to some of the main ones.

Chapter 4 covers JavaBeans. If you've read about JavaBeans and have worked with them, there's not much to learn from this chapter. If not, this provides a brief overview. Chapter 5 offers coverage of two Java frameworks – JFC and InfoBus. I found the chapter to be well organized, with detailed coverage on Java component frameworks. The Model/View and Listener patterns are revisited in the context of these frameworks.

Chapter 6 introduces the reader to Composite Foundation Architecture (CFA), a methodology and architecture for developing frameworks, components and subsystems within a larger complex system. The chapter offers decent coverage of architectural framework concepts and framework abstraction. The composite foundation architecture is described in relation to Java framework aspects such as packages and directories. A loan application example is used to illustrate the implementation of the CFA. Guidelines on modularity, state management, reuse and extensibility are also given.

Chapters 7 and 8 focus on distributed application architectures in Java. Popular component models such as RMI and CORBA are covered. The responsibilities of the different tiers of a distributed application are discussed. Last, enterprise frameworks such as EJB and CORBA are examined along with interesting perspectives on distributed frameworks and designs based on distributed systems.

If you're looking for depth in Java APIs and component model discussions, this may not be the right book for you. However, if you want a book that will help you design better applications, this is a good one to add to your library.

Pramati

www.pramati.com/j2ee.htm

Design

sign.com/javlin

Securing Your Company Data with EJBs

Harness the power of EJB security to protect company data from prying eyes

WRITTEN BY
JASON WESTRA

Often we think of security as a burden, a time-consuming process that requires us to jump through hoops just to get through a doorway or view a Web page on the company intranet. My first real appreciation for (or frustration with) security came a number of years ago. I was a PowerBuilder consultant in Minneapolis, helping the Federal Reserve Bank build its first-ever client/server application. Each day it was a hassle just to get past the security desk in the lobby, and the bombing of the World Trade Center in New York that year did nothing to ease the pain.

My next interesting security experience came last year on a project for the Department of Defense in Colorado Springs, Colorado. Our team was building a solution for the DOD in reply to a request for proposal they had made to update NORAD's command and control operations with new, distributed, component-based technologies.

On my first day at the site I was escorted to what looked like a bank vault, and I thought to myself, "Is this déjà vu or what"? A metal door about 1.5 feet thick stood between the outside world and me for the next three months. In fact, the development center was actually inside a SCIF (Secret Compartmental Information Facility). The government has several levels of security – one you're probably familiar with is Top Secret. Yes, Top Secret is real, not just something out of "Mission Impossible" or the "X-Files." Information considered even more sensitive than Top Secret is held within SCIFs, where neither cell phones nor pagers are reliable, if they work at all. I was told the walls played music from the inside out to defend against listening devices. I couldn't resist imagining men in an unmarked van outside getting an earful of Bob Marley's "Buffalo Soldier" as they tried to listen in on our design sessions!

Security of information is important not only for the government, but also for companies seeking to protect sensitive information about their business or users. Security can take many forms: swipe cards, retinal scanning or just Bob playing reggae in the walls of your office. However, securing your data over the Internet and intranet is another story. There was no question then that Enterprise JavaBeans, targeted at enterprise solutions, had to include support for security in its specification. This month I'll provide an overview of EJB security and how it relates to keeping your corporate data safe.

EJB Security Model

EJB is an open architecture capable of integrating with existing security standards while sheltering components from advances in technology. With this in mind it's understandable that the EJB security model is both simple and flexible.

The basic concept surrounding EJB security is role-based access control of corporate data. Enterprise JavaBeans isn't as concerned about communications integrity as SSL protocol is, nor is EJB worried about playing a little too boisterously in the "sandbox"! Instead, enterprise beans are believed to be trusted components that are focused on data security.

EJB provides role-based security via ACLs (access control lists) and principals. ACLs are named list entries that are used to control permission to server-side resources. For instance, an ACL may be created to grant system administrators permission to deploy new components into an EJB server or to manage what roles at a bank can perform monetary transfers between member accounts. ACLs typically contain a set of permissions that are associated with the principal(s) on the list. A principal simply refers to a user of the resource – a human user, an application or a component within an application. Principals can be members of more than one ACL. It's left to the EJB server vendor to provide an implementation of role-based security. Each user may have different rights, or they may be grouped together into ACLs with similar permissions.

The EJB security model is flexible. Its component model provides container-managed security enforcement of the context of each call to an enterprise bean. As with other container-managed features such as persistence and transaction management, security permissions never have to be hard-coded by the bean provider. Instead, they can be finalized during deployment of the bean, allowing a more flexible approach to development and reusability. If a fine level of security is needed in the bean, javax.ejb.EJBContext interface provides a means for it to question the caller's identity itself.

How Does It Work?

This month let's take a look at how EJB's role-based security works. I'll save coverage of hard-coded security checks against the principal in EJBContext for a later article. To begin with, there are two aspects of security to define for an enterprise bean: method-level security policies and bean identity.

METHOD-LEVEL SECURITY POLICY

During bean deployment, an AccessControlEntry for an enterprise bean is defined for each individual method and one may be made for an entire bean,

Object Switch

www.objectswitch.com/idc35/

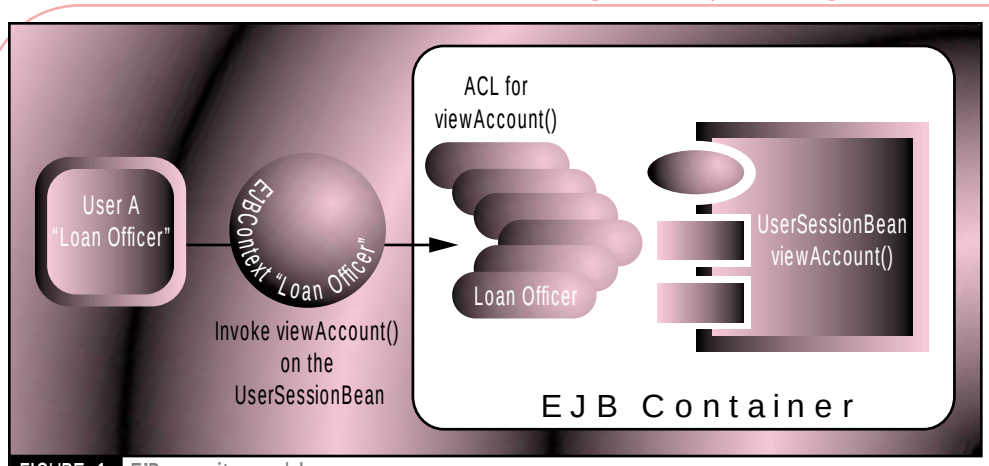


FIGURE 1 EJB security model

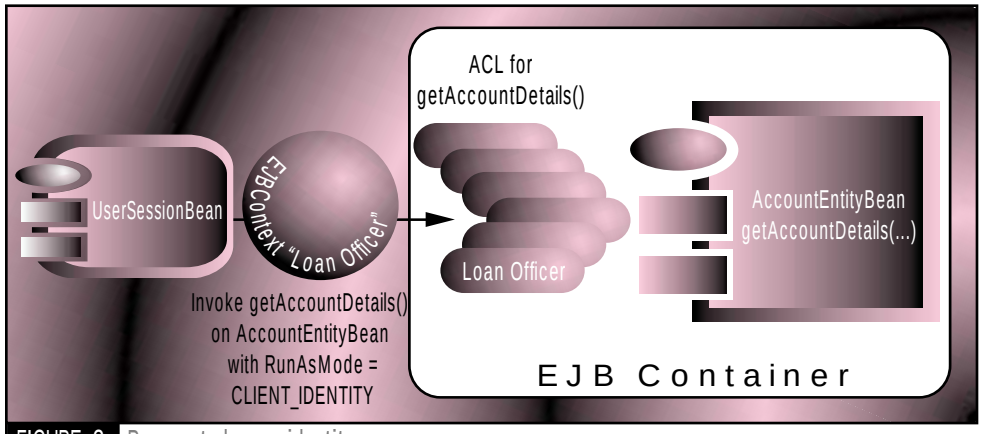


FIGURE 2 Propagated user identity

which applies to all methods without individual security policies. This entry associates a role to an entry in a previously defined ACL of who is allowed to access the bean's method(s).

BEAN IDENTITY

A bean instance has an identity, referred to as the "RunAs" security identity, associated with it at all times. This identity is determined at deployment and can be one of the following:

CLIENT_IDENTITY: Use the identity (principal) of the caller.
SYSTEM_IDENTITY: Use a predefined global system identity
SPECIFIED_IDENTITY: Use the identity specified by the RunAsIdentity security attribute.

When a bean attempts to access a system resource or another enterprise bean, this RunAs identity is associated with the method invocation (see below).

EJB Security in Action

When an operation on an enterprise bean component is invoked through its remote interface or its home interface, an EJBContext object is associated with the client's call and passed to the bean's container. The container uses the context to retrieve the caller's identity, or principal, and performs a lookup against the bean method's ACL to see if a corresponding access control entry exists for the client.

If an entry exists, the bean method is invoked. Also, the "RunAs" security identity of the bean is used for any future calls from the bean to system resources or other beans.

If an entry doesn't exist, permission to execute the bean method is denied, and a java.security.Exception is thrown. At this level it's the responsibility of the EJB Container to audit an attempted breach of security in the system. This detail is noted in 15.6.10 of the specification, but has yet to be formalized.

EJB Bank Security Scenario

This scenario represents a bank that contains sensitive account data for its members. Each enterprise bean in our example (UserSessionBean and AccountEntityBean) is protected through EJB's role-based security model. For instance, a user in the role of Bank Teller may have read-and-write access privileges to Account beans, while other roles such as Loan Officer may only view account details but not make any modifications to the account. Users log in under a particular role and assume that role for all subsequent interactions with the system.

Figure 1 illustrates the security enforcement occurring in the system when a user tries to access an account. A Loan Officer, User A, invokes a UserSessionBean to view an account at the bank. A context containing his or her identity is passed along with the request to the EJB Container. When the container intercepts the method call, the identity of the caller is taken from the context and compared to the ACL entries for the bean's viewAccount() method. For our example let's assume the security check is successful. Next, the UserSessionBean delegates to the AccountEntityBean to actually get the account information.

Figure 2 illustrates how the "RunAs" characteristics of a bean are forwarded to other resources and enterprise beans. In this scenario the RunAs identity is set to CLIENT_IDENTITY and User A's identity is passed onto the AccountEntityBean's EJB Container where it is verified once again for permission to invoke the getAccountDetails() method. Because CLIENT_IDENTITY was chosen during deployment of the UserSessionBean, this bean instance actually assumes the identity of the original caller.

The EJB security model, as demonstrated in this scenario, is simple, yet effective at controlling the access to sensitive bank account information. The permissions were granted in a component-based fashion. Access control entries for each method were determined at deployment of the enterprise beans, and no hard-coded security checks were needed. This fosters flexibility in deploying the bean into new environments, encouraging reusability of the component.

Conclusion

Enterprise JavaBeans provides a simple yet effective and flexible model for enforcing role-based security in your applications. EJB eases the implementation of security requirements by providing you with the ability (1) to graphically define security permissions at bean deployment, and (2) to control access to data at the method level of the enterprise bean. When designing security for your next EJB application, I recommend taking a close look at harnessing the power of EJB security to save precious development time and protect company data from prying eyes.

AUTHOR BIO
Jason Westra is a managing partner with Verge Technologies Group, Inc., a Java consulting firm specializing in Enterprise JavaBeans solutions.

jwestra@uswestmail.net

New Atlanta

www.newatlanta.com/

Cross-Database Portability with JDBC

WRITTEN BY SESH VENUGOPAL

Java programs can use the JDBC API to access relational databases, thereby cleanly separating the database system from the application. This approach holds the promise of cross-database portability, i.e., "write once, run on any database." In practice, several stumbling blocks stand in the way of fulfilling this promise.

This article shows examples of these stumbling blocks so you can see the common pattern of development. Drawing on this pattern, I'll develop a roadmap you can use to write applications that can work around these stumbling blocks. Finally, I'll demonstrate how the roadmap can be applied to build a portable application.

Breakdown of Portability

Three general situations in which the portability of code breaks down are using SQL escape syntax, translating external data types to database-specific types, and executing positioned updates on the rows of a table. These examples arise out of my experiments with writing JDBC-based applications using an organization schema with two tables, EMP and DEPT. These schemas were implemented in two different database systems, MS Access 97 and Oracle8 Personal Edition, both running on a Windows NT 4.0 workstation. The schemas for the respective database systems are shown in Figure 1. Note that the structure of the EMP and DEPT tables are the same in both databases, but the respective native data types of the columns are different, as shown in the box.

To drive the Access database, I used a Windows-supplied Microsoft Access driver and the JDBC-ODBC bridge from Sun (which came with JDK 1.1) as the JDBC driver of choice. For Oracle I used the JDBC OCI8 driver that supported JDBC 1.22.

SITUATION 1: SQL ESCAPE SYNTAX

The JDBC API is implemented in the java.sql package. A java.sql.Statement instance is used to execute simple SQL statements. JDBC specifies a so-called SQL escape syntax that may be used in Statement instances for various tasks such as pattern matching of strings, executing database-specific scalar functions, manipulating date and time values, calling stored procedures and executing outer joins. For each of these tasks JDBC specifies a syntax that's used by the developer, and the JDBC driver is responsible for translating this into database-specific code. Following is an outer join example that doesn't port.

• ***Outer join:*** Consider the data in the EMP and DEPT tables shown in Figure 2 (only the relevant columns are shown).

The following join operation would produce the result shown in Figure 3:

```
select dname, ename from dept d, emp e
where d.deptno = e.deptno
```

Note that the department OPERATIONS (DEPTNO = 40) doesn't appear in the resulting table because none of the entries in the EMP table has a DEPTNO value of 40. But what if you want to see every department represented in the result, even if it doesn't have any entries in EMP? You need to execute an outer join.

An outer join preserves unmatched rows in either the left table (left outer join) or the right table (right outer join). JDBC prescribes a SQL escape syntax for outer joins that looks like this:

```
{ oj outer-join}
```

where the keyword oj stands for outer join, and outer-join is of the form:

```
table left outer join { table | outer-join} on search-condition
```

If you want to preserve all the unmatched departments in the example, you can issue a left outer join by specifying the tables in order DEPT followed by EMP:

```
select dname, ename from
{ oj dept d left outer join emp e on d.deptno = e.deptno }
```

This works perfectly well with Access, but not with Oracle. The OCI8 driver doesn't support the oj escape syntax. Instead, you need to use the following equivalent SQL statement:

```
select dname, ename from dept d, emp e
where d.deptno = e..deptno (+)
```

in which the (+) at the end of the statement stands for outer join.

The result of this outer join is the table in Figure 4.

SITUATION 2: DATATYPE TRANSLATION

A large part of the work in making JDBC applications portable involves matching external data types to native database types (and vice versa). This matching is done in two steps: (1) the external type is matched to a JDBC SQL type (defined as a constant in the java.sql.Types class), and (2) the JDBC SQL type is translated by the JDBC driver to the native database type.

The next example shows how the table of a database can be populated. A naïve approach hard-codes datatype-specific information in the program, thereby making it nonportable. Following this I'll show a portable alternative that illustrates another stumbling block.

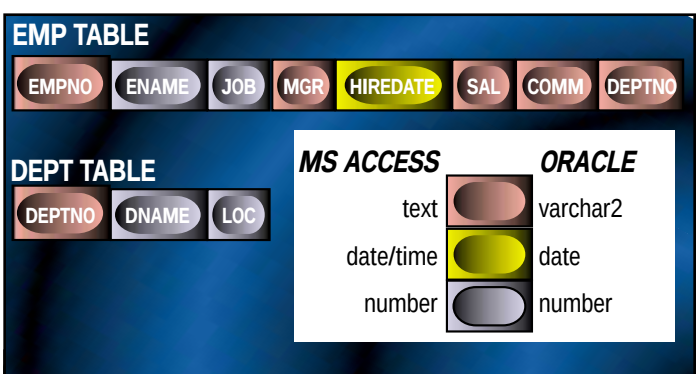


FIGURE 1 Organization schema for MS Access 97 and Oracle8

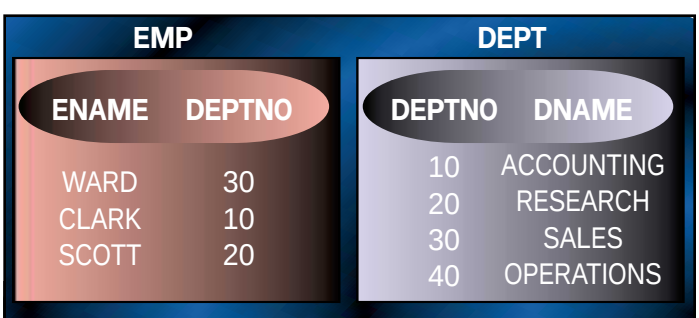


FIGURE 2 Sample data in EMP and DEPTNO tables

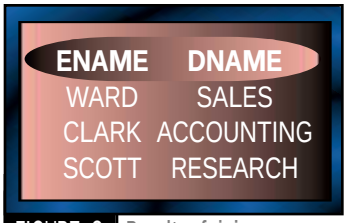


FIGURE 3 Result of join

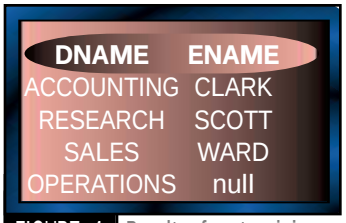


FIGURE 4 Result of outer join

- ***Example: Inserting data using PreparedStatement:*** A PreparedStatement instance is used to prepare and execute precompiled SQL statements. Suppose you want to insert rows of data in the table EMP. The pattern of the SQL statement to insert a row is always the same; it's only the values of the columns that change with every new row. To start with, you construct a PreparedStatement instance as follows:

```
PreparedStatement ps = conn.prepareStatement("insert into emp " +
       "(empno, ename, job, mgr, hiredate, sal, comm, deptno) " +
       "values (?, ?, ?, ?, ?, ?, ?, ?)";
```

The variable conn refers to the database connection. The PreparedStatement instance, ps, is parametrized with the values of the columns, indicated by the question marks. Each parameter is matched with a value before executing the prepared statement for some row. The following code segment shows how the empno column value is filled:

```
String next = st.nextToken();
 if (next.equals("null")) ps.setNull(1, Types.SMALLINT);
 else ps.setShort(1, (short)Integer.parseInt(next));
```

The first line reads an input token (using a StringTokenizer instance st to parse each input line) as a String object. The second line checks whether this token spells null, which indicates that the input does not have a value for this column. In this case a special setNull method is invoked on ps. The first argument to this method indicates the parameter position in the prepared statement (empno is the first); the second argument indicates the JDBC SQL type of the intended value. As mentioned earlier, the class java.sql.Types defines a set of constants corresponding to various SQL types, and the constant java.sql.Types.SMALLINT stands for the SQL type SMALLINT.

For every column in the EMP table, these three lines of code need to be implemented with appropriate modification in data types. There are a couple of serious drawbacks to this approach. One is that the program becomes ungainly and hard to maintain. Another is that all data types are exposed to the developer, so if a column type changes, or this program is ported to a different database, this code must be rewritten.

A better alternative postpones the datatype translation to runtime, thereby making the code portable. This alternative makes use of the PreparedStatement method setObject:

```
ps.setObject(1, next);
```

used instead of

```
ps.setShort(1, (short)Integer.parseInt(next));
```

The second argument, next, is a String instance that's automatically translated by the JDBC driver to the required database type, which in this case is a small integer. There is no coding of any datatype information. This statement can be set in a loop, using the loop index to control the first argument, which is the position of the parameter in the PreparedStatement instance.

If one or more of the column values is null, a little more work is required. The statement would then be written as:

```
ps.setNull(1, <SQL type>)
```

To maintain the datatype independence of the code to make it portable, the SQL type of the column value isn't coded directly here. Instead, it's discovered at runtime using the java.sql.DatabaseMetaData interface, and then plugged into the above statement.

The DatabaseMetaData interface provides metadata information for a database. Metadata is data that describes data. For instance, the EMP table contains employee information. This is data. Metadata would contain information on things like the number of columns in the table, the data types of these columns, whether a column can have null values, and so on. This is data about data.

The DatabaseMetaData interface provides methods that can be called to find out various metadata information about a database. For our example above, we need to find the data types of the columns of EMP so we can plug that information into the PreparedStatement setNull method call.

This alternative works with the MS Access database, but not with the Oracle database – the OCI8 driver refuses to pass the setObject invocation because it's unable to translate the String external type to the required database type.

SITUATION 3: POSITIONED UPDATE

When a SQL query is executed in JDBC, it returns a result set that represents the resulting table, consisting of a sequence of rows. A cursor is used to traverse the rows of a result set. The term *positioned update* refers to updating a database row referred to by the current position of the result set cursor. The following steps need to be taken to effect positioned updates from a result set:

1. Execute a SELECT FOR UPDATE statement. At the very least this will lock the rows of the table in the result set against other concurrent transactions. For this step to work, the database must support SELECT FOR UPDATE.
2. Get the cursor name of the result set. This will be used to reference the current row at the time of the update.
3. Construct a prepared statement to update a row, using the UPDATE…WHERE CURRENT OF <cursor name> form, with input parameters for the updated columns as well as for the cursor. For this to work, the database must support positioned update.
4. Traverse the result set, and for every row to be updated execute the prepared statement after setting all the input parameter values. The cursor name would refer to the row currently being referenced in the result set.

Now I'll show a pure JDBC code template using the above steps, written for full portability. This time around, the stumbling block to portability is even more severe.

- ***Example: JDBC template and Oracle specifics:*** Whether a database supports the required SELECT FOR UPDATE and UPDATE...WHERE statements described above can be discovered by using the DatabaseMetaData interface. Assuming a database does in fact support the required functionality, the code in Listing 1 serves as a template for updating all employee names in the EMP table to lower case.

MS Access doesn't support the required database functionality for positioned update. Oracle supports it, but the OCI8 driver doesn't implement the JDBC specification of cursor name. Instead, a completely different solution is adopted. The driver provides a ROWID, which is equivalent to the cursor name. A ROWID is added as a pseudocolumn to a query:

```
select name, rowid from emp
```

It may be retrieved using the ResultSet getString method:

```
String rowid = rset.getString(2);
```

It may be set as a parameter using the PreparedStatement setString method:

```
ps.setString(2, rowid)
```

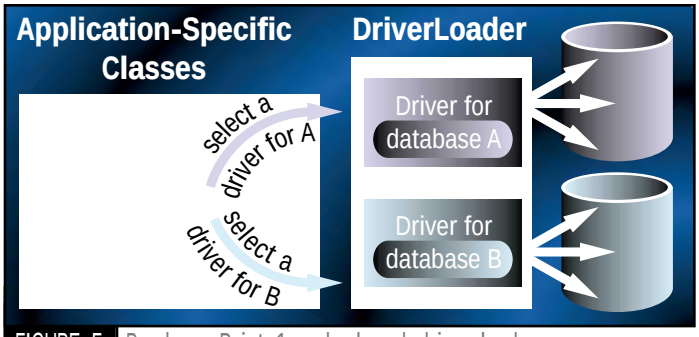


FIGURE 5 Roadmap Point 1 — back-end driver loader

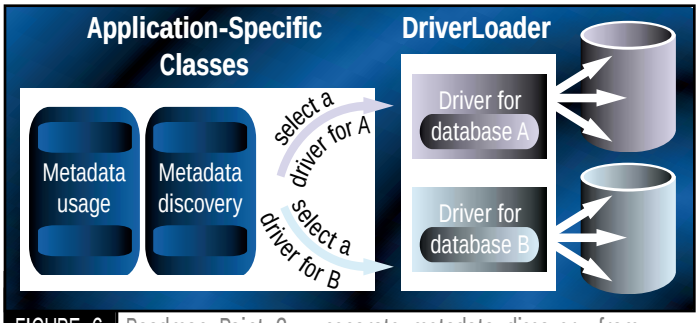


FIGURE 6 Roadmap Point 2 — separate metadata discovery from application-specific code

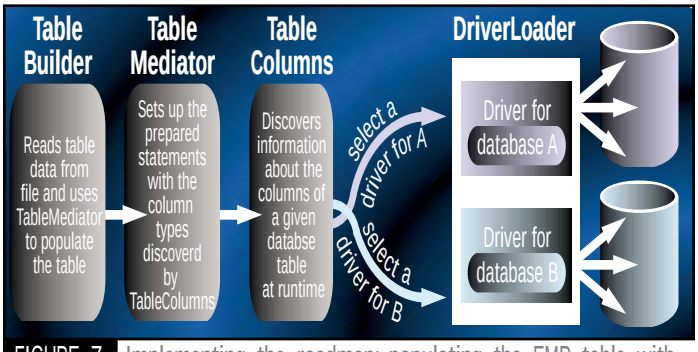


FIGURE 7 Implementing the roadmap: populating the EMP table with data from a text file

The JDBC-compliant template code shown in Listing 1 can be reworked for Oracle, using ROWID, at the cost of giving up portability.

Roadmap for Workarounds

The examples given here point to a common factor that contributes to the stumbling blocks: the JDBC driver. A JDBC driver implements the JDBC specification for a specific database system. For any given database system there is generally a wide choice of drivers available, including those from the database system vendor as well as third-party vendors. These drivers may differ in various respects, especially in the degree to which they implement the JDBC specification. Choosing an inappropriate driver can force the developer to write database-specific code in the application, thereby giving up portability.

A related common issue that is apparent from the examples is that even if the driver is appropriate for the task at hand, some database-specific information may have to be factored into the application. A naïve approach to this, which is to hard-code the required information into the application, makes the application nonportable. However, it's often possible to discover this information at runtime instead by using the DatabaseMetaData interface, thereby maintaining the portability of the application.

These observations suggest a two-point roadmap to work around the stumbling blocks:

- ***Point 1:*** Implement a "back-end" class that can load any given driver at runtime from a list of candidate drivers. This can be used not only to choose among a set of drivers for a single database, but among drivers for different databases if the application is ported to a different database. Figure 5 illustrates this point.
- ***Point 2:*** Separate metadata discovery from other code by implementing one or more classes that can serve to discover metadata at runtime. Depending on what kind of metadata needs to be discovered, you could design a suite of discovery classes that could be loaded ("plugged in") at runtime as required. Figure 6 illustrates this point.

Implementing the Roadmap

I'll now use the roadmap to sketch the process of building an application that populates the EMP table with data from a text file. Recall that a row of data may be inserted into the table by using a PreparedStatement instance, which in turn requires that the type of each column of EMP be discovered at runtime using the DatabaseMetaData interface.

Following Point 1 of the roadmap, the DriverLoader class is implemented, which loads either the OCI8 driver for Oracle8 or the JDBC-ODBC bridge for MS Access, as required at runtime. Other drivers for the existing databases, as well as drivers for other databases, may be added as needed.

Following Point 2 of the roadmap, the application is divided into three classes. One of these, TableColumns, implements the metadata discovery process. In this case it's the discovery of column types for the table EMP. Another class, TableMediator, uses this metadata and interacts with a third class, TableBuilder, that reads data from the input text file and sends it down the chain of classes to the database.

This chain of classes is illustrated in Figure 7.

Conclusion

The JDBC driver is the most critical piece in any Java database application. The driver must be picked with care, taking into account the architecture of the application, the extent to which the driver implements the JDBC specification, and the performance of the driver for various connection and database access operations.

I picked this specific set of three stumbling blocks for illustration simply because they provide a window into very different ways in which JDBC may be used in a database application. These examples and the workarounds demonstrated point to a general way of structuring a database application for cross-database portability. Specific refinements to this general approach can be adopted based on special requirements of the applications and the architecture; there's a lot of room for maneuvering within the proposed roadmap.

Author Bio

Sesh Venugopal holds a Ph.D. in computer science from Rutgers University. He runs his own IT and education consulting company, Intecus, Inc. (www.intecus.com), specializing in Web-based systems using the Java platform. Sesh is the author of a textbook, Data Structures: An Object-Oriented Approach with Java, *also online at www.intecus.com/bookpage.html.*

sesh@intecus.com

Listing 1: Pure JDBC Template for Positioned Update

```
ResultSet rset =  stmt.executeQuery("select ename from emp
for update");
   String cursor = rset.getCursorName();
   PreparedStatement ps = conn.prepareStatement("update emp
set ename = ? " +

"where current of ?");
   while (rset.next()) {
         String ename = rset.getString(1);
         ps.getString(1, ename.toLowerCase());
         ps.setString(2, cursor);
         ps.executeUpdate();
```

Elixir

www.elixirtech.com/

Parallel Worlds

Why Java and XML technologies will succeed

WRITTEN BY
SIMON PHIPPS

In the last few years the focus in computing has gradually moved away from the raw technology to settle on the total cost of ownership (tco) for a solution. What makes up the tco? That's hard to say, and everyone has a different answer, which usually depends on what they find easiest to fix. Most people agree that the tco isn't simply the sum of the prices of the parts that make the system, although it comes from those initially. A much greater cost arises from the cost of supporting the system in context.

A popular approach to reducing tco has been to try to centralize the administration of individual systems and/or the client desktop, yet that's only part of the answer. It's good to keep the amount of travel to a minimum, but what actually causes the administration to be needed? The answer, of course, is change, although not on its own. Change in isolation would only necessitate work on the change itself. But we all know that making a change in one part of a system results in support needs throughout the system.

The typical computer system is often heading toward "entropy death," in which ordered simplicity has tended toward interconnected complexity. While a cure for the symptom may be central administration, the actual disease mandates avoidance of the complex network of dependencies in the first place. It's this that Java technology and XML start to address, by eliminating the automatic codependency of systems, software and data.

A New World

The need for much of the support and administration comes from the web of dependencies woven by the software in our computers. To bring back the simplicity, we need to cut the dependencies. Where are they? There are several categories:

- *Software to platform*
- *Software to data*
- *Software to software*
- *Platform to platform*

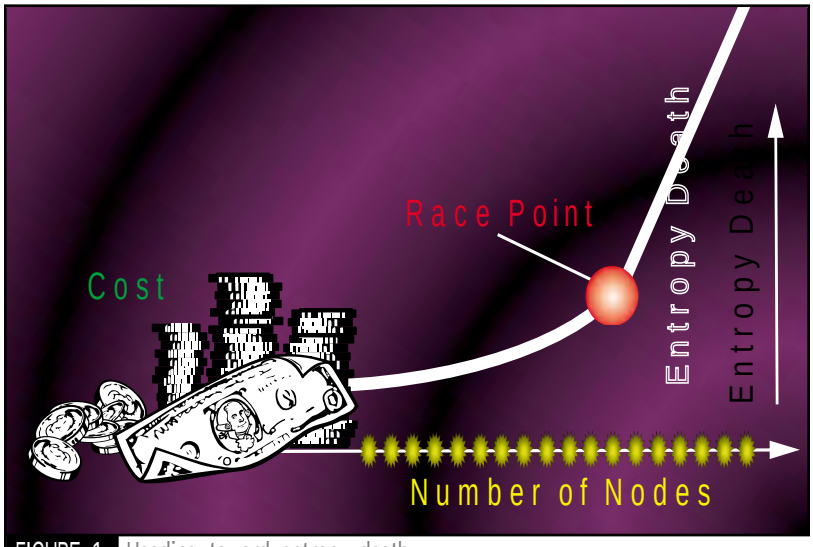


FIGURE 1 Heading toward entropy death

Cutting the cord of these dependencies isn't easy, but the new world of computing that's been developing over the last decade is finally coming to maturity and making it possible.

Let's first consider the computing model we've been living with. When computing was new, the choices were easy to make. I could pick any one from the limited range of computers, write software to run on it and create file formats to store the data in. Trouble was, the software and data would work only on that kind of computer, so when a different kind was used I had to use different software, or if I used different software on the same system I couldn't use the same data and had to learn a new user interface.

Many of the problems were solved by two standardization steps: everyone agreed to use the IBM PC and everyone used DOS and then Windows. A degree of simplicity came back. As time went on, though, it became clear that there was still plenty of scope for complexity to creep in. In particular, agreeing on the platform didn't break the platform dependency of the software – it just meant it was all codependent. And when an update came along, everything broke! In addition, there was no standardization beyond the power of monopoly in the world of the data. Just as the software depended on a particular level of the platform, so the data related to a particular level of a particular brand of software. A complex web of dependency was woven, in which a change at any point led to instability and perhaps failure in the whole web.

Codependency

The greatest enemy of computing is the creation of unintentional codependencies. As computer solutions are built, they involve relationships among software, hardware, platforms, development tools, and so on. Each is connected to every other by unseen connecting threads of codependency. Over time, the cost of owning any solution is proportional to the number of dependencies among the parts. But by the unintentional creation of many codependencies, the cost rises in an exponential rather than a linear way. The result is that the addition of further codependent elements increases the lifetime cost disproportionately. The point at which this begins to apply is the *race point*, and the condition beyond the race point is termed *entropy death* (see Figure 1). The inevitability of entropy death is set well before the race point by the act of choosing a system philosophy prone to codependency, the unwitting reliance of one part of a system on another, possibly mediated by some other element. The most common unwitting codependency is between software and the operating system it predicates.

This isn't to say that all codependencies can or should be avoided; some are inevitable. But in modern system specification and design they should be identified and justified in the same way as any other cost driver, taking into account not only the direct cost but also the lifetime cost inherited by connection to the dependency network. In general, software needs to be insulated from the environment in which it is used. In some situations use of native interfaces and binaries is unavoidable, but in these cases a platform-neutral "wrapper" around the native code is almost always valuable.

For example, consider the apocryphal case of a company that's used the macro language of an office suite as the basis for an office automation system. One day, installation of another piece of software (unknowingly) updates one of the DLL files used by the suite. Result? One of the macros no longer works. They finally manage to get it working again, but the new version needs an updated version of the spreadsheet program. To get that they have to install a whole new level of the office suite. Now none of the macros work! They crawl through them, updating and fixing, but among the other things the fixes mandate is a new version of the database driver. Sadly, that needs the latest version of the database to work. So they upgrade the database and…well, you can guess the rest.

The New Foundation

The problem is caused by the transmission of the impact of change from subsystem to subsystem. The integrated computing foundation currently in use in most systems acts as a transmission medium, allowing change in one place to have an impact elsewhere.

How can we escape this trap? The key is to disconnect data from software from platforms, to use standards-based choices so that version-version variations of the implementation have the smallest effect possible. By doing this we isolate changes from the transmission medium (the underlying platform) and prevent the impact of change from causing shockwaves of cost – we add the insulating layer mentioned earlier. What would be an optimal base of standards? The technology domains (see Figure 2) such a foundation (see Figure 3) would have to cover are:

- Network protocols holding systems together and providing access
- Delivery model that brings the solution to the audience that needs it
- Programming model by which the solutions are created
- Data structuring model for the information the solutions consume
- Security model that allows the right audience access to the right data and solution

Much of the change in the computer industry over the last decade has involved the rediscovery of technology ideas and their establishment as standards within that model. The mappings are:

- ***Network:*** TCP/IP, which has now become so widespread that it's no longer a topic of conversation.

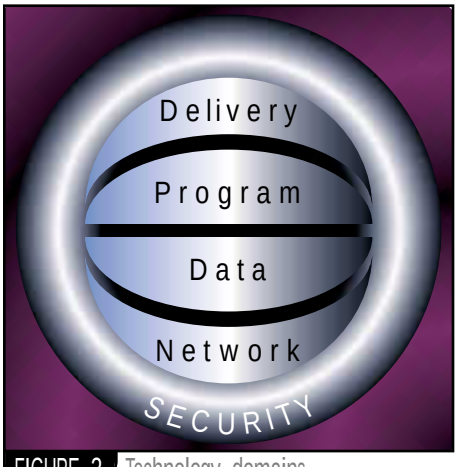


FIGURE 2 Technology domains

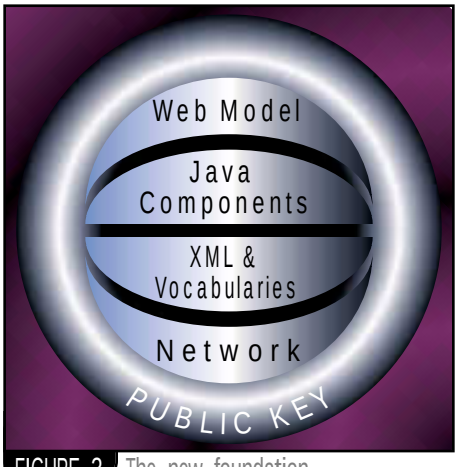


FIGURE 3 The new foundation

Author Bio

Simon Phipps, IBM's chief Java and XML evangelist, was part of the team that recommended Java techology to IBM in 1995. Since then he has spoken internationally on e-business foundation technologies and now has oversight of XML marketing in IBM worldwide. With over 20 years' experience in the industry, Simon has worked on networking, data communications and operating systems for various companies in many contexts, including development of the earliest commercial collaborative conferencing software.

- ***Delivery:*** Web model stateless client/server computing is the chosen delivery mechanism of a growing majority of business computer users. Rather than creating stateful clients that need costly maintenance and support, state is maintained instead at the server and "loaned" to the client.
- ***Program:*** Only four years from release, Java technology has established itself as the standard for new software in a vast number of enterprises, not least because its JavaBeans architecture allows component-based development to be used in earnest. This isn't to say that all code needs to be written in the Java language; it's platform-neutral Java bytecode binary programs that win. Where these aren't feasible, at the very least a wrapper of Java technology to insulate the rest of the solution from native code is essential.
- ***Data:*** Apparently new to the scene, XML is actually simplified SGML – 80% of the function for 20% of the complexity. Uptake throughout the computer industry has been huge, and it shows every sign of dominating data formatting in the future.
- ***Security:*** By removing the need to send full key information "in the clear," public key-based security systems are already dominant, especially on the Web.

From Technologies to Audiences

Alongside the agreement of the standards for the new world of computing has been a shift in the requirements for business solutions. In the past each solution would be built with only the requesting customer in mind. The focus was on who was using the solution and where they were; hence terms like *intranet*, *extranet* and *Internet*. But progress has meant that the focus now is much more on modeling the data and defining the relationship of the user to the data. There has been an inversion in the approach to computing solutions, and the focus has switched from technologies and systems to information and audiences. Today, defining a new solution involves defining the relationship that an audience has with a body of information. In most cases a given body of information will have multiple audiences. Thus, for an online shop, when customers view information, only their particular data is accessible to them, and it's presented in a way to suit them; when customer service staff from the vendor view the same information, both the scope and the presentation differ. It's the transition to a solutions-and-audiences view that presents the greatest challenge in IT today. But users can proceed with confidence since all of the technologies in the "new" tradition are in fact mature and proven, so the transition is one of emphasis and strategy rather than a leap into unknown technology.

Parallel Worlds

The fact that all five of the foundation technologies are well understood also offers another benefit. For many users migration to the new world of e-business is something evolutionary rather than revolutionary. They can take the first steps without scrapping the investment they've already made. This new world is thus a parallel world rather than an alternate one.

So why, after all that, will Java technology and XML succeed? There are several reasons:

- ***Proven technology:*** All five segments of the "new" foundation are based on the oldest, best established ideas in the industry: TCP/IP, "dumb" terminals, virtual machines, markup languages, public key systems – all proved by the experience of decades.
- ***User driven:*** In the final analysis, the move to the new foundation is driven by the needs and desires of the marketplace rather than by the fiat of any one vendor or even of a consortium. As the costs of computer technology become more of a focus item, and those driven by the upgrade arms race

XML: KEEPING THE INFORMATION IN THE DATA

You might have expected this whole article to be about XML, but it's so simple that would have been impossible. Having its roots in SGML (itself the development of work carried out in IBM in the late '60s), XML is very simply the idea of using tags for formatting all data, not just text intended for display in a Web browser. Already you may insert pseudo-HTML in your e-mail, like this:

```
<joke> Why did the chicken
cross the road ?</joke>
<punchline> To get to the other
side </punchline>
```

When you do this, you're actually creating a fragment of XML. You've defined a small vocabulary from which the tags you've chosen are drawn (let's call it JokeML), you've made sure it's well formed (has closing tags to match all the opening tags) and, apart from a line to say where the definitions of the tags can be found (a Document Type Definition or DTD) and a line to say it's XML, you're already an expert! Anyone who understands what a joke is will understand this data (although there's no guarantee they'll find it funny), so instead of just being data, it's turned back into information.

As more and more vocabularies are defined, XML is being used for tagging data in all sorts of applications. It has the great benefit of being digestible by computers as well as humans – after all, something like

```
<dl>
<dt> Why did the chicken cross
the road?
<dd> To get to the other side
</dl>
```

might clearly be a joke to human eyes, but a computer would never guess, whereas the XML fragment would be accessible to any program that understood JokeML.

XML will be used in the future wherever there's data – not just on the Web, and certainly not just in Web browsers. By putting the framework of meaning back into the data, XML brings life to the data, and turning the World Wide Web into a worldwide database makes communication between businesses possible without all the pain of the EDI process.

To learn more about XML and related standards, technologies and techniques, start at the XML zone on IBM's developerWorks: www.ibm.com/xml/.

> The key issue that should occupy us is not how I can cut the cost of administration and support, but how I can reduce toward elimination the amount of admin support that's needed?

toward entropy death become more and more obvious, the demand for the new foundation becomes greater and greater.

- ***Vendor supported:*** All five technologies form the basis of almost all vendors' new solutions. Vendors choosing an alternate at any point increasingly discover the market questioning their choice and suspecting an attempt at proprietary lock-in.
- ***Vendor neutral:*** All five technologies are beyond the control of any one vendor, so investments are protected from the risks of vendor lock-in as well as the design choices of any one vendor starting an upgrade race. The only possible exception to this are Java technology and public key, and it's worth taking time to consider why neither is a problem in this context.
- ***Platform neutral:*** All five technologies are independent of each other and of the platforms on which they're used. Thus they can all be implemented anywhere, insulating the systems that depend on them from codependency.

Java Technology: Public Property?

Can a technology apparently developed and controlled by a single vendor be considered open? It all depends on the attitudes and actions of the vendor and the time scales involved. In the case of the five domains in the new computing foundation, control has passed from the originator to "the mind of the market." For example, although the core ideas of public key systems are owned by one company, the industry has been willing to base almost all encryption and digital signatures on that technology because of a combination of the power of the technology and the attitude of the owners of the core patents.

In the same way, Java technology has become public property that is currently protected by the owner of the core technology. A move to standards body control would, however, be very positive. Standards bodies work better as museums rather than as factories; the parts of Java technology that are clearly established, such as the bytecode specification and the language, should be moved to the control of a suitable standards body as soon as practical – the third quarter of 2000, when the core patents of public key technology pass into the open, would be a good target. As long as the move toward full, externally controlled standardization continues apace, there probably won't be a problem. What's more, that ownership is nowhere as firm as in the case of public key systems. If the whole industry chose to implement Java technology differently, there would be almost no recourse. But that doesn't happen, because any company seen to violate the value of Java technology is shunned by the market. The fact that the base of standardization in Java technology is actually the binary format of bytecodes rather than the language is of course a big help. Thus, if we feel safe basing key parts of the computing infrastructure on public key systems, there's all the more reason to feel safe using Java technology. However, we should feel concerned if ever the technology owner puts brand equity before technology. Asserting, for example, that Java will never be made an independently managed standard would be a gross betrayal of trust.

Conclusion

The key issue that should occupy us is not how I can cut the cost of administration and support, but how I can reduce toward elimination the amount of admin support that's needed. To reflect on this changed concept, and to progress from the notions that sometimes turn consideration of tco into tcp (total cost of purchase), we should perhaps use a modified term to express the issue at hand: *lifetime cost of ownership*. The core assertion of this article is that the primary decision factor for new computer systems should be the cost of owning the system over its entire life, lco – software, network, and client and server hardware, complete with development, deployment, administration, management of impact during lifecycle and migration to replacement systems at sundown. The core proposal of this article is that this factor be controlled by minimizing the network of the codependent complexity that these various elements create. To achieve this a change of system philosophy rather than an instant change of technology is proposed. By basing future developments on a firm foundation of standards, entropy death can be avoided. And this is the reason Java and XML technologies will succeed, cool though the technologies themselves may be!

sphipps@uk.ibm.com

A Generic Client/Server Architecture for Java

Simplifying inter-app communications

WRITTEN BY
GENE CALLAHAN AND ROB DODSON

We set out to build a generic framework for creating Java client/server relationships. Our hope was to encapsulate all of the messy details of the relationship, allowing developers writing a client or a server to focus just on their particular application. This would allow our team to swiftly create client/server relationships relying on robust, fully debugged classes to handle communications. We wanted a framework that was simple and easy to use, but flexible enough to handle multiple communications methods.

Motivation

It has generally been difficult to set up the communications for a new client/server architecture. The first hurdle is to decide which of various methods to use: custom-parsed data on a socket, serialization, Remote Method Invocation (RMI), JavaSpaces, a proprietary middleware package, and so forth.

The next issue to be tackled is connection management. This involves specifying connection details such as host names and port numbers, restarting the server and reconnecting clients, graceful disconnection, reporting errors, logging and more. Much of the time these complex problems are solved in a way that's tailored to the specific client/server pair under consideration.

We had several criteria for an ideal solution to our situation. First of all, we wanted to pass serialized objects rather than some format we had to parse ourselves. This would allow us to rely on Sun's developing and maintaining our parsing code. We would also have access to books, online documentation, newsgroup help and outside experts whenever we needed assistance.

We wanted to use Java's listener event model, as it would allow for a consistent event and communications model for a program. Using this familiar model would significantly decrease the time it took a user of our framework to become familiar with it.

Support for multiple lower-level communication mechanisms was also a requirement. We already knew that our first server had to connect to Oracle, to an existing socket-based price server and to RMI-based calculation servers. We could envision other connection types – such as telnet, HTTP or SMTP – coming down the pike.

We needed our architecture to be simple and robust. We wanted a narrow API that would be easy to understand and maintain. As much as possible of the connection and event dispatching code should be in library classes so it could be coded and debugged once, then reused many times.

We also want the application itself to have enough control so it could handle errors, logging and connection operations when it needed to. At other times it could ignore these issues and let the lower-level classes handle them. This would allow our architecture to flexibly extend to future projects.

Applicability

The architecture we came up with suits our purposes quite nicely. We're developing for an equity-trading firm, and frequently create new clients for use by our traders and new servers to supply the client programs with calculation engines, data caches, trading models and so on. Subsecond response to queries is a requirement in many of our applications, since trading opportunities can appear and disappear very quickly. Servers with extensive data caching and high-performance calculation servers are also a must in our environment. Communication between these servers and our clients must be speedy as well. And we need to develop new client/server applications in a hurry to take advantage of new market conditions.

Our design is not suitable for our highest throughput jobs. For instance, we wouldn't contemplate rewriting our master price server, which handles hundreds of price updates per second, using this architecture. The overhead of serializing and deserializing objects would be too high to meet these requirements.

It's also not appropriate for multilingual situations. By this we don't mean that the United Nations couldn't use it – only that it doesn't talk anything but Java. It's possible, of course, to write code in another language that serializes or deserializes a Java object. But the large investment necessary to do so would negate the advantages of this architecture.

We decided that the initial trial of our architecture would be to use it to create a client/server pair, which would give our traders up-to-the-second information on stock options. We'll refer to this project several times below.

Design

Bruce Eckel convinced us of the superiority of the Java 1.1 event model in his excellent book *Thinking in Java*. In the paradigmatic use of the model, event handlers don't need conditional branching to determine what to do with a particular event because they rely on the Java-type system to do so.

Because we weren't handling a GUI, however, our model primary dispatch isn't on the recipient of an event. In a GUI, dispatching on the recipient makes sense – creating a listener to get mouse events over a button, for instance. But in our situation the source for events at the client is always the server, and on the server it's one of the clients. A "switch" on this source would make little sense. Instead, we register listeners for particular types of incoming objects. The classes that flow between the client and server become message types as well as possibly being significant application elements in their own right. To express interest in a particular class of object, the server and the clients create listener classes and call a registration method in the library.

Error messages and protocol-level events (e.g., a new client has connected, or the server has gone down) are also represented as classes, and are received and handled in the same way as application-level objects.

Because of our use of the listener pattern, neither our library nor its users

need conditional code or large switch statements to process communications. Listeners are registered with the library and placed as a value in a hashtable, a value for which the class they receive is the key.

Our architecture permits asynchronous calling of the listeners. The server can send out objects the client hasn't requested, such as updates to an object the client is viewing, in the same manner in which it sends out requested objects.

Our architecture also supports a generic monitoring program that the server can tailor to its own needs. It's easy to create both GUI and command-line monitors. They include built-in statistics gathering on the number of each class sent and received, as well as connection information such as application name, hostname and so on. To allow any particular server to tailor the monitor to its needs, the server can pass a list of monitoring commands to the monitor (see CommandList below). The monitor then puts them in a menu for easy access. Examples of commands we've implemented include shutdown, kill clients, turn server statistics on and turn them off. The monitor then periodically sends out a request for status object and receives back information on server performance and statistics. Because these monitoring facilities are built into the Channel class, you can worry about application design and not about building monitoring protocols. You can monitor your application in its early stages and not have to hack in monitoring later, as too often occurs.

Implementation

The classes we created to implement this architecture are as follows.

- ***Channel:*** There's one Channel object per logical service provided by a server. For example, all clients of our option server clients connect to it on the same channel. The Channel is the main class through which applications connect, read and write objects, handle errors and disconnect. The Channel therefore provides methods such as connect(), send() and goodbye(). There's no read() method because all objects are received through the ReceiveListener interface. This interface has a single method called recv(ChannelMsg). For an application to receive objects it simply implements a ReceiveListener and then calls Channel.registerListener(Class,ReceiveListener). The first argument is the class type of the objects that will be passed to the listener. Example:

```
public class FooListener implements
ReceiveListener
{
 public void recv(ChannelMsg msg)
 {
   Foo f = (Foo)msg.getData();
   System.out.println("Got a Foo: " + f.toString());
 }
}

channel.registerListener(Foo.class,
new FooListener());
```

- ***Device:*** Device is an abstract class. There's one Device per connected client, and therefore many per Channel. For example, the Channel in the option server has multiple Devices, each of which represents a connection to a different client. Creating new lower-level mechanisms is simply a matter of implementing new Device classes. Specific applications don't care about how the communications are carried out – they just send and receive objects. A descendant of the Device class could even implement a disk-file–based communication mechanism. The Channel calls device methods to do the actual sending and receiving of objects, as well as the connection/disconnection tasks. The Device class uses the common open/close/read/write model. Therefore, the most important Device methods are:

```
public abstract void open() throws IOExcep-
tion;
public abstract void close();
public abstract void write(Object o) throws
IOException;
public abstract Object read() throws IOEx-
ception, ClassNotFoundException;
```

  The Device abstract class doesn't define a constructor – the constructor will probably be different for each concrete Device, allowing parameters for that particular transport method. For example, the SockDevice uses sockets as its mechanism. Therefore the constructor takes hostname and port number parameters. (Figure 1 shows the relationship of Channel and Device objects.)
- ***SockDevice:*** This is the first concrete implementation of Device we created. It handles TCP/IP sockets as a transport mechanism. It uses the Java Socket classes and simply reads and writes serialized objects to and from the stream associated with the socket:

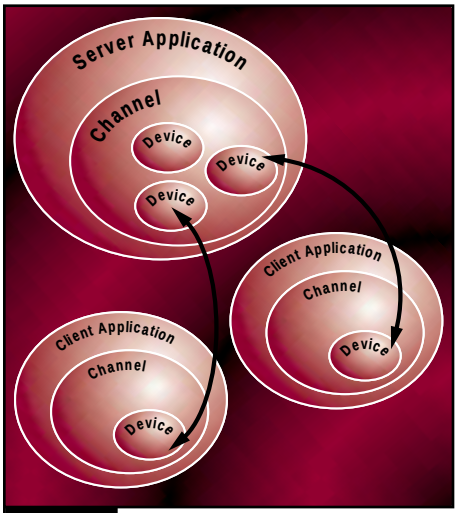


FIGURE 1 Application diagram

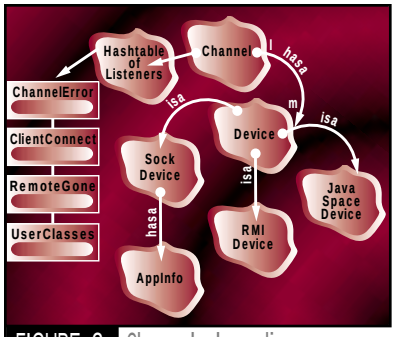


FIGURE 2 Channel class diagram

```
out = getWriter(sock.getOutput-
Stream());
out.writeObject(o);
out.flush();
out.reset();
```

The reset() call is included because serialized objects are cached by Java. If you send the same object twice, Java won't serialize it a second time. This is problematic if you change some fields in the class between the first and second call. The second call will use the cached class and your changes won't be sent. In our Device class the call to the reset() method can be toggled off or on. If you're sending the same class and aren't changing it, it will be more efficient not to call reset().

If some parts of an object intended for serialization are useful only to the server or client, they can be declared transient and never instantiated on the side that isn't interested in them. For example, the server may have a database connection for some objects, allowing it to fill in their fields. Since they're filled in before moving to the client, the client doesn't need the connection. Declare it transient, and you get the behavior you want.

EVENT CLASSES

A small number of simple event classes for managing and monitoring connections are built into our library. When certain events occur (e.g., a new client connects to a server), and if the application has supplied a listener, the application's listener will be called when the event occurs. Some of those classes are:

- ***ClientConnect:*** This is sent to a server application whenever a new client connects to it. The class contains an AppInfo object that lists information about the connecting client.
- ***RemoteGone:*** This is sent to the application (client or server) whenever a remote connection disappears.
- ***ServerConnected:*** Received by Channel clients when they have successfully connected to a server.
- ***Error:*** Sent into the application on any kind of general error. Contains error message and exception information.
- ***Goodbye:*** This is sent by a client to a server in an attempt to disconnect gracefully. If a client disconnects and the server doesn't receive a Goodbye object, it assumes the client has crashed and emits an Error object.
- ***AppInfo:*** Objects of this class are exchanged when a client and server first connect. It contains information about the sender such as hostname and user name. It may also contain device-specific info in a general info field. For instance, the SockDevice stores the port number there.
- ***CommandList:*** A server may send a CommandList to a monitoring client. This allows the monitor the ability to control the server.
- ***ChannelMsg:*** A ChannelMsg is passed to the recv() method of all ReceiveListeners. It provides access to the object that was sent. It also has handles to the Channel in use and the current Device over which the object arrived. These are most often used to reply to the sender of the object. For example:

```
recv(ChannelMsg msg)
{
    msg.getChannel().send(msg.getDe-
vice(), "Hi");
}
```

See Figure 2 for a diagram of the foregoing classes.

A SMALL APPLICATION

The ease of setting up connections is shown in the small code sample in Listing 1.

A NOTE ON THREADING

The waitForClients() call in the example server code above spawns off a thread to listen for new clients. Normally the application shouldn't return from this call. When a new client connects, a thread for the new client is created. This new-client thread will be the one that calls the ReceiveListeners. Applications are responsible for synchronizing their domain-specific data where needed.

The client side of the Channel also uses a separate thread to listen for objects returned from the server. Listing 2 is an example client application that first waits for a server, then calls Channel.waitForMsgs(), which launches the receive object thread. The listing provides the code to send a message to the server as well as some code to reconnect if there are any errors.

Consequences

The result of our efforts has been an extremely simple, flexible architecture for Java client/server relationships. A simple client/server pair can be created in a matter of minutes. In the specific application for our trading floor we were able to create a sophisticated and useful new client/server pair, working from an existing stand-alone client, in a couple of weeks. Our architecture can handle hundreds of thousands of price updates in a day. Our server caches stock and option information for several hundred securities and relays them to multiple clients in close to real time. A sign of the success of our design is that we've had to spend very little time on the classes described above once they were completed and debugged. This simple set of methods has provided the ability to create major applications with little attention to communications issues.

gcallah@erols.com robdodson@erols.com

AUTHOR BIOS

Gene Callahan, president of St. George Technologies, designs and implements Internet projects. He has written articles for several national and international industry publications.

Rob Dodson is a software developer who writes options-trading software in Java and C++ for OTA Limited Partnership. Previous projects include weather analysis software, tactical programs for Navy submarines and code for electronic shelf labels.

Listing 1

```
Channel channel = new Channel(new Sock-
Device(hostname, port));
public class MyErrorListener
{
 public void recv(ChannelMsg msg)
  {
   Error e = (Error)msg.getData();
   System.err.println(e.getErrorString());
   // do something with the error!
  }
}
Channel.registerListener(Error.class,
new MyErrorListener());
channel.waitForClients(true);
```

Listing 2

```
while (true)
{
 // Wait for a server
 while (channel.getCurrentDevice().isDe-
 viceOk() == false)
 {
  try
  {
   channel.waitForMsgs(false);
  }
  catch (Exception e)
  {
   Log.println("server not there");
   try { Thread.sleep(2000); } catch
      (Exception ee) {}
  }
 }
 try
 {
  channel.send(new Command("status"));
 }
 catch (Exception e)
 {
  // Problems? Go back and wait for server
     break;
 }
  if (channel.getCurrentDevice().isDe
  viceOk() == false) break;
}
```

HIGH-PERFORMANCE WEB APPLICATIONS USING THE JAVA SERVLET API AND JSPs

How to develop Web sites that can scale to a large number of hits

WRITTEN BY RUSLAN BELKIN & VISWANATH RAMACHANDRAN

There are several books and articles out there on dynamic-content generation technologies such as CGI, NSAPI, server-parsed HTML, server-side JavaScript, Active Server Pages and ColdFusion. Recently, Java Servlets and JavaServer Pages (JSPs) have emerged as a very popular technology and a lot of material has been written about them. Most of the articles focus on programming model features, ease-of-development issues and integration with tools. However, an increasing concern of Web site developers is to develop sites that can scale to a large number of hits. This is especially true of corporate software developers and those building the brand-name dot.com sites of tomorrow. This article focuses on how Web developers can design their server-side applications for scalability and performance using Java Servlets and JSPs.

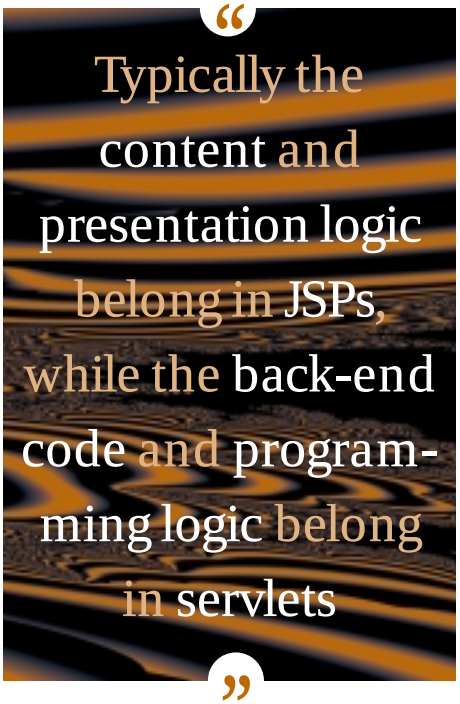
Java Servlets and JSPs

The Java Servlet API was designed to be a basic Web server extension mechanism in Java. It was modeled after NSAPI and ISAPI (which are the C/C++ extension mechanisms for the Netscape and Microsoft Web servers) and originally it was just the native plug-in API of the Java Web Server. It's a simple API that supports a request-response model, thus mirroring the nature of HTTP protocol. The simplest servlet is one that just returns a page saying "Hello world" to the client (see Listing 1). In the listing the servlet also checks for a form parameter called "name." If the name is supplied by the client, "Hello name" is printed instead. The functionality of the servlet is implemented by simply overriding the doGet method of HttpServlet.

In addition to request parameters, the servlet API can give the server plug-in developer the ability to manipulate most of the HTTP protocol-specific information in the server. This functionality is exposed through request and response objects (for HTTP request and response processing), and session objects (for handling HTTP sessions).

JSP technology was created to further simplify Web application development by separating the logic of the application from its content. This allows less sophisticated Web developers to take advantage of the latest Java technologies. JSPs provide developers with capabilities to embed the code logic directly into HTML pages. Another important feature is the ability to instantiate and use JavaBeans in a natural manner. This feature allows task partitioning between content developers and application logic developers. Listing 2 contains the same program written as a JSP.

A JSP is a document in a specialized markup language (with fragments of Java code); a servlet is a Java program. While there's overlap between the functionality of servlets and JSPs, typically the content and presentation logic belong in a JSP and the back-end code and programming logic belong in servlets.



How to Measure Performance

In order to design high-performance, scalable Web sites, it's essential to first understand the metrics by which they're measured. This section focuses on various metrics used to measure the performance of server software or a Web site. Since we're focusing on Web servers, the discussion is specific to the HTTP protocol.

- ***Latency:*** *The amount of time the Web server takes to process a request.* This is an important metric, as you don't want customers of your Web site to wait too long for a request to be processed, especially when they're likely to hit the "Reload" button to further increase the load on the site. It's important to keep both the average and the maximum latencies as low as possible. If you see high latencies with low CPU utilization, you may want to investigate if there's a lock contention somewhere in your Web applications. Typically measured in milliseconds.
- ***Throughput:*** *The quantity of data the Web server running a particular application is able to push through per unit of time.* This is important, especially for dynamic content applications that are generating rich content (including multimedia content). Typically measured in kilobytes per second.
- ***Connection rate:*** *The number of new connections the Web server is capable of accepting per unit of time.* This metric gives an idea about how well your Web site will scale to large numbers of client accesses. The maximum feasible connection rate is usually measured by increasing the load until average latencies rise considerably and/or CPU utilization reaches 100%. Typically measured in connections per second.
- ***Request rate:*** *Number of HTTP requests the Web server is capable of processing per unit of time, given any reasonable mix of client behavior (HTTP 1.0, HTTP 1.1, pipelined requests, keep-alive requests).* Note that this is different from the connection rate because multiple HTTP requests may be served on the same connection. However, the measurement technique is similar to the connection rate.

While latency and throughput are performance metrics, connection rate and request rate are metrics of scalability. To judge your Web site's performance and scalability, it's important to use anecdotal evidence of speed, but it's also important to measure scientifically, based on typical client accesses.

Session Management

HTTP protocol, as a model of communication between the client (a Web browser) and the server, is stateless in principle. While this makes it easy to implement, it certainly makes it harder for a meaningful Web application that preserves state or session information to be developed. A typical example of a session-aware Web application is a shopping cart in which to store selected items for checkout at some point in the future.

The need to maintain reliable session information for Web applications actually sparked a whole new class of software called *Web application servers* (e.g., Netscape Application Server, NetDynamics, WebLogic). These products, in addition to their legacy-integration capabilities, introduced a whole host of features that reliably enabled maintenance of state on the server side. However, the use of server-side session management facilities may cause degradation in scalability because:

- There may be lock contention while synchronizing on session objects.
- Maintaining the remote state across multiple server processes may introduce further overhead due to interprocess communication.

Don't use server-side state or session objects unless you absolutely have to. In many cases all the information can be transmitted between the client and the server via cookies or URL rewriting. This is especially true of nontransactional applications in which the reliability and integrity of the session is not as critical. Listing 3 gives code fragments where request parameters may be used instead of server-side state management to maintain the client's session.

Application Partitioning

With the advent of multitier programming models on the server, it's typical for Web sites to be architected with most of the programming logic residing in Web and application servers while the data resides in databases. A typical architecture is for Web servers to talk HTTP with clients, with an application server being the middle tier that talks to the database. The Web and application server usually run in different process spaces, and communicate using a proprietary or standard protocol. The Web and application servers are often distributed across multiple boxes.

In light of this, it's critical to partition your Web application in an appropriate manner. An ideal practice is to place stateless or light-state objects (which don't need to carry a reliable state) onto the front-end Web server, while keeping more complex objects on the application server of your choice. This often translates into keeping static HTML, content files and JSPs on the Web server, and transactional applications on the application server. For intermediate-sized applications there are trade-offs to keeping them on either the Web or application server. By

	HTTP 1.1 (WITH NO PIPELINING)		HTTP/1.1 (UP TO 5 PIPELINED REQUESTS)	
	Request Rate (per second)	Average Latency (ms)	Request Rate (per second)	Average Latency (ms)
HelloWorldServlet	589	25.4	1048	15
WASPServlet	247	54.1	334	53

Note: These numbers don't represent an offical benchmark or any other claim and may not be referenced in any way other than as an illustration in the context of this article. Hardware and software setup: Sun SPARC Ultra II 296MHz dual-CPU, Solaris 2.6, Exact VM JDK 1.2.1_03, iPlanet Web Server 4.0, WebBench Windows NT clients were used to drive the load.

TABLE 1 Effect of HTTP Pipelining

implementing such a partitioning, the content, presentation and simple applications of your Web site enjoy low latency and high request rates (by virtue of running on the Web server), while the more critical applications enjoy reliable state maintenance and transactional semantics (by virtue of running on the application server).

Custom Session Management

Since most nontrivial Web applications implemented using servlets and JSP will make heavy use of HTTP sessions, it's critical to select a platform that implements them efficiently. The most common implementation provided by Web servers today is in-memory session managers, which are very fast but may not provide the reliability or scalability desired for a high-end Web site. Other implementation strategies such as database-resident or shared-memory session managers may be considered. Often the best session manager will depend on the characteristics of the Web application being developed.

It's important therefore to choose an application deployment platform that enables the use of custom session managers through a plug-in API. When we were designing iPlanet Web Server 4.0 (formerly Netscape Enterprise Server), we decided to provide and support a session manager API within the servlet engine. We believe this will provide advanced users and third-party vendors with the capability to create their own custom session management solutions.

When to use custom session management? Consider, for example, when you already have a very efficient and proven architecture deployed to store and retrieve data efficiently across the network: the plug-in API for session manager will enable you to use this existing back-end solution to effectively store and retrieve session information just by implementing a few helper classes.

HTTP Protocol and JVM Issues

When choosing the appropriate platform for Web applications, one should always keep in mind how well HTTP protocol itself is implemented and integrated with the rest of the system. Since HTTP protocol is the way browsers speak to Web servers, the choice of the Web server platform is very important. The following features of HTTP 1.1 increase the overall performance of the system tremendously, and it's critical to select a Web server that implements them well.

- ***Pipelining:*** Modern browsers are capable of sending requests without waiting for the response from the server for each one of them. As you can see from Table 1, HelloWorldServlet has almost double the request rate when both the client and server support HTTP pipelining. The latency also decreases about 40%. The performance difference with a complex example, WASPServlet, that exercises most of the Servlet API isn't as drastic but is still noticeable.
- ***Chunked encoding:*** This is the HTTP 1.1 protocol feature that enables Web servers to send responses to the clients in chunks while keeping the connection alive. Consider, for example, a server-parsed HTML file (SHTML), which calls a servlet and a CGI. We expect the SHTML file to be preparsed by the Web server and cached. The servlet and CGI will be executed for each request. Since the servlet and CGI don't know about each other, they may both attempt to set the Content-Length header. To maximize the performance, the Web server could buffer the output and adjust the Content-Length header appropriately, or use chunked encoding so the connection between the browser and the Web server remains open.

One of the reasons we chose an in-process architecture for the servlet engine in iPlanet Web Server 4.0 was to be able to take advantage of the advanced HTTP 1.1 features of the Web server, such as pipelining and chunked encoding.

It's also important to select a JVM that has optimal performance characteristics. Ideally, your Web server allows you to plug in any JVM of your choice, and its plug-in layer is designed so as not to introduce additional overheads. For example, iPlanet Web Server 4.0 allows any JNI-compliant JVM/JDK/JRE to be plugged in (JDK 1.1 or Java 2). It also employs a number of advanced techniques to minimize the effect of the JVM on the rest of the system. For example, you can allocate a separate thread pool to run Java applications. Request threads allocated from that pool will be attached to the JVM only once, thereby decreasing the overall overhead. On the other hand, when the garbage collector decides to suspend these threads, it will have no effect on the rest of the system.

Programming Practices

Perhaps the most overlooked aspect of developing high-performance plug-ins is the need to write good Java code. Some suggestions:

- Try to scope all your references carefully so the garbage collector can take care of them easily (i.e., don't generate excessive garbage if you don't need to).
- Pool expensive resources (threads, database connections, etc.) instead of trying to construct them on every request. Listing 4 shows a code fragment in which an expensive object is managed by creating a pool for it in the servlet init method rather than by creating a new one on every request.
- Be extremely careful with synchronized methods as they may result in a lock contention. Try to make critical sections as short as possible.

You may consider using third-party or your own locking mechanisms to avoid inefficiencies of Java language (such as an absence of reader-writer locks) or Java libraries (for example, many Java collection classes, such as java.util.Hashtable, are inefficient, even in modern JVMs).

- Keep in mind that garbage collection can have an adverse effect on all threads that the JVM knows about (they can be arbitrarily suspended by the garbage collector). This can change the timing characteristics of your application.

Conclusions

Web site developers today have a wide variety of programming languages, models and tools to select from. While designing their Web applications for high performance and scalability, they need to keep in mind the following points:

- Measure the performance using well-understood metrics such as latency and request rate.
- Avoid server-based state management for nontransactional parts of the application.
- Partition the application across the Web and application servers, keeping the transactional components on the application server.
- Use a custom session manager if necessary.
- Deploy on a Web server that supports the HTTP 1.1 protocol correctly as well as the use of modern JVMs.
- Write good Java code, use pooling of expensive resources and avoid excessive synchronization and poor scoping practices.

Author Bio

Ruslan Belkin, lead engineer and architect of Java Servlet and JSP support in iPlanet Web Server 4.0 (formerly Netscape Enterprise Server), has over 10 years of experience in the industry and is a member of the Servlet API expert group. Ruslan has worked on Java, CORBA, distributed objects, component models and scripting languages, with special focus on high-performance implementations of standards.

Viswanath Ramachandran is a researcher working on JSP and JavaScript support in the iPlanet Web server group. A member of the JSP expert group, he holds a Ph.D. in computer science from Brown University in the field of programming languages. Current research interests include JavaScript performance, compilers and Web protocols.

ruslan@netscape.com vishy@netscape.com

Listing 1: HelloWorldServlet.java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class HelloWorldServlet extends HttpServlet {

public void doGet (HttpServletRequest req, HttpServletRe-
sponse res)
 throws ServletException, IOException
    {
      res.setContentType("text/html");
      String name = req.getParameter ("name");
      If (name == null)
         name = "world";

      PrintWriter out = res.getWriter();
      out.print("<html>");
      out.print("<head><title>Hello World</title></head>");
      out.print("<body>");
      out.print("<h1>Hello " + name + "</h1>");
      out.print("</body></html>");
    }
}
```

Listing 2: HelloWorld.jsp

```
<html>
<head><title>Hello World</title></head>
<body>
<% String name = request.getParameter ("name");
if (name == null)
     name = "world";
%>
<h1>Hello <%= name %> </h1>
</body>
<html>
```

Listing 3: ServerSessionServlet.java and ClientParametersServlet.java

```
 public class ServerSessionServlet extends HttpServlet {

 public void doGet (HttpServletRequest req, HttpServletRe-
 sponse res)
  throws ServletException, IOException
     {
       HttpSession session = req.getSession ();
       ……

       UserInfo info = search4user (session.getValue
       ("username"));
     }
 }

 public class ClientParametersServlet extends HttpServlet {

public void doGet (HttpServletRequest req, HttpServletRe-
sponse res)
 throws ServletException, IOException
     {
       String username = req.getParameter ("username");

       UserInfo info = search4user (username);
     }
}
```

Listing 4: Pooling of resources (UnpooledServlet.java and PooledServlet.java)

```
public class UnpooledServlet extends HttpServlet {

public void doGet (HttpServletRequest req, HttpServletRe-
sponse res)
 throws ServletException, IOException
     {
       res.setContentType("text/html");
       ….
       ExpensiveObject o = new ExpensiveObject ( …. );
       ….
     }
}

public class PooledServlet extends HttpServlet {

public void init (ServletConfig config)
     {
       super.init (config);
       expensiveObjectPool = new ExpensiveObjectPool();
       for (int i = 0; i < 10; i++)
        expensiveObjectPool.add (new ExpensiveObject ( ….));
     }

public void doGet (HttpServletRequest req, HttpServletRe-
sponse res)
 throws ServletException, IOException
     {
       res.setContentType("text/html");
       ….
       ExpensiveObject o = expensiveObjectPool.get ();
       ...
       expensiveObjectPool.recycle (o);
     }
}
```

JDJ Record Circulation

www.sys-con.com

JDJ Record Circulation

www.sys-con.com

JDJ Record Circulation

www.sys-con.com

JDJ Record Circulation

www.sys-con.com

Building the NEW ENTERPRISE

JAVA DEVELOPER'S JOURNAL ANNOUNCES...

JavaCON 2000

CONFERENCE
September 24-27, 2000

EXHIBITION
September 25-26, 2000

SANTA CLARA CONVENTION CENTER
Santa Clara, California

Join 100% Java Developers

Over 2,500 of your peers will be at JavaCon 2000, along with the industry's most respected technical experts, soughtafter gurus and advanced users who will show you how to maximize Java for the enterprise. Make 2000 your year! Dedicate yourself to 4 days of Java intellect and master new skills from those who are defining Java's future.

WWW.JAVACON2000.COM

Java Developer's Journal
Puts a Twist on Today's Hottest Topics

JavaCon 2000 is your only opportunity to learn from the experts at *Java Developer's Journal* – who best combine technical expertise and practical vendor know-how.

Benefit by Attending

- *The tips and techniques you'll learn will help you do your job better.*
- *Discover new applications being developed today that you will need tomorrow.*
- *Sessions are designed for the users at all levels with special sessions just for gurus.*
- *Network with fellow software developers as well as recognized Java experts.*
- *You'll learn how Java is being used for large-scale enterprise applications.*

Two-Day Exhibit

A full-scale exhibit hall packed with leading vendors will be on hand to demonstrate the latest products and answer your questions. All are welcome to fun-filled networking opportunities.

Exhibit Hours:
Monday, September 25, 12:00 – 7:30
and Tuesday, September 26, 12:00 – 5:30

Are You a Java Guru?

Why Not Join Our Faculty?

If you're anxious to share your unique Java experience, please send an email to stewart@camelot-com.com for details on how you may join the faculty. Topics of Interest Include but Are Not Restricted to:

- Java in the Enterprise
- Embedded Java
- Java Success Stories/Real World Java
- Advanced Java Application Development
- XML & Java
- Java in the Industry/ Java Business Applications
- Server-Side Java
- Java Testing and Debugging
- Object Oriented Concepts and Design with Java
- Java and Open Source
- Component Software and Frameworks

Call for papers
Deadline: April 17, 2000
Acceptance: May 15, 2000

Take A Look at What You'll Learn:

- Building Mission Critical Applications with Java
- Advanced JFC/SWING Component Integration
- Real-Time Java
- Designing High Performance e-Commerce Systems with EJB
- Combining XML and Java
- Java 2 Enterprise Edition
- Java Advanced Programming
- Java Security
- Java Runtime Internals
- Building a Multithreaded Server in Java
- Methods for Effective Java Unit Testing
- Developing COM and MTS Components in Java
- Developing Multi-tier Applications Using the Servlet API
- Object Oriented Analysis and Design with Java
- Dynamic Bytecode Generation with Java
- JDBC Technology
- Developing Large Scale Applications with Java and CORBA
- 2D and 3D Graphics in Java
- Designing Java Business Applications
- Java and Legacy Systems
- Using the Java Naming and Directory Interface API
- Java Gaming
- Programming for Devices (J2EE)
- Programming for the Desktop (J2SE)
- Java Commerce
- Using Java Agents
- Jini and JavaSpaces
- Java Testing and Debugging
- Java Exception Handling
- Garbage Collection Techniques
- Using Java with UML
- Writing Consumer Applications Using Personal Java API
- Smart Card Application Development
- Database Integration with Java

SPONSORED BY: JAVA DEVELOPER'S JOURNAL

PRODUCED BY: CAMELOT COMMUNICATIONS

JDJ NEWS

Sun Halts Java Standards Efforts

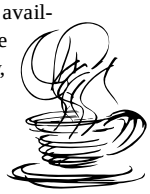
(*New York, NY*) – Sun Microsystems, Inc., announced at December's Java Business Conference in New York City that it is halting its efforts to make Java an industry standard.

Making the Java language an industry standard would give other companies, such as its partner, International Business Machines Corp., a stronger position in defining the future direction of the software.

This past spring Sun submitted Java to ECMA, a European standards body, after it withdrew from the International Standards Organization, another standards body. Sun said it is withdrawing its proposal to ECMA due to similar issues. www.sun.com

Sun Offers J2SE for Free

(*New York, NY*) – At the Java Business Conference Sun also announced the availability of its core Java technology, the Java 2 Platform, Standard Edition, at no charge. Since the release of the first version of the Java technology in January 1996, Sun has licensed the Java platform source code, including the Java programming language, runtime compiler and class libraries, for various fees and royalties.



Beginning January 31, the source code for J2SE will be offered without charge. The binary runtime environment will continue to be offered free as well. www.sun.com

ComponentSource Launches World's First EJB Software Components onto the WWW

(*Marietta, GA*) – ComponentSource, a leading Internet-based distributor of software components and related services, announced an agreement to supply the world's first-ever Enterprise JavaBeans globally over the Net.

The components, developed by The Theory Center (recently acquired by BEA Systems Inc.), are based on the EJB 1.1 standard and offer core functionality to save corporations from having to understand the complexities of building e-business solutions from scratch. www.componentsource.com

Tower Technology Unveils TowerJ 3.5

(*Austin, TX*) – TowerJ 3.5, which includes support for the Java 2 specification, will enter beta testing this month and be generally available during the first quarter.

Java developers will be able to deploy dynamic server apps based on the Java 2 spec while significantly improving performance. TowerJ 3.5 includes enhanced support for servlets, EJB and JavaServer Pages. www.towerj.com

SourceGuard 4.0 Shipped by 4thpass

(*Seattle, WA*) – 4thpass has begun shipping the next version of its deployment tool for the Java platform. New features include Code Pruning, which allows developers to trim applets and cut their download time in half, and the addition of custom packaging for embedded applications, which enables Java software vendors to customize the deployment of their embedded applications and reduce the overall cost of memory-constrained Internet-enabled appliances. www.4thpass.com

Geek Cruises Departing November 2000

(*Palo Alto, CA*) – Geeks and other computer professionals are invited to attend Java Jam, a seven-day cruise through the western Caribbean seas that combines the fun of a luxury cruise with Java instruction.

Designed for intermediate and advanced Java programmers, courses and seminars such as "Tips for the Java Wizards," "Developing Mission Critical Applications Using JDBC" and "Jini Architecture Overview" will be taught by expert instructors. They include Jon Meyer (coauthor of the book *Java Virtual Machine* and research scientist at NYU's Media Research Laboratory), Bill Day (consumer devices software designer) and Ian Darwin (UNIX and Java developer as well as author of three books and nearly 60 magazine articles).

Courses range in length from two-day and one-day to half-day formats, and are scheduled for times when the ship is in transit.

Java Jam is a way for programmers to learn new skills, and to network and relax. Companies can use these cruises as bonuses or incentives for outstanding employees. "Employers can offer a great vacation to employees, while ensuring that they will learn new skills in the process. Everyone benefits!" according to Neil Bauman, president of Geek Cruises. There's also plenty of entertainment for non-geek companions.

Java Jam departs from Fort Lauderdale, Florida, in November 2000. Port stops include Half Moon Bay, Bahamas, Georgetown, Grand Cayman and Cozumel, Mexico. The $575 program fee includes all seminars, materials and a bon voyage cocktail party. Stateroom prices start at $999 per person. www.geekcruises.com

And the Winners Are...

Best IDE: *JBuilder 3* (Inprise)

Best Java Application Server: *Weblogic 4.5* (BEA Systems), *SilverStream Enterprise Applications Server* (SilverStream Software), *PowerTier EJB 5.0* (Persistence Software), *Dynamo 4.5* (ATG)

Best JavaBean: *JFC Suite v2* (ProtoView Corporation), *GLG Toolkit* (Generic Logic Inc.), *JClass Enterprise Suite* (KL Group)

Best Java Database Product: *PointBase* (PointBase, Inc.), *Secant Extreme Persistence Object Server* (Secant Technologies)

Best Installation Tool: *InstallShield Java Edition* (InstallShield), *InstallAnywhere 2.5* (ZeroG Software)

Best Java Modeling Tool: *GD Pro 4.0* (Advanced Software Technologies, Inc.), *Together/J* (TogetherSoft), *PowerDesigner 7.0* (Sybase), *Rational Rose 98i* (Rational Software)

Best Java Book: *Enterprise JavaBeans* by Richard Monson-Haefel

Best Java Profiling Tool: *JProbe Profiler and Memory Debugger* (KL Group), *Optimizeit 3.0* (Intuitive Systems)

Best Java Testing Tool: *WebLoad* (RadView), *WinRunner* (Mercury Interactive), *SilkTest* (Segue Software)

Most Innovative Java Products of the Year: *AlphaWorks* (IBM), *JRun* (Allaire Corporation), *Jasmine ii* (Computer Associates), *Enterprise Reports 3.0* (EnterpriseSoft)

JDJ NEWS

Introducing TAME Version 4.0

(*Chicago, IL*) – VirtualSellers.com Inc. has released TAME version 4.0, which uses an interpretive language that is UNIX and Windows NT compatible. By reducing bandwidth requirements, developers can save up to 50% in coding time compared to Microsoft Active Server Pages. In addition, XML is embedded in the language, reducing the time required to develop tags.

VirtualSellers.com's new TAME programming language, which includes a dynamic Web page engine, provides Web access to databases, giving developers the ability to create solutions that can be deployed on all operating systems and server environments, resolving the common problems associated with many of today's non-XML browsers.

www.virtualsellers.com

Aether Systems Releases AIM 2.0

(*Owings Mills, MD*) – Aether Systems, Inc., has released an upgraded version of its wireless data software, Aether Intelligent Messaging, version 2.0. The newest version of AIM includes application programming interfaces for Java and Microsoft COM environments, in addition to AIM's C/C++ programming environment. AIM also now supports Solaris and Windows NT back-end servers.

During the first quarter a special developer toolkit for AIM 2.0 will be available for download from an Aether "Developer Zone" on the Aether Web site.

www.aethersystems.com

Gen-it for Java Code Generator Now Shipping

(*Montreal, Quebec*) – Codagen Technologies Corp. announces the availability of Gen-it for Java code generator. Codagen's patent-pending technology reduces the time associated with the coding and maintenance of software applications by enabling developers to make changes within the modeling tool or at the generator instructions level rather than through tedious and error-prone manual coding. Gen-it for Java is architected to work with the industry's leading UML modeling tools. It's available for download from the Codagen Web site at an introductory price of $4,900US per user. www.codagen.com

PointBase Introduces Release 3

(*Mountain View, CA*) – PointBase, Inc., announces new products and capabilities for extending, managing and integrating data across the Internet. With Release 3 PointBase adds its new UniSync option for synchronizing data across multiple database systems. The company has also announced its Embedded Server database product for demanding multiconnection embedded applications and its Device Edition for managing SQL data on memory-constrained devices such as PDAs and Internet appliances.

www.pointbase.com

Pramati Launches Server and Studio for J2EE

(*Hyderabad, India*) – Pramati Technologies, Ltd., announces the release of its Server and Studio based on the Java 2 Platform, Enterprise Edition.

Apart from compliance with the J2EE specification, Pramati intends to differentiate its server in three key ways: scalability and performance, manageability and monitoring, and tools for the J2EE Power user.

www.pramati.com

Sun's Withdrawal from ECMA May Not Signal End of World

Java is already an industry standard. Numerous surveys attest to the existence of over a million developers using standard Java. Those million developers rely on Sun maintaining Java as a standard so that [it] won't become splintered. This is one of the major appeals of Java. Because Sun has developed, maintained and protected the Java standard, the million-plus programmers can rely on their Java software being portable to many platforms. Sun's legal battles with Microsoft over the 100% Pure Java standard was very significant. Microsoft was attempting to destandardize Java and start a splintering process [that would] weaken and potentially destroy Java. Fortunately for us all, Microsoft did not succeed.

As with most successful industry standards (e.g., SQL, C and FORTRAN), the standards are established at an industry level first and then eventually canonized through a public standards process such as ANSI or ISO. In Sun's standardization effort they need to strike a balance among (a) making sure there is one standard and avoid a diverging set of standards, (b) involving the community in the process of developing the standard, and (c) protecting the standard from the threat of Microsoft. This is a difficult balance to achieve and hence this is one of the reasons you have seem some false starts in introducing Java into the public standardization domain. I believe over time Sun will achieve this difficult balance.

The reality is that Sun's Java community process is more open to the Java community than a public standards process would be. Typically speaking, in the public standards committees it's the large vendors who have the loudest voice. Each of these vendors has an agenda that is driven by their competitive issues and not necessarily for the overall good of the community. Those not represented by a large vendor aren't typically represented. I believe that those who are concerned about the public standardization process are a very vocal minority and it makes for interesting discussion, but the vast majority of Java programmers are quite happy that Sun has maintained the control over the Java standard to the degree necessary to avoid divergence. The last thing the Java community wants is a public standardization process that is prone to design by committee and disruption by Microsoft.

—Bruce Scott, CEO, PointBase, Inc.

My sense is that a solid majority of JavaLobby members and Java developers are either supportive of Sun's withdrawal from ECMA or they do not feel that de jure standardization is important enough to matter. It is unlikely, at this point, that any strong opposition to Sun's chosen course will emerge within the ranks of the developer community. Sun's stewardship of Java has satisfied many, and I can say from experience that Java developers are a demanding group. Sun's momentum is so strong that even powerful standards bodies will be unlikely to gain a foothold with any alternative standard specification.

—Rick Ross, Founder, JavaLobby

I'm saddened by this, but not overly concerned. From my discussions with various industry experts during the Java Business Expo, I'm not surprised that Sun stopped the process, as it involved releasing copyright for parts of Java, something they've never been willing to do. I think the basic core pieces of Java that were to be standardized are in fact already a de facto standard, so this was largely an empty exercise. The weight of many companies is solidly behind Java; this won't impact its momentum. In fact,I see something positive in this. It's been my experience that standards bodies often move at the same pace as glaciers. In a world where the whole business model can be turned on a dime in 18 months, a standards organization will never be able to adapt to a change. Having Sun in charge allows us all to benefit from the quick reaction time of a single entity. So I regard it as more of a minor curiosity and disappointment than any real damage to the Java community as a whole.

—Sean Rhody, Editor-in-Chief, Java Developer's Journal

ADVERTISING INDEX

ADVERTISER	URL	PH	PG
4TH PASS	WWW.4THPASS.COM	877.484.7277	29
AMERICAN CYBERNETICS	WWW.SOFTEXPORT.COM	800.899.0100	27
APACHE 2000	WWW.APACHECON.COM	650.404.9944	89
APPLIED REASONING	WWW.APPLIEDREASONING.COM	800.260.2772	35
BEA WEBLOGIC	WWW.WEBLOGIC.BEASYS.COM	800.817.4BEA	2
CAREER CENTRAL	WWW.CAREERCENTRAL.COM/JAVA	888.946.3822	28
CAREER OPPORTUNITY ADVERTISERS		800.846.7591	90-101
CERTIFY ON-LINE	WWW.CERTIFYONLINE.COM	877.JAVA YES	66
COMPUTERJOBS.COM	WWW.COMPUTERJOBS.COM		39
CONCENTRIC NETWORK	WWW.CONCENTRICHOST.NET	800.476.0196	77
DEVELOPMENTOR	WWW.DEVELOP.COM	800.699.1932	87
ELIXIR TECHNOLOGY	WWW.ELIXIRTECH.COM/DOWNLOAD/	65 532.4300	63
EMBARCADERO	WWW.EMBARCADERO.COM/ADMINISTER		65
EMBARCADERO	WWW.EMBARCADERO.COM/DESIGN		67
EMBARCADERO	WWW.EMBARCADERO.COM/DEVELOP		69
FIORANO SOFTWARE, INC.	WWW.FIORANO.COM	408.354.3210	45
IAM CONSULTING	WWW.IAMX.COM	212.580.2700	61
IBM	WWW.IBM.COM/DEVELOPERWORKS		13
JAVA DEVELOPER'S JOURNAL	WWW.JAVADEVELOPERSJOURNAL.COM	914.735.0300	79
JAVA DEVELOPER'S JOURNAL	WWW.JAVADEVELOPERSJOURNAL.COM	914.735.0300	79-82
JAVACON 2000	WWW.JAVACON2000.COM	914.735.0300	83
JAVELIN	WWW.JAVELINTECH.COM	612.630.1063	47
JDJ CONSULTING SERVICES	WWW.OPENJOBS@SYS-CON.COM		41
JDJ STORE	WWW.JDJSTORE.COM	888.303.JAVA	88
KL GROUP INC.	WWW.KLGROUP.COM/PERFORMANCE	888.328.9597	15
KL GROUP INC.	WWW.KLGROUP.COM/INTERFACE	888.328.9596	71
KL GROUP INC.	WWW.KLGROUP.COM/DEADLINE	888.328.9596	104
METAMATA, INC.	WWW.METAMATA.COM	510.796.0915	49
NEW ATLANTA	WWW.NEWATLANTA.COM/	678.366.3211	57
NUMEGA	WWW.COMPUWARE.COM/NUMEGA	800.4-NUMEGA	19
OBJECT DESIGN	WWW.OBJECTDESIGN.COM/JAVLIN	800.962.9620	52-53
OBJECTSWITCH CORPORATION	WWW.OBJECTSWITCH.COM/IDC35/	415.925.3460	55
OFFICE.COM		212.995.7742	47
PERSISTENCE	WWW.PERSISTENCE.COM		17
POINTBASE	WWW.POINTBASE.COM/JDJ	877.238.8798	25
PRAMATI	WWW.PRAMATI.COM/J2EE.HTM	914.876.3007	51
PROTOVIEW	WWW.PROTOVIEW.COM	800.231.8588	3
QUICKSTREAM SOFTWARE	WWW.QUICKSTREAM.COM	888.769.9898	20
RESPONSE SYSTEMS SERVICES, INC.		212.295.4305	41
RIVERTON SOFTWARE CORPORATION	WWW.RIVERTON.COM	781.229.0070	85
SEGUE SOFTWARE	WWW.SEGUE.COM/ADS/CORBA	800.287.1329	11
SIC CORPORATION	WWW.ACCESS21.CO.KR	822.227.398801	43
SILVERSTREAM SOFTWARE, INC.	WWW.SILVERSTREAM.COM	888.823.9700	103
SOFTWARE AG	WWW.SOFTWAREAG.COM/BOLERO	925.472.4900	21
SYBASE INC.	WWW.SYBASE.COM	800.8.SYBASE	31
SYMANTEC	WWW.VISUALCAFE.COM		4
TIDESTONE TECHNOLOGIES	WWW.TIDESTONE.COM	800.884.8665	37
TOGETHERSOFT LLC	WWW.TOGETHERSOFT.COM	919.772.9350	6
VISICOMP, INC.	WWW.VISICOMP.COM	831.335.1820	73
VISUALIZE INC.	WWW.VISUALIZEINC.COM	602.861.0999	87
VSI	WWW.VSI.COM/BREEZE	800.556.4VSI	75
YOUCENTRIC	WWW.YOUCENTRIC.COM/NOBRAINER	888.462.6703	33

Apachecon 2000

www.apachecon.com

Innovation Without Disruption

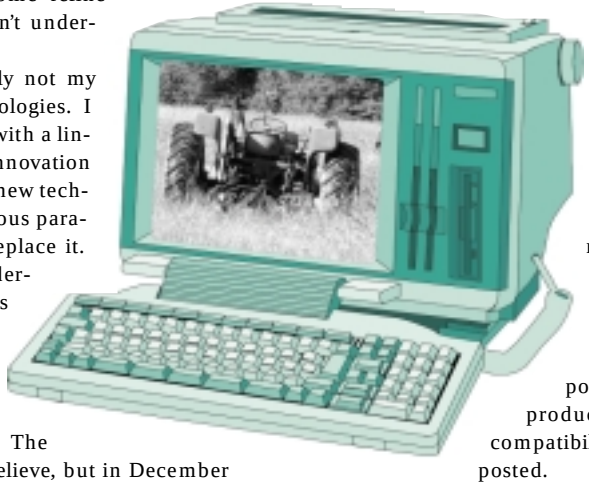
WRITTEN BY GEORGE PAOLINI

When I was a lad, I tell my kids, life was hard. We had to walk everywhere, for instance. And not only that, we had to carry our own data. On things called floppies. Back then, we had to manually move information from one computer to the next. We did all this work by hand.

When modems came along, we were sure we could do away with the floppies. Of course, the first modems I used transmitted at 300 baud – about the speed of a John Deere tractor in a cornfield. And then there were the protocol incompatibilities. On my first job, for instance, I spent days with a couple of black boxes and cumbersome acoustic couplers trying to get a Compugraphic in one office to wake up the Harris in another. (By the way, these things were real computers, green-screen, line-editing rigs, not the fancy turbo-graphics models of today.)

I never did get the machines to talk. Even dozens of calls to the two companies produced no workable solution. Apparently the Harris – the newer model – had made some "refinements" that the Compugraphic couldn't understand.

This was my first, but unfortunately not my last, brush with incompatible technologies. I spent all of the '80s and the early '90s with a lingering question: Why does the cost of innovation have to be disruption? It seemed every new technology couldn't just augment the previous paradigm, it had to rip it to shreds and replace it. (Not that I take it personally, you understand, but I still have emotional scars as a result of the transition from LPs to CDs.)



Then, just five years ago this month, I saw the light. Well, actually, I saw a prototype Java Web browser. The screen was singing, literally. Hard to believe, but in December of '94 the world was still pretty impressed with the ability to represent a static page of information uniformly on any computer. And here was executable content inside a Web browser. It was utilizing the Web protocols, not breaking them. Innovation without disruption.

The technology advancements the Java community has made in the ensuing years closing out this decade are nothing if not phenomenal, culminating with the release this month of Java 2, Enterprise Edition. We've gone from dancing Web pages to harvesting all that legacy business logic and extending it out to the network. So what's made it work? Well, to be sure, Sun has done its part. So have Oracle, IBM and about 200 licensees of the technology. And a million-plus Java programmers, such as you.

The key really has been one simple rule: innovate compatibly. Make any changes you want, but don't break anything in the process.

We formalized this rule in what we now call the Community Source License and the Java Community Process.

The Community Source Licensing program means access to the source code by simply clicking on a Web page. As easy as that click is, it comes with a responsibility: innovate, but remain compatible. Take the source code, use it for R&D purposes, modify it if you wish. Once you reach the stage of creating a product, you're required to pass compatibility testing to ensure that these modifications don't break anything.

Beyond the licensing model is the Java Community Process, a framework for evolving the Java platform that harnesses the intellectual capital of the entire Java community. Here's how it works.

Any company, organization or individual who has signed a Java Specification Participation Agreement (JSPA) can submit a request for new or additional functionality to the Java platform. If the request passes the first set of hurdles (a check to ensure that this functionality doesn't already exist or isn't underway as a project somewhere), an expert group is formed.

This group is responsible for writing the specification. The leader of this group, usually the individual who submitted the request, is responsible for developing consensus within the group as well as rendering tie-breaking decisions when consensus can't be reached.

After the spec has been drafted and reviewed by the 240 companies that have signed JSPAs, it is posted for public review and comment. Once those comments are reviewed and necessary changes are made, the draft is posted. Then the hard work begins – to produce a reference implementation and compatibility tests. When finalized, these are also posted.

That's the Java Community Process in a nutshell. Since the process was introduced, 41 requests have been submitted to augment the platform; 36 have been approved.

Innovation without disruption. Which leads me to wonder what stories of hardships you'll have to share with your kids….

AUTHOR BIO

George Paolini, vice president of marketing of Software Products and Platforms at Sun Microsystems, is responsible for managing all public relations; Internet, strategic and technical marketing; branding; and trade shows and events within that business unit. George is also a member of the Java Developer's Journal *Editorial Board.*

george.paolini@sun.com