WHAT IS JAVAMAIL?

JAVA™ DEVELOPER'S JOURNAL

The World's Leading Java Resource

Volume: 4 Issue: 10, October 1999

JAVADEVELOPERSJOURNAL.COM

Guaranteed Lowest Prices...

JDJStore.COM

SYS-CON PUBLICATIONS

Make the Internet more relevant and give your customers better—and faster—service
CREATING NEWSFEEDS USING JAVA APPLETS

BEA

www.weblogic.beasys.com

Protoview

www.protoview.com

Sun Microsystems

www.sun.com/service/suned/java2

SEAN RHODY, EDITOR-IN-CHIEF

Holy Wars

When I was a teenager, my parents taught me never to argue about sex, politics and religion. Later on I also learned that it's never a good idea to argue with drunks. Now I find myself in the unenviable position of having to step into the middle of a "religious" debate.

In the July issue of ***JDJ*** (Vol. 4, issue 7) we ran a feature story regarding the use of Java with DCOM. It touched off a great deal of debate, both pro and con, concerning the suitability of publishing this article in a magazine like ***Java Developer's Journal***.

Arguments against had a couple of themes. One main theme was anything that comes out of Redmond is bad, and there are Java-based alternatives to everything Microsoft. ***JDJ***, as the standard bearer for Java, should have nothing to do with the "Evil Empire." A second, somewhat more reasonable theme was that Microsoft won't be supporting Java 2, so their technology is going away.

On the pro side, a number of readers felt this article was right on the mark. They're working in an environment where interoperability comes before portability, and this article provided information they needed to get the job done.

Rather than arguing with either of these sides, I thought I'd simply state where ***JDJ*** stands on this issue and explain its position. As the editor-in-chief of ***Java Developer's Journal***, I look for a variety of articles. I look for high-level articles that discuss the principles of Java programming, technically detailed articles that explain useful Java techniques, and product-related ones that discuss the integration of Java with a particular tool.

That being said, ***Java Developer's Journal*** will not restrict the content of the magazine based on who creates the technology being discussed. We have regular columns devoted to what I think are the mainstream interests of the language, and they evolve as the times change and the language grows. I try to mix in a variety of topics while sticking to the editorial calendar we set every year. My policy is that ***JDJ*** will be inclusive rather than exclusive. When we plan the magazine, we clearly realize that every reader won't read every article. That's fine. Our goal is to reach a broad audience every month with enough variety and content that they continue to read.

There will be the occasional article regarding Microsoft technologies. But there won't be a regular column regarding Microsoft, nor will we be turning the magazine into Microsoft Java Journal. Our focus will remain, as always, on all Java technologies. If you're not interested in topics concerning Microsoft, flip past it to the next article.

I know this may set off another flurry of discussions. I think that's good – I prefer to see people express their opinions instead of just sitting back and staring at you. If you want to send me e-mail regarding this, please do so. And please keep reading.

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 specializes in application architecture – particularly distributed systems.

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
ASSISTANT 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

JIM CRAFTON, FRANK GATES, RACHEL GOLLUB, JOHN KEOGH, JIM MILBERY, IAN MORAES, SEAN RHODY, AJIT SAGAR, BRUCE SCOTT, JON SIEGEL, 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 DESIGNER:	ROBIN GROVES
GRAPHIC DESIGN INTERN:	AARATHI VENKATARAMAN
SYS-CON RADIO EDITOR:	CHAD SITLER
WEBMASTER:	ROBERT DIAMOND
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

NuMega

www.compuware.com/numega

Presenting Java

WRITTEN BY AJIT SAGAR

In the fast-changing world of Internet-based technologies, perception is everything. Is a business solution implemented in a particular technology truly cross-platform? Is it scalable? Is it robust? Is it easy to use? Does it do what it set out to do? Most times the answers to these questions are based on the perception of the functionality offered by the application. In a distributed application a large part of the burden of providing the perception falls on the designers of the user interface. One of Java's salient features – platform interoperability – is achieved via the perception of user interface portability.

I say "perception of portability" because seamless cross-platform portability is paradoxical by its very definition. Let's take a minute to consider what porting applications between platforms really means. Porting applications between platform A and platform B inherently implies that platforms are incompatible. What this boils down to is that the assembly code generated by an application on platform A will be different from the assembly code generated by the same application on platform B. The executable is different. The combination of hardware, operating system and compilers used by the application is different for the two execution environments. Indeed, it's this very difference that allows different vendors to tout their value-adds for the industry. And all this is good because it promotes healthy competition.

The Ultimate Thin Client

Sun Microsystems' "the network is the computer" message has been consistent right from the beginning. Since Java was introduced in the marketplace, Sun has touted it as the technology that will make this a reality. For the network to be the computer, the burden of installing and running the intelligent piece of software applications needs to shift application functionality to tiers that are different from the client user interface tier. In other words, the intelligence and complexity in a solution need to move out of the presentation tier and be buried in different tiers of a distributed application. The presentation tier should be concerned solely with presenting data served up by the next tier. The UI client should be as thin as possible.

The catch, however, is that the same application should be able to offer the same functionality via different clients. And the world of clients has morphed from dumb vt100 terminals to PCs sitting on every user's desk to PDAs, cell phones, set-top boxes and pagers. Java offers core technologies that support distributed networking like Jini as well as different flavors of the Java platform for distributing the application across various tiers of a distributed application. In theory, the same content can be presented across all these devices. As a result, porting the user interface across the various applications should be seamless. All the devices run Java. As long as the code is pure Java, we should be able to drag class files from a PC and drop them into a 3COM palm.

In reality, it's not as simple as that. Any portable code has to adhere to the least common denominator of features that can be provided across the different display units. This defeats the purpose of the different devices, each of which provides unique features in the way it presents the data. This uniqueness manifests itself in the user interface, which is the user's window into the device. The more realistic approach is to leverage the least common denominator of functionality as much as possible, while keeping in mind that parts of the user interface won't be portable across devices and platforms. The challenge is to decide how much of the functionality needs to be provided in the user interface. The thin-client model may still exist; however, the data served up by the next tier will be different for different devices.

Wait a minute....Does this mean I can keep my user interface device/platform dumb and drive the user interface from the other tiers of the distributed application? That would be neat. Well, as the saying goes, if it's too good to be true, it probably is. A big assumption here is that the features offered by a particular UI device or platform can be optimally leveraged by the "server." This is certainly not true. In fact, the different levels of sophistication of the user interfaces translate to the value-add provided by the vendor. The minute you start mixing and matching unique features offered by the device with the least common denominator offered by Java, guess what? Your client just gained weight.

Java Clients vs Web Clients

Today it behooves Java-based computing solutions to offer several types of clients. One is a Java client (fat client) that takes advantage of specific strengths of an application and presents them in the user interface. This is implemented as a Java application. The second type is a thinner client that uses Java applets to add dynamism to a Web UI on the client. A third type of client is the pure Web client that achieves dynamism via scripting languages. The latter two types have at least one thing in common: they serve up HTML (or XML), i.e., they run in browsers. One of the advantages of the Web clients is that they achieve some level of UI device portability. However, they inherit the issues related to browser portability. Hence, the burden of UI configuration shifts from Java code to scripting code. The point is that you still have to provide unique implementations to accommodate your environment's idiosyncrasies.

There's no silver bullet. Designing user interfaces is hard. Creating the right balance of responsibility across the different tiers of distributed applications is hard. However, as long as the appropriate perception is maintained, the end user should be unaware of these complexities. And abstracting complexity is the reason we use software to create business solutions.

ajit@sys-con.com

AUTHOR BIO

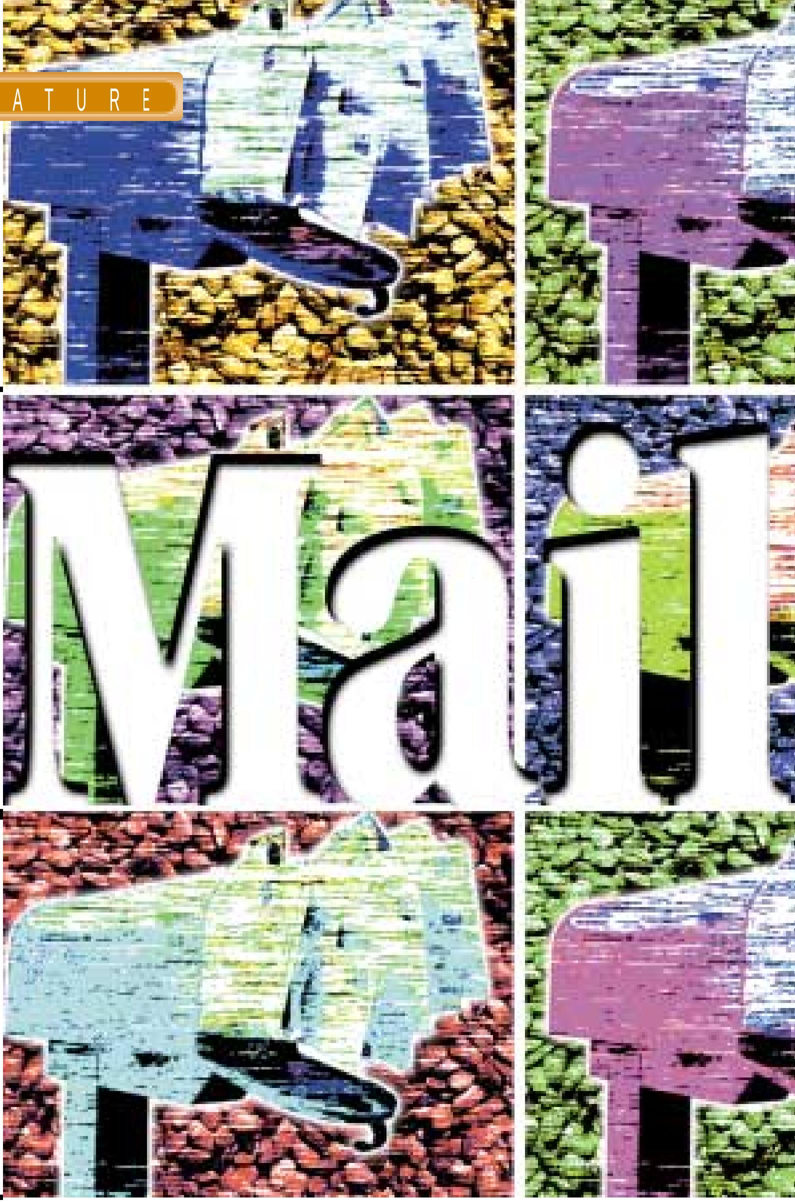
Ajit Sagar, a member of the technical staff at i2 Technologies in Dallas, Texas, holds an MS in computer science and a BS in electrical engineering. He focuses on Web-based e-commerce applications and architectures. Ajit is a Sun-certified Java programmer with nine years of programming experience, including two and a half in Java.

Soft
Wired

www.javamessaging.com/ibus

J D J   F E A T U R E

Framework for developing Internet-based e-mail client applications



WRITTEN BY IAN MORAES

E-mail functionality is an important system requirement in areas such as e-commerce, customer care, work-flow management and unified messaging. In addition, some application architectures may need to support not only standard mail protocols but also proprietary ones. If you're charged with the task of developing an e-mail client application in Java, you have a number of architectural and design options: building your own services layer to support multiple mail protocols, purchasing third-party components that provide e-mail features or using JavaSoft's JavaMail framework.

This article focuses on the JavaMail framework for developing Internet-based e-mail client applications. The JavaMail API shields application developers from implementation details of specific mail protocols by providing a layer of abstraction designed to support current e-mail standards and any future enhancements. JavaMail increases a developer's productivity by allowing the developer to focus on the business logic of an application rather than on mail protocol implementation. It provides a platform- and protocol-independent means of adding e-mail client features to your applications. JavaMail version 1.1 is a standard Java extension and requires JDK/JRE 1.1.x or higher.

Overview of Internet Mail Protocols

Before getting into the details of JavaMail, a quick overview of some messaging terms and Internet e-mail protocols (IMAP, POP, SMTP) is in order. A message is described in terms of a header and content. A message header comprises information such as sender (from), recipient (to), and message ID (a unique message identifier). A message's content is the actual message body and can comprise multiple parts in the form of text and attachments, as shown in Figure 1.

An e-mail client is used to transfer messages to and from an e-mail server, which is responsible for sending and receiving e-mail messages across the Internet. These servers store a user's messages either permanently or until retrieved by an e-mail client.

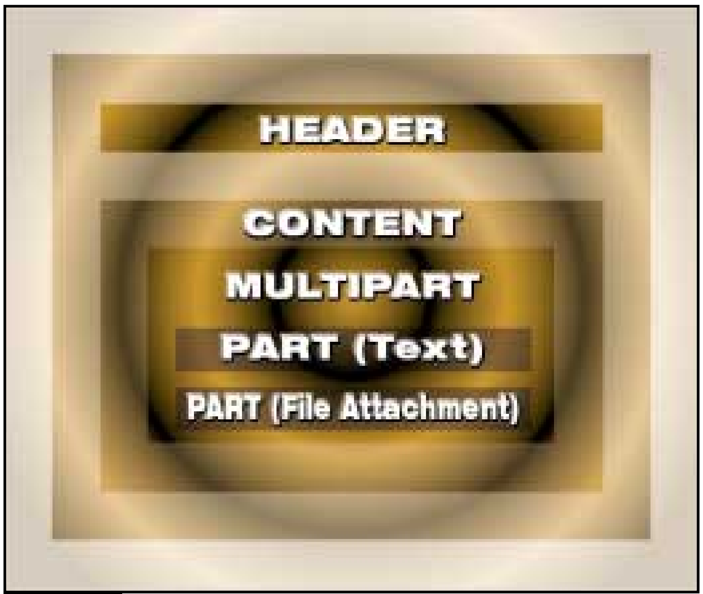


FIGURE 1 Structure of an e-mail message

To develop an e-mail client you need to deal with protocols for sending and receiving messages. As shown in Figure 2, the common protocol for sending Internet e-mail messages is SMTP (Simple Mail Transfer Protocol). The original SMTP specification limited messages to a certain line length and allowed only 7-bit ASCII characters. The MIME (Multipurpose Internet Mail Extensions) specification builds on SMTP by removing the maximum line length for messages and allowing new types of content (e.g., images, binary files) to be included in e-mail messages. The MIME specification overcomes these limitations by defining additional fields in a message header to describe new types of content and message structure. MIME defines the content-type header to specify the type and subtype of message content. For example, a message with an HTML attachment would have a content-type header set to “text/html”. SMTP and MIME are typically used together to send Internet e-mail messages.

Two types of protocols are used to retrieve messages from Internet mail servers: POP3 (Post Office Protocol 3) and IMAP4 (Internet Message Access Protocol 4). Although POP3 is more widely used than IMAP4, the latter has a number of advantages. First, IMAP supports multiple folders on a remote mail server whereas POP3 supports only the Inbox folder. Second, IMAP supports message status flags (e.g., indicates whether a message has been previously seen); POP3 doesn’t. These types of protocol features are important considerations in designing your application.

JavaMail provides implementations of SMTP, IMAP4 and POP3. For more information on these Internet mail protocols, you can consult the pertinent RFCs.

JavaMail Architecture

Now that you have a basic understanding of Internet mail terms, we can begin to discuss the JavaMail architecture. As shown in Figure 3, the architecture can be described in terms of three main layers. JavaMail’s layered architecture allows clients to use the JavaMail API with different message access protocols (POP3, IMAP4) and message transport protocols (SMTP).

The top layer is the application layer that uses the JavaMail API. The second layer is the JavaMail API that defines a set of abstract classes and interfaces for supporting e-mail client functionality. This is the layer that frees a developer from having to deal with protocol-specific complexities. JavaMail provides concrete subclasses of these abstract classes for Internet mail. The JavaMail API layer depends on concrete implementations of protocols.

The implementation layer forms the third layer of the JavaMail architecture. Since JavaMail is protocol independent, it’s up to service providers to implement specific message access and message transfer protocols. Service provider implementations play a role similar to that of JDBC drivers. A provider registry allows service providers to register their protocol implementations to be used by JavaMail APIs. The JavaMail Web site has more information on third-party service providers.

JavaBeans Activation Framework

JavaMail interacts with message content through an intermediate layer called the JavaBeans Activation Framework (JAF), part of the Glasgow specification (a future release of the JavaBeans component model specification). In terms of dealing with e-mail messages, JAF provides a uniform way of determining the type of a message’s content and encapsulating access to it. The JAF is implemented as a standard Java extension. Sun provides a royalty-free implementation of JAF that requires JDK 1.1.x or higher.

JAF is used to get and set a message’s text and attachments. JavaMail provides convenient methods to interact with JAF. For example, MimeMessage’s setText() method can be used to set a string as a message’s content with a MIME type of “text/plain.” Another example is MimeMessage’s getContent() method, which returns a message’s content as a Java object by invoking methods on JAF’s DataHandler class.

JAF can also be used to view e-mail attachments such as a text file (.txt) or an image (.gif) by instantiating a JavaBean that supports a particular command (e.g., view) on a specific type of message content. As

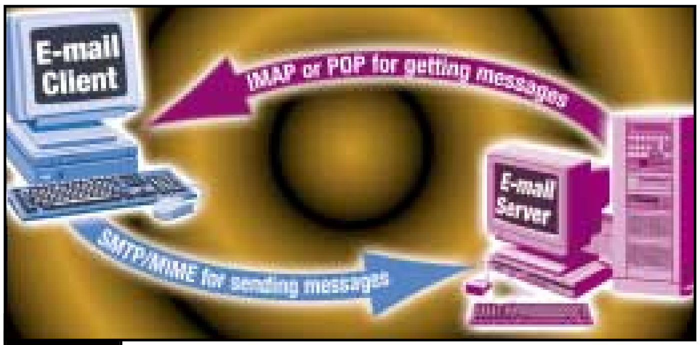

E-mail Client
IMAP or POP for getting messages
E-mail Server
SMTP/MIME for sending messages

FIGURE 2 Protocols for sending and getting e-mail messages

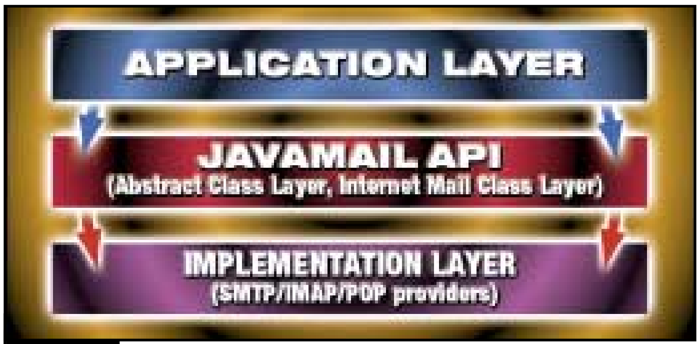

APPLICATION LAYER
JAVAMAIL API
(Abstract Class Layer, Internet Mail Class Layer)
IMPLEMENTATION LAYER
(SMTP/IMAP/POP providers)

FIGURE 3 Overview of JavaMail architecture

CLASS NAME	DESCRIPTION
SESSION	This javax.mail class represents a mail session and serves as the main entry point for the JavaMail API. The Session class controls access to the Store and Transport objects.
TRANSPORT	This javax.mail class represents a transport agent that sends a message to its recipients using a specific message transfer protocol. Transport is implemented by a service provider.
STORE	This javax.mail class defines a message store (a database and an access protocol) for a set of message folders. Folder objects are accessed through a Store object. Store is implemented by a service provider.
FOLDER	This javax.mail class comprises messages and subfolders in a treelike structure. It also defines methods to retrieve and delete messages. Message objects are accessed through a Folder object. Folder is implemented by a service provider.
MESSAGE	This javax.mail class models an e-mail message. A Message object interacts with its content using JAF. The javax.mail.internet.MimeMessage class extends Message to represent a MIME message. Message is implemented by a service provider.
ADDRESS	This javax.mail class models an e-mail address. The javax.mail.internet.InternetAddress class extends Address to support Internet e-mail addresses.
DATAHANDLER	Message content is represented as a javax.activation.DataHandler class that wraps around the actual data. The DataHandler class provides an interface to the JAF.

TABLE 1 Important classes and interfaces when using JavaMail

shown below, the JAF DataSource object, which encapsulates the attachment, is used to create a DataHandler object. The DataHandler object uses a CommandInfo object to retrieve the pertinent JavaBean that can be used to perform a specific operation on an attachment. The JavaBean component can then be added to a frame, as shown in the code snippet below. Currently, reference implementations of JAF-aware JavaBeans are available to view text, GIF and JPEG files. CommandInfo, DataSource and DataHandler are all JAF classes.

```
// file name represents the attachment
FileDataSource attachFds = new FileDataSource(attachmentFilename);
DataHandler dh = new DataHandler(attachFds);
CommandInfo viewCi = dh.getCommand("view");
Frame attachmentWindow = new Frame("View Attachment");
// add the bean to view the attachment to the main window
attachmentWindow.add((Component)dh.getBean(viewCi));
```

Examples Using JavaMail

You now have a basic overview of the JavaMail architecture and JAF so we can discuss the main JavaMail classes and methods needed to support an e-mail client. Table 1 describes some fundamental JavaMail classes.

To illustrate the use of JavaMail in an e-mail client application, I'll consider four major use cases: configuring a connection to e-mail servers, sending a message, getting messages from an e-mail server and deleting messages.

Configuring a Connection to an e-Mail Server

Before you can send or receive messages from your mail server, you need to establish a mail session between the mail client and the remote mail servers. Mail user properties are used to initiate a connection to mail servers. The Session class can manage the mail user properties used by the JavaMail API.

```
// Setting mail user properties
mailProperties = new Properties();
mailProperties.put("mail.transport.protocol", "smtp");
mailProperties.put("mail.smtp.host", "someSmtpHost");
```

The Session object is a factory for Store and Transport objects. A Session and Store object can be obtained as follows:

```
// Get a Session object
Session session = Session.getDefaultInstance(mailProperties, null);
// Get a Store object
Store store = session.getStore();
```

One issue to consider as you design the application architecture of your e-mail client is the dependency between your business layer and JavaMail. To reduce tight coupling between your application's business layer and the JavaMail subsystem, the Facade design pattern can be used. For example, mail user configuration can be passed into a Facade (singleton) to assemble the appropriate JavaMail objects (Session, Transport and Store) and perform any other initialization (e.g., security). As a result, dependencies between your business layer classes and the JavaMail subsystem are reduced, and your business layer can use a simpler interface such as MailFacade.configure(Properties p).

A use case pertaining to e-mail server connectivity is support for "disconnected e-mail operation," which involves maintaining e-mail message stores on both the remote server and a local client, performing operations on both stores, and then being able to synchronize these two stores. JavaSoft's IMAP provider implements interfaces that can be used to support disconnected operation.

At the time of this writing, secure messaging (e.g., support for S/MIME) is currently missing from JavaMail. S/MIME builds security on top of MIME to provide secure electronic messaging for Internet mail in the form of message authentication and data security. Although not available in JavaMail, it can be obtained from a third party. The JavaMail Web site has more information on this and other third-party Web sites.

Sending e-Mail Messages

Once a mail session has been established, an e-mail message can be created and sent. First, a MimeMessage object needs to be constructed. Then, as shown below, the message object is initialized with the recipient's e-mail address(es), the subject of the message and the message text. The message is then sent using the Transport object.

```
// Create new message
Message msg = new MimeMessage(session);
// Initialize the message
msg.setFrom(new InternetAddress(senderEmailAddress));
msg.setRecipients(Message.RecipientType.TO,InternetAddress.parse
  (recipientEmail,false));
msg.setSubject(subject);
msg.setText(messageText);
Transport.send(msg);
```

JavaMail can also be used for developing mail-enabled servlets (e.g., using a browser to send e-mail to a support center). An important architectural consideration is reuse of the mail functionality in your business layer by both Web and Java clients. This can be accomplished by using a layered architecture and appropriate design patterns. For example, a simple mail Facade that shields a client from the JavaMail API can be used by both a Java client and a servlet to send an e-mail messge; a client needs to provide the view and controller components (Model-View-Controller design pattern) of the application.

When sending messages, users assume that e-mail clients support address books, but JavaMail currently does not. However, the JavaMail API doesn't preclude you from developing your own mechanism (e.g., using XML – for implementing local address books or using JNDI to access an LDAP-enabled server for global address book features).

EnterpriseSoft

www.enterprisesoft.com

Getting e-Mail Messages

After you've successfully established a mail session, an e-mail client can retrieve your e-mail messages. To retrieve your messages use the Session object to obtain a Store object, which can be used to obtain the Inbox folder, as shown below.

```
 // Opening the Inbox Folder
 store = session.getStore();
 store.connect();
 Folder folder = store.getFolder("INBOX");
 folder.open(Folder.READ_WRITE);
```

“JavaMail is a relatively new framework that will inevitably continue to mature and evolve”

After a folder has been successfully opened, it's used to get message totals and the messages, as illustrated in the following code snippet.

```
 // Getting message totals and messages
from a folder
 int totalMessages = folder.getMessage-
Count();
 Message[] msgs = folder.getMessages();
```

Note that the Message objects returned from the getMessages() method call are designed to be lightweight objects. For example, the Message objects returned could comprise only message header details. Retrieval of the actual message content is deferred until the message content is actually used. Thus the getMessages() method isn't designed to be a resource-intensive operation.

A typical e-mail client first displays a summary of messages in the form of header details such as sender (from), recipients (to), subject and sent date. A summary line for each message can be obtained as follows.

```
 Address[] from = msgs[i].getFrom();
 Address[] to = msgs[i].getRecipients(Mes-
sage.RecipientType.TO;
 String subject = msgs[i].getSubject();
 Date d = msgs[i].getSentDate();
```

After viewing the message summary, a user typically decides to view the actual message content in the form of text and/or attachments. Now we're ready to get a message's content, which can be retrieved in the form of an object. The retrieved object depends on the type of content. If the content is a message with multiple attachments, a Multipart object (a container of BodyPart objects) is returned. If the Part's content is text, then a simple String object is returned. To retrieve a message's content, you can invoke Part's getContent() method, as illustrated below. Note that Part is an interface implemented by Message and BodyPart classes.

```
 Object o = part.getContent();
 If (o instanceof String) {
  // content is text
 } else if (o instanceof Multipart) {
  // recursively iterate over container's contents to retrieve attachments
 } else if (o instanceof  Message) {
  // message content could be a message itself
 }
```

You've now seen how messages are retrieved using JavaMail. If your application supports both IMAP and POP, you may need to develop different algorithms to download messages, depending on your particular use case and performance requirements. For example, when developing applications for use with low bit-rate clients using POP (which doesn't maintain flags to indicate unread messages), downloading all messages each time becomes a performance issue. Thus you may need an algorithm that uses provider-specific methods to prevent a redownload of messages. Depending on your requirements, other algorithms may be needed.

Each of these different download algorithms can be encapsulated as Strategy classes (Strategy design pattern) that share a common interface. A Strategy Factory can return strategy objects that can be used to download messages, depending on particular user configurations. This approach allows you to switch from one download algorithm to another, depending on the user configuration, and avoid having to use protocol-specific conditional statements. For more information on the Strategy and other design patterns, consult *Design Patterns* by Gamma et al. (1995).

When you download messages from a server, a feature that's available in some popular e-mail clients is an Inbox Assistant to process incoming messages (e.g., delete messages based on user-specified rules). Currently, JavaMail doesn't provide direct support for features such as automated message filtering.

Deleting e-Mail Messages

A standard use case for e-mail clients is deleting a message. Using JavaMail, deleting messages from a folder is a simple two-step process. First, messages are marked for deletion. Those messages that have been marked are then deleted from a folder by either explicitly invoking a Folder's expunge() method or closing a folder with the expunge parameter of the close() method set to true.

```
 // Set delete message flag
 message.setFlag(Flags.Flag.DELETED,
 true);
 // Delete marked messages from folder
 folder.close(true);
```

Conclusion

This short introduction should help you use JavaMail 1.1 to develop an e-mail client application. JavaMail is a relatively new framework that will inevitably continue to mature and evolve. Nevertheless, it can be used to rapidly develop an e-mail client application using a higher-level API without having to perform the arduous task of implementing specific mail protocols and developing an architectural infrastructure to support multiple protocols. You can obtain more information on JavaMail/JAF, including sample programs for sending and retrieving e-mail messages, from the references listed below.

Resources

JavaMail: www.javasoft.com/products/javamail/index.html
JavaBeans Activation Framework:
www.javasoft.com/beans/glasgow/jaf.html
Gamma, E., Helm, R., Johnson, R., and Vlissides, J. (1995).
Design Patterns: Elements of Reusable Object-Oriented Software.

AUTHOR BIO

Ian Moraes, Ph.D., is a principal engineer at Glenayre Electronics, where he works on the architecture and design of a unified messaging system. Ian has developed client/server systems for the telecommunications and financial services industries.

imoraes@atlanta.glenayre.com

Unify

www.ewavecommerce.com

What an 'L' of a month!

From IV drips and rejected usernames/passwords to . . . Neil Diamond (?)

WRITTEN BY
ALAN WILLIAMSON

What a month this one has been! Life has this wonderful way of letting you know that no matter what you're feeling at any given moment, you just can't predict what's going to be round the next corner. One of our chaps is at this precise moment lying flat on his back, bored senseless, in our local hospital. It all began last weekend.

Murray was having a quiet weekend – not doing much, just relaxing. He climbs into bed on Sunday evening with a wee nagging pain in his stomach. By the time Monday morning rolled round he was feeling a pain never before felt and the doctor was called immediately. Doctor then called an ambulance, and within five hours Murray was on an operating table having his ruptured appendix removed.

In a very short space of time poor Murray's world was completely turned round. All plans he had, all plans we had, are on hold for a number of weeks until he's back on his feet again. Poor bugger. I don't envy his position at all, complete with a drip into his arm. I have to admit to being a bit of a wimp with regard to this sort of thing. Needles and blood don't really work for me. I'm the sort of person that can nearly pass out watching "ER," and as far as I'm concerned nurses who have to deal with this sort of thing daily are pure angels. And if all that isn't enough, no e-mail access! The man is in hell! I think we should spearhead a campaign to have hospitals linked up to the Internet with at least 64K of bandwidth to each bed.

One thing about hospitals and the whole general health thing, it certainly focuses your mind. With us all going in on a daily basis we are more thankful for our own state of health, and now even more determined to try and maintain a general state of good health. However, we've been having some rather interesting conversations regarding all the software that must be running throughout the hospital. In the ward in which Murray is laid up is a vast array of devices that are continually flashing, reading, monitoring, alerting, whatever.

Let's imagine that the world is the utopia we Java developers believe it will be one day in the fact that Java has found a home in many of these application areas. Assume for the moment that Java is the underlying software running within a medical monitoring tool. Now, given that, let's pretend something goes wrong. Since this is all make-believe so far, let's pretend that a java.io.IOException is thrown for some reason. I don't know...the port reading the sensor from the patient goes wrong. What do you do? What is the worst-case scenario here? Well, death, I suspect.

When you think about it, how many of us would have faith enough in our own code to have our lives depend on it? I know there are a number of classes I've written that I'd probably risk somebody else's life on, but not my own!

Okay, we jest here. But when you think about how Java is structured, there's really so much to go wrong that's not even your responsibility. We all assume the virtual machine we entrust our bytecodes to has been coded to a very high standard and will run 100%. Of course it's not 100%, but we as developers have to make that assumption – otherwise we're merely building a house of cards.

That said, I believe I'd be more comfortable with, say, Java as opposed to a C/C++ alternative. Many other languages aren't quite as tolerant of mistakes as Java. The exception handling is very good, considering that many applications can still run after an error has been reported, which is in contrast to a bad memory reference in C/C++ that has the potential of bombing the program completely.

If anyone has had the experience of writing mission-critical health applications, I'd love to hear from you. Join our mailing list and let us know just how much testing goes into this kind of software. I think I'm scared to know this information in case it's not as much as I would expect. So, fingers crossed.

One L

This month we also ran into a funny wee problem that had us chasing our tails for at least a couple of days. I'm more than confident this is a situation many of you have had happen at some point in your development. Let me take you through the tale.

One of our clients, who will remain nameless, had this bright idea of using the password anjelia1, with a "one" at the end as opposed to a lowercase "L." Not a problem, we figured, and never gave it much thought until we started to move the end system from our development server to their production server.

For some reason we couldn't get our database connection up and running. It kept saying "incorrect username and password combination." The problem was further augmented by the fact that we had no control over the database as it was running on an external server. We couldn't even change the username and password to one that we knew.

So we had to work with the one we were given. But we were convinced we had the right username/password entered. Then one of us joked that the "1" looked very like a lowercase "L." Well, in actual fact, with the font choice in our telnet window and the font of the normal text in Netscape, they rendered to be pixel for pixel exactly the same. We had no idea what the "L" we had typed in the configuration file.

VisiComp

www.visicomp.com

Needless to say, we changed the configuration file very, very carefully, ensuring that our hands typed a "one" as opposed to an "L." Well, after restarting the server, the whole thing burst into life and the problem was solved.

It may seem obvious now, but at the time it was an eventuality we didn't even think about. So the moral of the tale is that from this point forward we have banned all use of ones and zeros from all usernames and passwords. Drastic I admit, but a decision I feel comfortable in enforcing.

I'm running out of space at this point so I'll just have to remember to tell you next time about a really bizarre encounter one of our chaps had with a Microsoft lawyer. If you knew how remote we really are, you'll think it even more bizarre. Next month, I promise.

Mailing List

As you know, each month I plug the mailing list that this column spawned. Well, some good news on this front. No, this isn't the last plug, but in fact a report of something good being born from the whole discussion list. We've been discussing many interesting topics, and an idea that's been bouncing around was that of an open-source Java library. This is where a core set of libraries would be held that would serve as a handy set of utility classes. We looked into offerings from other sites trying to do the same thing, but felt many of them lacked proper management. So at this precise moment we're looking to form our own version of this. For more information on this and other threads of discussion, come and join our mailing list. The good news is that getting off the bugger is just as easy as getting on to it.

To join send an e-mail to listserv@listserv.n-ary.com with subscribe straight_talking-l in the body of the e-mail. From there you'll get instructions on how to participate on the list.

Salute of the Month

This month I'd like to take my hat off to three people who, together, do a fine piece of work that all of us in the Java community benefit from. I'm talking about the core team behind the Apache JServ project, which enables the Apache Web server to run Java Servlets: Jon Stevens, Stefano Mazzocchi and Pierpaolo Fumagalli. They not only make Apache attractive to use, they're also pretty cool guys. We haven't come up against any combination like Apache + JServ that makes running servlets so easy, without compromising performance.

Jon "read-the-faq" Stevens deserves a special mention for the effort he puts into maintaining the JServ mailing lists. We've come across many in the corporate world who feel really uncomfortable with the lack of official support for using the likes of Apache and other open-source software. But the speed at which Jon answers questions puts a lot of companies that charge for this level of support to shame. So, Jon, on behalf of the developer community, thank you, sir, for a job well done.

Book Review

This month I haven't had time to read any books, which is truly a shame. If anyone has any good references to books they think I should read, please e-mail me. Would love to hear from you. But as one door closes, another opens. This month we've discovered the wonder of Neil Diamond. The man is a saint! I never appreciated what I dismissed as his middle-aged music before, but after hearing one of his compilation albums I realized I hadn't known just how much stuff he wrote that's been covered by many other artists. So I'm on a journey of Diamond-discovery and who knows where that will lead me to.

On that note, I have to tottle off now and get on with some real work….Sweet Caroline… bom-bom-bom!

alan@sys-con.com

AUTHOR BIO

Alan Williamson is CEO of n-ary (consulting) Ltd, the first pure Java company in the UK. A Java consultancy company with offices in Scotland, England and Australia, they specialize solely in Java at the server side. Alan is the author of two Java Servlet books and contributed to the Servlet API. He can be reached at alan@n-ary.com (www.n-ary.com).

SlangSoft

www.slangsoft.com

Segue

www.segue.com/ads/corba

CREATING NEWSFEEDS

Make the Internet more relevant, and give your customers better—and faster—service

WRITTEN BY JOHN KEOGH

The conventional way to present up-to-date information is to keep it on your Web site or a Web site you have some access to or control over so you can modify the information as needed. This article describes a way to create newsfeeds using Java applets so that the applet can be embedded anywhere in any page, and the applet distributor can keep the presented information up to date by modifying information on the site. Ways to extend the ability of the applet using CGI programming are also examined. This solves problems for theaters, clubs and other organizations that want their information on many pages but want to maintain it in just one spot. Establishments that want to distribute information about sales and events, and, of course, news organizations, can also benefit from distributing newsfeeds. This article also discusses a way to ensure that the information isn't loaded from cache (anticaching).

Conceptually, this applet connects back to the server and retrieves information – from a flat file or a script – as a stream of bytes that are cast to chars and appended to a StringBuffer. The toString() method in StringBuffer is called, and the resulting string is parsed to create a Vector containing the news and any other information returned by the server. When you want to use the applet as a newsfeed, you can permit other sites to embed the applet in their pages (using the CODEBASE="URL" attribute, with the URL being the directory on your server where the class files are, just as you can embed an image loaded from another server in another page, as shown in Figure 1). The applet then connects back to the server from which it was downloaded and gets the information, irrespective of the page in which it's embedded.

The Code

The source code for the applet, contained in the file newsfeed.java, is less than 300 lines, yet it contains all the functionality necessary to create a simple newsfeed. It uses Java 1.0 event handling to keep the event handling simple and to work with older browsers. To compile it using a JDK 1.1 compiler, you'll need to use deprecation:

```
javac newsfeed.java –deprecation.
```

The included packages and classes are needed for painting (awt.*), storing information in a vector (Vector), parsing the information returned from the server (StringTokenizer), networking (net.*), input/output classes and required exceptions (io.*), and working with dates (Date).

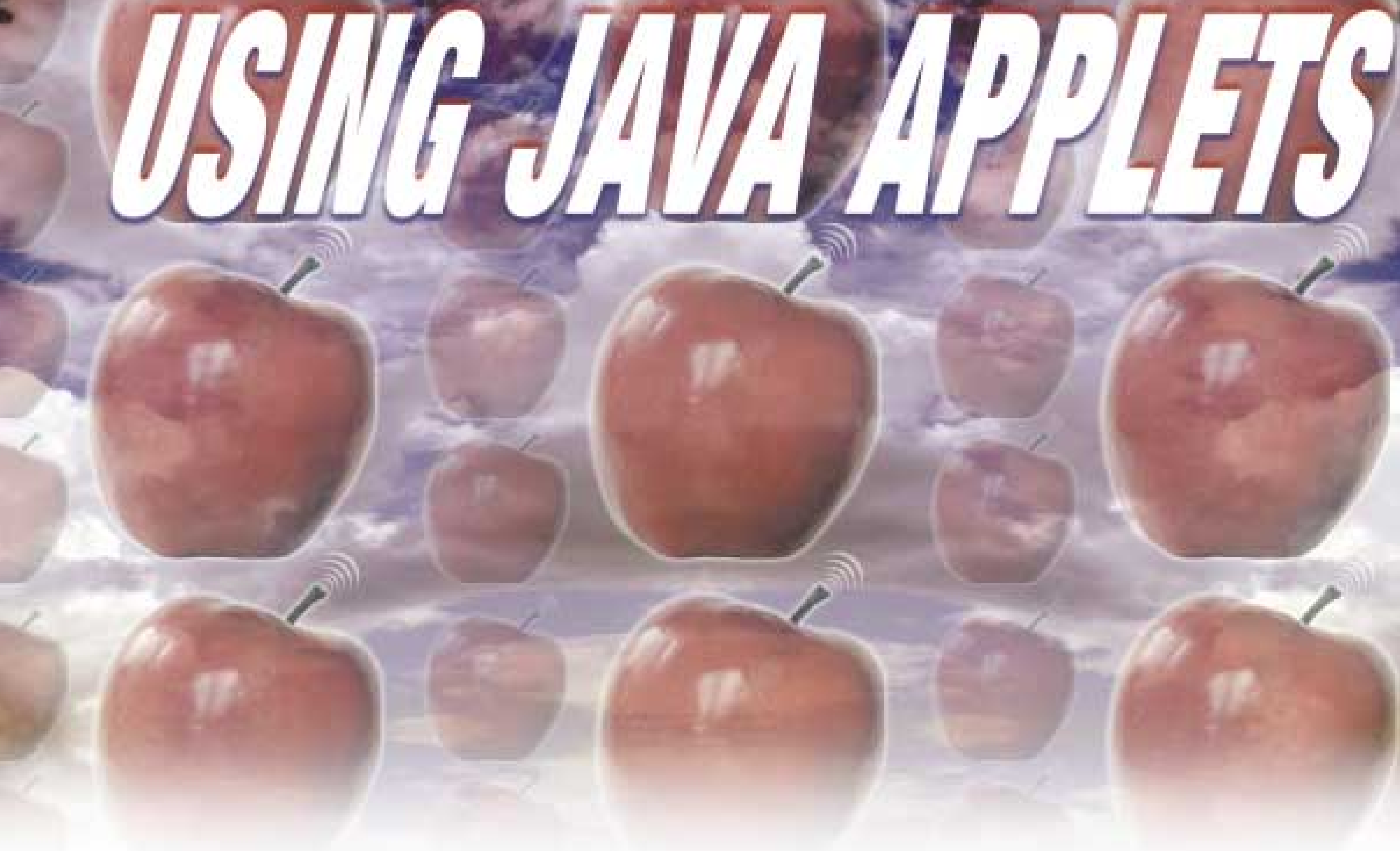
The integer variables that have global scope within the class – speed, iHeight, iTimesThrough and iCurrentPosition – store the speed of the thread and the font height and keep track of the number of times the retrieved news has been presented and the current position in the list of news items. The initial value of iTimesThrough, 11, is explained in the paint() method. The Thread variable, called newsthread, is used to implement the Runnable interface. An Image object is used for the background image. The String objects – strCurrentString and strCurrentURL – are used to store the current news item and the associated URL. The boolean bCalledFromRun, if true, increments iCurrentPosition when paint() is called. The Vector veInformation is used to contain the news items retrieved from the server.

The class extends Applet and can be run either in a browser or in AppletViewer. Implementing Runnable permits the class to spawn and run in its own thread. To implement the Runnable interface, we need to implement start(), stop(), and run() methods (discussed below).

Unlike most applets, newsfeed doesn't override init(); all the initialization is done in the initialize() method. This method is called from paint() the first time that paint() is called and every 10 times thereafter to refresh the news.

The initialize() method is the key method of the applet. The Date object that's created is used to get the current time in milliseconds (stored in lUnique), which is used to circumvent caching (anticaching is

USING JAVA APPLETS



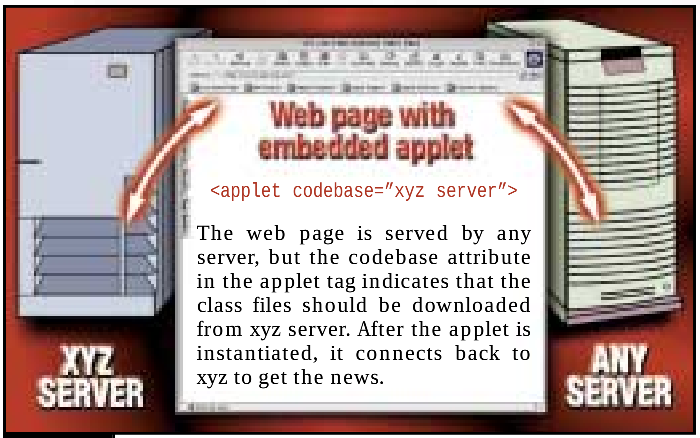

Web page with embedded applet
<applet codebase="xyz server">
The web page is served by any server, but the codebase attribute in the applet tag indicates that the class files should be downloaded from xyz server. After the applet is instantiated, it connects back to xyz to get the news.
XYZ SERVER
ANY SERVER

FIGURE 1 Applets and pages can be loaded from different servers

discussed later in this article). The passed-in graphics object is used to draw a string indicating that the news is being downloaded and to get the font metrics, from which the font height is determined and stored in iHeight.

The strType variable stores the type of news that's going to be requested. This information is used only if a script will process the request. Setting iCurrentPosition and strCurrentURL variables to 0 and ".", respectively, clears them.

Listings 1 contains five examples of URLs that can be read from. If you're experimenting with the applet using AppletViewer, you'd include the file called news.txt in the same directory as the applet (the file format of the news file is discussed later) and then run the applet in AppletViewer. Remember, when running in a browser, an applet can generally connect back only to the server from which it was downloaded. The URL string to use depends on whether you're using a flat file or a script, and the sophistication of the script. If you're using a flat file, you'd use "/news/news.txt". The applet gets the host name using getCodeBase().getHost() and creates a URL with the returned host and the path "/news/news.txt". If the applet was downloaded from www.lithic.com and you used the path "/news/news.txt", the URL would be http://www.lithic.com/news/news.txt. The "?"+lUnique circumvents caching. You can, of course, use other paths and file names, but the file must be a publicly accessible ASCII file in the correct format. The last three examples of URL are scripts, discussed later in this article.

One problem in retrieving the current news is that the requested URL may be loaded from cache. This can be overcome by creating a unique URL each time, which can be done by including a unique number in the query string. One way to accomplish this is to include the current time in the query string. If the time between queries is greater than the resolution of the system clock, this should overcome the caching problem. Using a Date to get the current time in milliseconds and appending it to the query strings of the script URLs is an effective anticaching strategy. The same strategy can be used with URLs that are not scripts, because they generally ignore the query string (an HTTP server will generally serve the same page if you type in index.html or index.html?123). The key thing is to present the browser with a URL that is unique so it won't find it in cache.

After the URL object is instantiated, openStream() is called, which returns an InputStream for reading from that connection. The InputStream object is used to instantiate a DataInputStream, which we read from until we reach the end of file, represented by an EOFException. The bytes returned from the DataInputStream are cast to chars, then appended to a StringBuffer. The string representing the contents of the URL is recovered by using the toString() method in StringBuffer. This is a general-purpose method for creating a string from the content of a URL – if you try this using (String)url.getContent(), you may get a ClassCastException. The vectorize() method takes the retrieved String and a token String as arguments and returns a Vector containing the news. In the information retrieved from the file or script, we expect the zeroth line – which becomes the zeroth node in veInformation – to be the name of an image stored in the codebase. After this image name is stored in strImage, the zeroth is removed, leaving as many nodes in the Vector as there

FIGURE 2 Applet background for a company

FIGURE 3 Applet background for a club

are news items. After the image is loaded, the variable that tracks the number of times the new items have been presented is set to 0.

To simplify working with the information returned from the server, the vectorize() method creates a Vector from it. The two parameters passed in are the String to parse and the String on which to tokenize it. If the String to parse ended with the token used to parse, we'd end up with a final piece of zero length. To avoid this, if the string ends with the parse character, it's pulled off. Because most files or streams of bytes sent via a server are punctuated by a <CR><LF>, both are checked for and removed if they end the strIn. Most of the work is done using a stringTokenizer. After a Vector is instantiated, the stringTokenizer is instantiated with the string to parse on and the string to parse. In our model the string to parse on is always a new line, but passing in the string to parse on makes the method more flexible. The code then tokenizes the string, adding each token to the Vector. The code that handles tokenizing by "\n" checks to see if a \r or \n is present and pulls them off the end of each token (they're assumed to be on the end; if your code needs these characters for anything, you may need to modify this code). The Vector is then trimmed to size and returned.

The paint() method, which is called repeatedly from run(), is used as the engine of this applet. The size of the applet is determined first, to be used to clear the applet's graphics context. If this is the first time through (iTimesthrough variable is initially set to 11, so initialize() will be called the first time through paint()), or if the number of times the news has been presented exceeds some arbitrary number of times (here 10), the news is refreshed using the initialize() method.

If an image is available, it's drawn on the background. The text is offset 45 pixels to be to the left of the image (it could also be drawn on top of the image). The examples in Figures 2 and 3 are just names for a company or club, but in practice they could be an advertisement or something associated with the news item. The image name is downloaded with the news, so it can be changed with each refresh.

Each line of news is in two parts: the URL is everything up to the first space, and the news item is everything after. After the current line of news is retrieved, the iCurrentPosition variable is incremented if bCalledFromRun is true. Initially the paint method is called repeatedly as the background image is drawn. To prevent an initial race through the news, the current position increments only if the paint() method was called from run(). New news is then drawn, in blue, 45 pixels from the left side of the applet and iHeight down (the iHeight variable was set to the font height in the initialize() method). Checking on whether the iCurrentPosition variable is bigger than veInformation, the Vector, which holds all the news items, permits the news to "wrap" back to the first news item if the current news item is the last. The iTimesThrough variable is incremented if the news wrapped, tracking the number of times that the current news has been displayed.

The mouseDown(), mouseEnter() and mouseExit() methods work together to create a mouse interface for the applet. In mouseDown(), if the current URL (set in paint()) isn't a ".", a URL object is instantiated and passed to the showDocument() method in the applet context for this applet. Using a "." permits presenting news items for which there is no link. The mouseEnter() method draws the current news item in red and shows the link in the status bar with showStatus(), both of which are common in applets that show menus or image maps. The thread is stopped in mouseEnter() so the current news item will remain displayed. The mouseExit() method undoes what the mouseEnter() method did by converting the text color back to blue, clearing the status and then starting the thread again.

The start(), stop() and run() methods are used to implement the runnable interface. A Thread object called newsthread is instantiated and started in the start() method. The run() method is called repeatedly, and the thread is paused for speed milliseconds; then repaint() is called. The stop() method is called when the applet stops.

To update the news in real time, just upload a new flat file or change the data in the database if you're using a database with a script. Flat files can also be changed by using CGI scripts that allow an authenticated user to modify, delete or add to the contents of the flat file. The next time the applet accesses the flat file or script, it'll get the current information (sometimes flat files are cached by the server, but in our experience this hasn't been a problem).

Working with CGI Scripts vs Flat Files

Whether using a flat file, a CGI script that uses a flat file for data or a CGI script that uses a DSN and a database, the applet expects the data to be presented in a certain format:

```
line1: Image<optional \r><\n >
line2: URL1[space]Text for news item 1<optional \r><\n >
line3: URL2[space]Text for news item 2<optional \r><\n >
.
.
.
Line: URLn-1[space]Text for news item n-1<optional \r><optional \n>
```

Any data source that can supply text in this format can act as a data source for the newsfeed. In practice, the designer would probably choose something with more flexibility and capability, but this works fine for our example. In any case, if you use a flat file, remember to upload it using ASCII mode.

The news.txt file has information in the format above, so it can be parsed directly by the applet. The information in cginews.txt must be supplied to the applet via newsfeed.pl, a Perl script. If you use the script and the cginews.txt, be sure that the permission of the script is executable, the path to the Perl interpreter is correct and the cginews.txt is world readable. Note that \n is read from the cginews.txt file; if the terminal character on each line is not a \n , the information won't parse correctly when it's received by the applet. (Perl programming is beyond the scope of this article, but if you have some experience, these scripts should be a starting point. If not, the flat file can provide most of the same functionality.)

The Perl script newsfeed.pl first retrieves the query string (this is what's after the "?"). If the URL accessed was http://www.lithic.com/cgi-local/newsfeed.pl?ls930353830, the query string would be ls930353830. The numbers after the Is represent the anticaching strategy discussed above. The first two characters are passed to the PrintNews function, which prints characters 2 through the last character of every line in the cginews.txt file that start with these two letters. In the example case the two letters are "ls". The first two characters of each line, which are the code compared to the type passed in, are removed before the line is printed. Using a code permits a single applet with a single data source to supply different news depending on the query string. This could be combined with cookies on the page the applet was embedded in, so user preferences could dictate what news was sent.

A guardian is set so that if no lines were printed for a given two-letter combination, a "." and a line indicating there was no news would be printed out.

Modifications to the Perl script and minor modifications to the applet would permit a more sophisticated interaction with the user. For example, if the URL http://www.lithic,com/cgi-local/moresubstantialnewsfeed.pl?type=Is&docbase=[doc base]&time=[time] were constructed and opened, where [doc base] and [time] are understood to be the page that the applet is embedded in and the current time, and the information was parsed by the script, the type and codebase would be known (knowing the codebase would permit returning an appropriate message to unauthorized newsfeeds, rather than the news). The time is still used for anticaching, but it may also be interesting statistically.

KL Group

www.klgroup.com/swingsuite

Conclusion

The newsfeed is easy to set up and distribute, but if you have some difficulty, here are some things to check.

- In the newsfeed.java, did you choose the right string for the URL (for example, if you have a news.txt in a feed directory, make sure you create a string that points at /feed/news.txt)?
- Check if the URL creation code was changed to url=new URL(str) (this is indicated in comments in the code).
- Did you upload the news file in ASCII mode and the class files in binary mode?
- You should use the Perl script only if you have some experience with Perl.
- You may want to remove the ?=code after the .txt if you're using a flat file and having problems.

This implementation of a newsfeed, though limited, created a very small class file that displays news, responds to button clicks when there is a link associated with a news event, and circumvents caching. Many improvements are possible: adding more colors, offsets, speed, audio clips and so forth. Additionally, double buffering would offer smoother graphics. The applet offers a functional framework in a small package.

Using a newsfeed solves many of the problems associated with the static nature of Web pages and solves access issues. For example, a document on a CD-ROM could present current information (provided that the person reading the HTML document in which the newsfeed was embedded was attached to the Internet). Used correctly, newsfeeds based on the applet described in this article can be a way to make the net more relevant, provide better service to your customers and substantially decrease the time it takes to disseminate information.

AUTHOR BIO

John Keogh is president of Lithic Software Corporation, an Internet software and services company. He has a background in computer science, chemistry and technical writing. John programs in C, C++, Java, Perl, SQL and several other languages. You can visit his Web site at www.Lithic.com/.

keo@lithic.com

Listing 1: newsfeed.java

```
// newsfeed.java a class for creating a newsfeed
// includes

import java.awt.*;
import java.util.Vector;
import java.util.StringTokenizer;
import java.net.*;
import java.io.*;
import java.util.Date;

//class that implements a simple newsfeed
public class newsfeed extends java.applet.Applet
                                implements Runnable
  {
  int speed=2000, iHeight=0, iTimesThrough=11,
      iCurrentPosition;
  Thread newsthread;
  Image image;
  String strCurrentURL,  strCurrentString;
  boolean bCalledFromRun=false;
  Vector veInformation;

// called from paint when needed, don't want to
// call from init, because we need a graphics
// object for this
  private void initialize(Graphics g)
    {
    String strType=getParameter("type");
    if(strType==null)
      strType="ls";

    //a date is used to prevent caching
    Date date=new Date();
    long lUnique=date.getTime();

    iHeight=g.getFontMetrics().getHeight()+2;

    g.setColor(Color.black);
    g.drawString("Getting current news...", 45,
                  iHeight);

    strCurrentURL=".";
    iCurrentPosition=0;
    //first one for use on your own system, with
    //appletviewer, second third, fourth, or fifth
    //for use on the Web, depending on if you use
    //a flat file in a directory called news, or a
    //script in a cgi-local or cgi-bin directory, and
    //how the script parses the query string
    String str="news.txt";
    //String str="http://"+getCodeBase().getHost()+
    //           "/news/news.txt?"+lUnique;
    //String str="http://"+getCodeBase().getHost()+
    //           "/cgi-local/newsfeed.pl?"+
    //           strType+iUnique;
    //String str="http://"+getCodeBase().getHost()+
    //           "/cgi-bin/newsfeed.pl?"+
    //           strType+lUnique;
    //String str="http://"+getCodeBase().getHost()+
    //          "/cgi-bin/newsfeed.pl?type="+
    //           strType+"&time="+lUnique+
    //           "&docbase="+getDocbase();

    //a url object is created, then a stream is created
    //to retrieve the content of the url
    URL url=null;
    String strContent="";
    try
      {
      //first one for use on your own system, with
      //AppletViewer, second for use on the Web
      url=new URL(getCodeBase(), str);
      //url=new URL(str);
      }
    catch(MalformedURLException m){return;}
    try
      {
      StringBuffer sb=new String-
      Buffer("");
```

BlueSky

www.blueskysoftware.com

```
      InputStream is=url.openStream();
      DataInputStream dis=new DataIn-
      putStream(is);
      while (true)
        {
        try{sb.append((char)dis.read-
        Byte());}
        catch(EOFException f){break;}
        }
      strContent=sb.toString();
      }
    catch(IOException e){return;}

    //after the contents of the url are
    //retrieved, they are parsed to
    //create a Vector which contains
    //the information
    veInformation=vectorize(strContent, "");

    //get the background image, then
    //remove it
    String strImage=(String)veInforma-
    tion.elementAt(0);
    veInformation.removeElementAt(0);

    //load the image, if there is one
    if(strImage.compareTo(".")!=0)
      image=getImage(getCodeBase(),
      strImage);

    iTimesThrough=0;
    }

  //parses the returned information
  //based on <CR>
  private Vector vectorize(String strIn,
                          String strParse)
    {
    //if it ends with our parse charac-
    ter, pull it off the end
if(strIn.endsWith("\r")||strIn.endsWith(
"")||strIn.endsWith(strParse))
      strIn=strIn.substring(0,
      strIn.length()-1);
if(strIn.endsWith("\r")||strIn.endsWith(
"\n")
      strIn=strIn.substring(0,
      strIn.length()-1);

    Vector ve=new Vector(1);

    //get ready to tokenize the string
      StringTokenizer t = new StringTo-
      kenizer(strIn, strParse);
    int iLines = t.countTokens();

    if(strParse.compareTo("\n")==0)
      {
      for(int i=0; i<iLines; i++)
        {
        String str=t.nextToken();
        if(str.indexOf("\n")!=-1)
          str=str.substring(0,
          str.length()-1);
        if(str.indexOf("\n")!=-1)
          str=str.substring(0,
          str.length()-1);
        ve.addElement(str);
        }
      }
    else
      {
      for(int 1=0; i<iLines; i++)
        ve.addElement(t.nextToken());
      }

    ve.trimToSize();
    return ve;
    }

  public void paint(Graphics g)
    {
    Dimension d=this.size();

    //the first time through, and each
    //ten thereafter, get the news
    if(iTimesThrough>10)
      {
      //clear this
      g.setColor(Color.white);
      g.fillRect(0, 0, d.width, d.height);

      initialize(g);
      }

    //clear
    g.setColor(Color.white);
    g.fillRect(0, 0, d.width, d.height);

    //draw the background image
    if(image!=null)
      g.drawImage(image, 0, 0, this);

    //get the current text and url,
    //then draw the text
    String strTemp=(String)veInforma-
    tion.elementAt(iCurrentPosition);
    strCurrentURL=strTemp.substring(0,
    strTemp.indexOf(" "));
    strCurrentString=strTemp.sub-
    string(strTemp.indexOf(" ")+1);
    g.setColor(Color.blue);
    g.drawString(strCurrentString, 45,
    iHeight);

    //only advance if called from run
    if(bCalledFromRun)
      iCurrentPosition++;
    bCalledFromRun=false;

    if(iCurrentPosition>=veInfor-
    mation.size())
      {
      iCurrentPosition=0;
      iTimesThrough++;
      }
    }

  //overide update for smoother graphics
  public void update(Graphics g)
    {
    paint(g);
    }

  //If there is a mousedown go to the url
  //associated with this news event.   A
  //"." is used if there is no url
  //associated with the news item
  public boolean mouseDown(Event e, int
  x, int y)
    {
    if(strCurrentURL.compareTo(".")!=0)
      {
      URL url;
      try{url=new URL(strCurrentURL);}
      catch(MalformedURLException
      m){return true;}
      this.getAppletContext().showDocu-
      ment(url);
      }
    return true;
    }

  //if the mouse enters the applet,
  //repaint the text red and stop run-
  //ning. Set the status to the url
  //associated with the news item
  public boolean mouseEnter(Event e,
  int x, int y)
    {
    Graphics g=this.getGraphics();

    //clear
    Dimension d=this.size();
    g.setColor(Color.white);
    g.fillRect(0, 0, d.width,
    d.height);

    //draw the background image
    if(image!=null)
      g.drawImage(image, 0, 0, this);

    g.setColor(Color.red);

    g.drawString(strCurrentString, 45,
    iHeight);
    if(strCurrentURL.compareTo(".")!=0)
      showStatus(strCurrentURL);
    else
      showStatus("No Link");
    this.stop();
    return true;
    }

  //if the mouse exits the applet,
  //repaint the text blue and start
  //running again. Clear the status
  public boolean mouseExit(Event e, int
  x, int y)
    {
    Graphics g=this.getGraphics();
    g.setColor(Color.blue);
    g.drawString(strCurrentString, 45,
    iHeight);
    this.start();
    showStatus("");
    return true;
    }


  //start stop and run are required to
  //implement the Runnable interface.
  //Speed can be adjusted to present
  //information faster or slower.
  public void start()
    {
      newsthread = new Thread(this);
      newsthread.start();
    }

  public void stop()
    {
      newsthread.stop();
    }

  public void run()
    {
      while (true)
        {
        bCalledFromRun=true;
          try {Thread.currentThread()
          .sleep (speed);}
        catch (InterruptedException e){}
          repaint();
        }
    }
 }
```

Listing 2: news.txt

```
lithic.gif
tmtm.gif
lshttp://www.lithic.com/java/Person-
alChatware.html Have a look at the plu-
gin API in lpc
lshttp://www.lithic.com/java/calcula-
tor.html See Lithic Software's newest
applet
```

Sybase

www.sybase.com

```
lshttp://www.lithic.com/onlinesystems/in
dex.html Check out LSC Online Systems
tmhttp://www.toastmasters.org/ Toastmas-
ters main page is more useful than ever
tmhttp://www.lithic.com/tm/tm.html Grand
Junction Clubs now have a page
```

Listing 3: newsfeed.html

```
<HTML>
<HEAD>
<title>Lithic Newsfeed</title>

<!newsfeed.java a page with the news-
feed.class embedded>

</HEAD>


<BODY BGCOLOR=White>

<CENTER>
<APPLET CODE="newsfeed.class" width=300
height=20>
<PARAM NAME=type VALUE="ls">
</APPLET>
</CENTER>

<BODY>

</HTML>
```

Listing 4: newsfeed.pl

```
#!/usr/bin/perl
# newsfeed.pl

#newsfeed.pl a script for sending out news
MAIN:
  {
  $file="cginews.txt";
  $querystring=$ENV{"QUERY_STRING"} ;
  print("Content-type: text/html");
  if($querystring)
    {
    $query=substr($querystring, 0, 2);
    &PrintNews($file, $query);
    }
  }

#prints out all the lines that begin
with the
#appropriate letters the letter is the
token
#sent in the query string
sub PrintNews
  {
  local($file, $control)=@_;
  local($guardian);

  $guardian="";

  open(TIMES, $file);
  while(<TIMES>)
    {
    $type=substr($_, 0, 2);
    if($type=~/$control/)
      {
      $content=substr($_, 2);
      print($content);
      $guardian="t";
      }
    }
  close(TIMES);

  #if nothing was found, just print out
  that no news is currently available
  if(!$guardian)
    {
    print("");
    print(" no news currently avail-
able");
    }
  }
```

Listing 5: cginews.txt

```
lslithic.gif
tmtm.gif
lshttp://www.lithic.com/java/Person-
alChatware.html Have a look at the plu-
gin API in lpc
lshttp://www.lithic.com/java/calcula-
tor.html See Lithic Software's newest
applet
lshttp://www.lithic.com/onlinesystems/in
dex.html Check out LSC Online Systems
tmhttp://www.toastmasters.org/ Toastmas-
ters main page is more useful than ever
tmhttp://www.lithic.com/tm/tm.html Grand
Junction Clubs now have a page
```

Instantiation

www.instantiations.com

PointBase

www.pointbase.com/devlic/jdj

Palming Java

Cross-device portability

WRITTEN BY
AJIT SAGAR

I'd like to start this month's article with some of my impressions of JavaOne '99. Last year was far more exciting with promises of new magic kits and potions handed out in abundance. This year there was a definite touch of reality in the air with less sleight of hand and more live rabbits actually jumping out of the hat and onto the stage. The smoke and mirrors were still there, but there was some substance behind them.

The "real" feel to JavaOne is due to the fact that Sun seems to have finally gotten its story straight in terms of what "ubiquitous" really means. When you look at WORA (write once, run anywhere), the promise still holds true. However, *what* is it that you end up running anywhere? Is it the same Java? I don't know about you, but I feel more comfortable knowing that the Java that's going to run on my pager isn't the same one that runs my banking application.

The Java platform has been segregated into three platform editions. Sun's Web site explains their strategy: "Recognizing that 'one size doesn't fit all,' Sun has regrouped its innovative Java technologies into three editions: Micro (J2ME), Standard (J2SE) and Enterprise (J2EE)."

A detailed discussion on the purpose of each edition of the Java platform is best left for another time. This month I'd like to focus on some of the application areas of the Java 2 Micro Edition – specifically, the 3Com PalmPilot consumer device and how it leverages the features of the Java platform. We'll also look at why the PalmPilot may be the killer application Java has been looking for.

Consumer Devices and Java

Java started out as a language for embedded consumer devices, specifically for set-top boxes like the toaster and the television. Over the last four years, Java has made its impact on the enterprise as the language and platform of choice for designing distributed, enterprise-level business solutions. Indeed, Java is one of the key enablers for the rapidly evolving areas of e-commerce.

One of these areas is the world of intelligent consumer devices where the intelligence is built into the network. The various devices, such as cell phones, pagers and smart cards, need to be able to access the network via a common computing platform. The J2ME edition of Java attempts to provide such an environment. The most practical requirement on a software platform that can be embedded in these consumer devices is its footprint, which must be extremely small. One of the main features of the J2ME is its tiny footprint.

Sun has been giving away the Java language and associated APIs for free, but that doesn't bring the money in. With the acceptance of Java as the common platform for consumer devices, Sun has a foot in the door of an extremely large market. Now they can really start collecting on their investment in Java.

Java 2 Micro Edition

J2ME bundles the APIs for software development in the consumer space. This includes devices ranging from rings, smart cards and pagers to more intelligent PDAs like the PalmPilot and cell phones – all the way up to appliances with set-top boxes such as TVs. With J2ME Java provides a complete end-to-end solution for creating networked products and applications for the consumer and embedded markets.

The J2ME framework further segregates the major types of consumer devices by grouping them into a limited number of categories with varying levels of built-in intelligence. To help content developers, each category has a profile; it's defined in the form of a specification of the minimum set of APIs useful for a particular product, and a specification of the Java Virtual Machine functions required to support those APIs.

The PalmPilot — Java's Saving Grace?

Back to JavaOne – another obvious thing was the change in what Sun was peddling this year. Last year it was all about network computers. The NC was going to be the ultimate consumer-end device; everything else would reside on the network. Another thing I noted – there wasn't much talk about rings and buttons.

This year, though the message – "The Network Is the Computer" – was still the same, the vehicle for conveying it was radically different. It seems that the ultimate device for taking Java to the streets has been identified as 3COM's PalmPilot. I see this as a smart move. Sun is tapping into the large market the Palm has and will have in the future.

My take on the alliance is that the Palm has given Java the home it was desperately looking for. Even though Java was finding applicability in other consumer devices, it wasn't making a significant impact. Over the last two years Sun has introduced tailored JVM technology to serve products in the consumer and embedded markets. These include Personal Java technology targeted at screen phones, high-end PDAs and set-top boxes, and Java Card technology targeted at smart cards, the Java Ring, I-Button, etc., which have yet to get the buy-in from consumer-oriented vendors. A more intelligent device was needed that would do a lot more than allow you to brew your coffee. The Palm comes with a large set of application suites, so it already has the beginnings of industry verticals stemming from it.

KVM — The Palm's Keys to "Ubiquity"

The relationship between the Palm and Java is going to be one of symbiosis. The market for one feeds the market for the other. To increase its lead in the marketplace, specifically against the CE, the Palm needs to have an open, nonproprietary interface for working with other devices. This is made possible by Sun's Java KVM, which forms the core of J2ME.

The KVM is so named because its size is measured in the tens of kilobytes, around 80–100K. It fits in the tiniest handheld devices such as pagers, and needs to run on an underlying operating system. In the case of the PalmPilot, this operating system is the PalmOS. The KVM is a new Java runtime environment built from the ground up to make an extremely lean implementation of the JVM.

New Atlanta

www.newatlanta.com/

The KVM has been developed by Sun in collaboration with other industry partners, such as service providers. These partners are crucial for making sure the KVM can truly enable mobile network devices such as digital cellular phones, pagers, mainstream personal digital assistants, low-end analog set-top boxes and small retail payment terminals.

The KVM binary code will be available in prerelease form for 3Com's Palm II and Palm V. Sun anticipates that a wide range of wireless devices containing the KVM will become available early in 2000.

PalmOS — Java's Gateway to the Consumer?

As mentioned above, the KVM needs an OS to run on. The PalmOS software will serve as a primary reference platform for application development using KVM. Sun and 3COM are collaborating to provide an end-to-end solution for delivering content and Java applications to Palm computing platform devices via Sun's software products. The next level of integration will involve joining Java with 3Com's Palm.net service, the recently announced wireless service for the Palm VIII.

Rubbing the Magic Lamp...

Another enabler for this new world of consumer devices is Java's Jini technology, which allows a device on the network to discover other devices and query them for the services they offer. The Palm will be another such device. With Jini, the Palm can become truly network-enabled and leverage services from a plethora of computers, appliances, enterprise-level systems and consumer devices.

Putting It in the Commerce Perspective

J2ME and the KVM help solidify Sun's vision of "providing the dot in the .com" networking world. This ties in with the e-business model of service-based businesses on the Web. The current commerce market is rapidly shifting from a product-based paradigm to a service-based one. This means that instead of installing software products, organizations are shifting toward a model in which the products are hosted at remote locations and their services are available across the network. All the necessary software can be downloaded across the Internet.

This isn't a reality yet, but Java enables this business model because of its rich support for networking, dynamic nature and portability. The Palm is one of the many devices that can play in this area. Once the underlying infrastructure is in place, these devices can play more substantial roles in e-commerce transactions. For example, the Palm can act as a kiosk for purchasing goods, quoting services, auctioning products and so on.

Cross-Device Portability

It looks like Java and the Palm have entered into a marriage made in heaven. However, fidelity isn't necessarily a precondition for this relationship. One of Java's greatest claims to fame is its portability. In the consumer device segment, i.e., using J2ME, it translates to portability across these devices. The Palm is one of the many consumer devices that will support an edition of the Java VM. Indeed, several other consumer devices already support Java on their operating systems. Basically, Java allows them to speak the same lingo.

Trading Places

The market for e-business is expected to grow to over a trillion dollars by 2003. My contention is that in about 10 years the only kind of commerce that will exist in this world is e-commerce. Programming languages come and go, but the applications they give birth to last a long time. Devices like the Palm are introducing completely new suites of applications that will help define the end-user interface in the world of e-commerce. The combination of the Palm and Java creates a powerful, exciting platform that's revolutionizing the way e-business is conducted today.

ajit@sys-con.com

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

Quick Steam

www.quickstream.com

Tidestone

www.tidestone.com

By now you might be asking yourself the question...

What Is JavaMail?

WRITTEN BY RACHEL GOLLUB

JavaMail is a set of abstract classes that create a framework for sending, receiving and handling e-mail, along with implementations of those classes. The package Sun provides contains implementations of IMAP and SMTP, allowing you to get started immediately on sending and receiving mail. They also provide a separate POP3 implementation that I'll describe below. The framework makes it easy to create your own cross-platform mail application without an in-depth knowledge of e-mail. Methods and classes that allow you to access mail folders, download messages, send messages with attachments and filter mail are included.

JavaMail has a number of uses in personal and Enterprise-level programming. It can be used to create personal mail filters, simple mailing lists and customized personal mail applications, as well as to add full e-mail capabilities to an Enterprise application or create a full-fledged e-mail client. A number of products currently available are built for or around JavaMail; many are listed on the JavaMail Third Party Product page (http://java.sun.com/products/javamail/Third_Party.html). Several companies have written marketing applications that use JavaMail to send customized mail to groups of contacts, and many companies have written new e-mail clients using JavaMail and its extensions.

Background

Knowing a little about what e-mail is and how it is sent and received is helpful in creating e-mail applications. E-mail protocols are, in the most basic sense, a way of transferring data from one machine to another, possibly going through several other machines along the way. SMTP has existed, essentially in its current state, since the early 1980s, and POP and IMAP aren't much younger. The preceding is a very basic introduction to these protocols; see "Where to get more information" for more information.

SMTP – Simple Mail Transfer Protocol – allows two mail servers to communicate using a simple language, and provides a step-by-step protocol for exchanging information. To use the example that follows, you'll need an SMTP server. If you're using UNIX, you probably have one on your machine already. If you're not, or don't have a server installed, you can ask your system administrator for the name of your SMTP server.

IMAP – Internet Mail Access Protocol – and POP3 – Post Office Protocol Version 3 – are client/server mail protocols. SMTP delivers mail to a central location, where the user can either log in and read it directly or use a client/server mail protocol to read it remotely. IMAP is designed to keep mail on a remote server and let the user interact with it there, while POP3 is designed to forward a user's mail to a single machine, where the user can go offline and read it, if necessary. In general, people with slow connections (dial-up or otherwise) tend to use POP3 because they can connect and download their mail without having to keep the connection open afterwards. People with fast connections and multiple machines usually use IMAP so they can read mail from whichever machine they happen to be on without losing access to the mail they read elsewhere. In the following examples, you will need a POP3.

How Do I Get Started?

First, you'll need the JDK (I use 1.2 in this article), JavaMail package and the JavaBeans Activation Framework extension (JAF). JAF is a standard extension that provides a way to identify and correctly process unknown data. You can download each from the Java Web site. JDK 1.2 (also known as the Java 2 SDK) is at http://java.sun.com/products/jdk/1.2/, and JavaMail and JAF are at http://java.sun.com/products/javamail/index.html. Follow the installation instructions given on the download pages, and install the packages. JavaMail comes with a number of examples in the demo directory, many of which are easy to use immediately or can be modified for specific situations.

A simple example is msgsendsample.java. This sends a simple message (included in the example) to an e-mail address provided by the user. To use this, go to the demo directory and compile msgsendsample.java. Then, on the command line, type:

```
 java msgsendsample <your email address> <your email address> <your
SMTP server> false
e.g.
 java msgsendsample rmg@silentq.com rmg@silentq.com
smtp.silentq.com false
```

The command line arguments are <address to> <address from> <mail server> <debug>. You can actually specify any e-mail address for the from argument and the message will appear to be from that e-mail

Fierano

www.fiorano.com

address, but a look at the full headers will show the real source. The false at the end indicates that you don't want to see the debug information; change that to true for a look at what the example is doing. The complete SMTP connection dialog will be printed out, as well as some debug information for the example.

A look at the source shows how this example works:

```
Properties props = new Properties();
props.put("mail.smtp.host", host);
```

JavaMail needs the mail host property set to determine where the SMTP server is located. If it's on your local machine, the send can happen locally. Otherwise it will automatically connect to the server machine and send from there.

```
Session session = Session.getDefaultInstance(props, null);
```

This gets a mail session (notice it uses the properties just created). The default mail session is created if it doesn't already exist, and permissions are created or checked. Since the Authenticator (security object – the second parameter listed) is null, there's no security on this object, and anyone can access it.

```
Message msg = new MimeMessage(session);
msg.setFrom(new InternetAddress(from));
 InternetAddress[] address = {new InternetAddress(args[0])};
msg.setRecipients(Message.RecipientType.TO, address);
msg.setSubject("JavaMail APIs Test");
msg.setSentDate(new Date());
msg.setText(msgText);
```

This section creates the actual message object and fills in the to, from, subject, date and content. There are also options to set the reply to, content and content type, and other header information. Since this is a MIME – Multipurpose Internet Mail Extensions – message, it can have several parts, none of which have to be plain text. You may want to set a section to HTML or add an attachment. You can set a DataHandler (in the JDK) using setDataHandler() in MimeMessage to handle nontext parts. This is a simple one-part text message, so you can use the setText() method instead.

```
Transport.send(msg);
```

This actually sends the message, using SMTP. If the send fails, it will throw exceptions based on the problem with the send. For example, try sending to a nonexistent address for example, ("rachelfoo"). You'll see a SendFailedException, with some details about the address that failed.

With a few changes you can modify this example to send mail programmatically. Changing the <to>, <from>, <host>, <subject> and <content> values to method arguments and adding a constructor to set them will allow you to instantiate this object from another class and send mail automatically. In addition, you can add a hashtable of substitutions and include these in your message text to personalize the e-mail – see Listing 1. This listing creates the hashtable individually for each e-mail, but this could be customized to read from a database or file.

A More Complex Code Sample

The next example will read and optionally delete messages from a POP3 server. The first step is to download and install a POP3 provider. The example uses Sun's POP3 provider. Follow the directions from the JavaMail page at Sun (listed above) to download and unzip the package, and set your classpath correctly. Since you may want to use other providers later, you need to add this provider to your provider list. To do this, create a new directory, copy the mail.jar file into that directory and unjar it (jar xvf mail.jar). You'll see several directories, including a META-INF directory, which generally contains information about the jar file that includes it, but in this case also contains information critical to JavaMail. This is important to remember. If you ever install JavaMail as part of a product, be sure to include this directory in your product package.

To add the POP3 provider, create a file called javamail.providers with the following line in it:

```
protocol=pop3; type=store; class=com.sun.mail.pop3.POP3Store;
```

Then jar the directories again (jar cvf mail.jar *), back up the old mail.jar and move the new mail.jar into place. You now have a POP3 provider added to your provider list.

The POP3 example is included here as Listing 2. Download and compile the program – you'll need a POP3 mail server to test it. The example takes parameters <username> <password> <server> <delete>. The last parameter specifies whether you want to delete any messages you read from the server. Test the program with:

```
 java MailPrinter <your POP3 username> <your POP3 password> <your
POP3 mail server> false
e.g.
 java MailPrinter rachel mypassword pop3.silentq.com false
```

You'll see a list of all your new messages with their headers. The thread will continue, downloading all messages every five minutes until you stop it. This could easily be changed to append to a file; with the delete option set to true, it would constantly update your local mail file. It could also be altered simply to check the number of messages and, with a small user interface, could be made into a mail notification system.

The source can be broken down as follows:

```
Properties lProperties = System.getProperties();

Session lSession = Session.getDefaultInstance(lProperties, null);
```

As in the last example, the session is started with no security. This time it's started with the system properties instead of an empty Properties object.

```
Store store = null;
Folder folder = null;
try {
   store = lSession.getStore(protocol);
}
```

This part gets the store type for the given protocol (POP3). Store is an abstract class that models a message store. This allows connections to various types of actual message stores with no loss of generality.

```
store.connect(host, user, password);
```

Now it connects to the remote server (host) with the given username and password. Obviously, it would be unwise to do this part programmatically without a fair number of precautions – storing your password in plain text in the source is a security problem. It might be wise to store the password encrypted in your database or file system and retrieve and unencrypt it only when needed.

```
folder = store.getDefaultFolder();
...
folder = folder.getFolder("INBOX");
```

This initializes the folder with the standard default mailbox, indicated by the key word "INBOX". In this context INBOX stands for "primary folder for this user on this server" – it will vary by protocol, server and user.

```
folder.open(Folder.READ_WRITE);
...
Message[] messages = folder.getMessages();
```

ObjectSwitch

www.objectswitch.com/giotto45

Here the folder is opened and the messages are downloaded. This includes their headers and all parts of the message (in the case of MIME messages). This doesn't remove them from the server; it just copies them into memory.

```
if (delete)
  messages[i].setFlag(Flags.Flag.DELETED, true);
...
folder.close(true);
store.close();
```

Flags indicate the status of the message in the folder. Here each downloaded message is marked deleted, and when the folder is closed the "true" flag indicates that all deleted messages should be removed.

```
 for (Enumeration e = pPart.getAllHeaders(); e.hasMoreElements(); )
{
   Header header = (Header) e.nextElement();
   System.out.println(header.getName() + ": " + header.getValue());
 }
```

This section gets each of the header lines from the message and prints them out. You could change this part to print out only the headers you're interested in using: getMatchingHeaders() instead of getAllHeaders().

The rest of the example recursively divides the message into parts and prints each part. This example could easily be expanded to filter out messages with selected subjects or messages from particular addresses. It could store the messages in a database or to a file. In all, JavaMail gives you powerful tools to filter and manipulate e-mail messages with very simple code.

For More Information

1. JavaMail home page: http://java.sun.com/products/javamail/index.html
2. RFC 821 (SMTP specification: http://info.internet.isi.edu/in-notes/rfc/files/rfc821.txt
3. IMAP, POP3: ftp://ftp.cac.washington.edu/mail/imap.vs.pop

   For anything not covered in these sources, a simple Net search will probably find what you need. E-mail is a frequently discussed topic on the Net, and there are FAQs and articles everywhere. Good luck!

Author Bio

Rachel Gollub, founder of SilentQ Software Company, learned Java as a JavaSoft engineer from 1995 to 1997, and has worked at several start-ups since then. She writes articles and gives presentations on Java and related subjects.

rmg@silentq.com

Listing 1

```
/*
 * SendMessage.java
 */

import java.util.*;
import java.io.*;
import javax.mail.*;
import javax.mail.internet.*;
import javax.activation.*;

/**
 * This is copied almost directly from
 * msgsendsample.java, by Max Spivak,
 * with changes by Rachel Gollub
 * (rachel@silentq.com).  More details
 * are available in the accompanying
 * article.
 */
public class SendMessage {

  public boolean send(MessageObject message) {
    //create some properties and get
    //the default Session
    Properties props = new Proper-
    ties();
    props.put("mail.smtp.host", mes-
    sage.host);
    // Uncomment this to debug
    //props.put("mail.debug", args[3]);

    Session session =
      Session.getDefaultInstance(props,
      null);
    session.setDebug(false); // true to
    debug

    try {
      // create a message
      Message msg = new MimeMessage(ses-
      sion);
      msg.setFrom
        (new InternetAddress(mes-
         sage.from));
      InternetAddress[] address =
        {new InternetAddress(mes-
         sage.to)};
      msg.setRecipients
        (Message.RecipientType.TO,
         address);
      msg.setSubject(message.subject);
      msg.setSentDate(new Date());
      msg.setText(replace
        (message.custom,
message.text));

      Transport.send(msg);
    } catch (MessagingException mex) {
      System.out.println
        ("\n--Exception handling in " +
         "msgsendsample.java");

      mex.printStackTrace();
      System.out.println();
      Exception ex = mex;
      do {
        if (ex instanceof
          SendFailedException) {
          SendFailedException sfex =
            (SendFailedException)ex;
          Address[] invalid =
            sfex.getInvalidAddresses();
          if (invalid != null) {
          System.out.println
            ("    ** Invalid Addresses");
          if (invalid != null) {
            for (int i = 0;
              i < invalid.length; i++)
              System.out.println
                ("       " +
invalid[i]);
            }
          }
          Address[] validUnsent =
            sfex.getValidUnsentAddress-
es();
          if (validUnsent != null) {
            System.out.println
              ("    ** ValidUnsent
Addresses");
            if (validUnsent != null) {
              for (int i = 0;
                i <
validUnsent.length; i++)
                System.out.println
                  ("
"+validUnsent[i]);
              }
            }
          Address[] validSent =
            sfex.getValidSentAddress-
es();
          if (validSent != null) {
            System.out.println
              ("    ** ValidSent
Addresses");
            if (validSent != null) {
              for (int i = 0;
                i < validSent.length;
i++)
                System.out.println
                  ("          "+valid-
Sent[i]);
              }
            }
          }
        System.out.println();
        } while ((ex = ((MessagingEx-
ception)ex)
          .getNextException()) !=
null);
   return false;
 }
      return true;
    }

  /**
   * This is a simple replace method.
It
   * searches for the keywords, and
uses
   * a StringBuffer to replace them.
   *
   * @param macros The keywords and
values.
   * @param message The message to
change.
   *
   * @return The changed message.
   */
  public String replace (Hashtable
macros,
                           String mes-
sage) {
    for (Enumeration e = macros.keys();
      e.hasMoreElements(); ) {
      int index = 0;
      String key = (String) e.nextEle-
ment();
      String value = (String)
macros.get(key);
      while ((index =
        message.indexOf(key, index))
!=-1) {
        StringBuffer text =
          new StringBuffer(message);
        message = text.replace(index,
          index + key.length(), value)
          .toString();
        index += value.length();
      }
```

Cerebellum

www.cerebellumsoft.com

```
    }
    return message;
  }
}


/*
 * MakeMessage.java
 */

import java.util.Hashtable;

/**
 * This is a test class that creates
some
 * MessageObjects to send with SendMes-
sage.
 *
 * @author Rachel Gollub (rachel@silen-
tq.com)
 */
public class MakeMessage {

  /**
   * The main method initializes the
objects
   * and sends them.
   */
  public static void main(String
args[]) {

    SendMessage message = new SendMes-
sage();
    MessageObject object =
      makeObject("rachel@silentq.com",
                 "Rachel", "choco-
late");
    message.send(object);
    object = makeObject("jeremy@silen-
tq.com",
                        "Jeremy",
"chicken");
    message.send(object);
    object = makeObject("miriam@silen-
tq.com",
                        "Miriam",
"milk");
    message.send(object);
  }

  /**
   * This creates the actual MessageOb-
ject
   * for each recipient, filling it
with
   * standard and custom values.
   */
  public static MessageObject makeObject
    (String to, String name, String
food) {
    MessageObject object = new Mes-
sageObject();
    Hashtable macros = new Hashtable();
    macros.put("$firstname", name);
    macros.put("$favoritefood", food);

    object.to = to;
    object.custom = macros;
    object.text =
      "Dear $firstname,\n" +
      "\n" +
      "We'd like you to know that " +
      "$favoritefood is on sale this
week!\n" +
      "\n" +
      "Sincerely,\n" +
      "The Management";

    object.subject = "SALE!";
    object.host = "pop.idiom.com";
    object.from = "rachel@silentq.com";

    return object;
  }
}
```

```
/*
 * MessageObject.java
 */

import java.util.Hashtable;

/**
 * This class holds message informa-
tion.
 *
 * @author Rachel Gollub (rachel@silen-
tq.com)
 */
public class MessageObject {
  String to;
  String from;
  String host;
  String subject;
  String text;
  Hashtable custom;
}
```

Listing 2

```
/*
 * MailPrinter.java
 */

import java.io.BufferedReader;
import java.io.InputStream;
import java.io.InputStreamReader;

import java.util.Enumeration;
import java.util.Properties;

import javax.mail.Address;
import javax.mail.FetchProfile;
import javax.mail.Flags;
import javax.mail.Flags.Flag;
import javax.mail.Folder;
import javax.mail.Header;
import javax.mail.Message;
import javax.mail.Multipart;
import javax.mail.Part;
import javax.mail.Session;
import javax.mail.Store;

/**
 * This class monitors a given mailbox
every 5
 * minutes, and prints the contents to
stdout.
 * It will also optionally delete mes-
sages it
 * reads.  More detailed comments are
provided in
 * the accompanying article.
 *
 * @author Rachel Gollub (rachel@silen-
tq.com)
 */
public class MailPrinter extends Thread
{

  public static void main(String
args[]) {
    // These args are <user>, <pass-
word>, <host>,
    // <delete?true:false>
    MailPrinter popThread =
      new MailPrinter(args[0], args[1],
                      args[2],
args[3]);
    popThread.start();
  }

  String protocol = "pop3";
  String host;
  String user;
  String password;
  boolean delete = false;

  /**
   * This just initializes the global
variables.
   */
  public MailPrinter(String pUser,
    String pPassword, String pHost,
    String pDelete) {
    user = pUser;
    password = pPassword;
    host = pHost;
    if
(pDelete.toLowerCase().equals("true"))
      delete = true;
  }

  /**
   * The run() method makes a mailbox
connection
   * every five minutes to check for
mail.
   */
  public void run() {
    while (true) {
 doStore();
      try {
 Thread.sleep(300000);
      } catch (Exception e) {
 e.printStackTrace();
      }
    }
  }

  public void doStore() {
    Properties lProperties =
      System.getProperties();

    Session lSession =
      Session.getDefaultInstance
        (lProperties, null);
    // Set debug here
    lSession.setDebug(false);

    Store store = null;
    Folder folder = null;
    try {
      store = lSession.getStore(proto-
col);
    } catch (Exception e) {
      System.out.println
        ("Couldn't get store from pro-
tocol.");
      e.printStackTrace();
    }
    // Connect
    try {
      if (host != null || user != null
   || password != null)
 store.connect
   (host, user, password);
      else
 store.connect();
    } catch (Exception e) {
      System.out.println("Unable to
connect.");
      e.printStackTrace();
    }

    // Open the Folder
    if (folder == null)
      try {
 folder = store.getDefaultFolder();
      } catch (Exception e) {
 System.out.println
   ("Couldn't get folder.");
 e.printStackTrace();
      }
    if (folder == null) {
      System.out.println("No default
folder");
    }

    // Get the default folder
    try {
      folder =
folder.getFolder("INBOX");
      if (folder == null) {
 System.out.println("Invalid folder");
```

Object International

www.oi.com

```
     }

     if (folder != null) {
folder.open(Folder.READ_WRITE);
int lTotalMessages =
  folder.getMessageCount();

if (lTotalMessages == 0) {
  System.out.println("Empty folder");
  folder.close(false);
  folder = null;
  store.close();
}
     }
   } catch (Exception e) {
     e.printStackTrace();
   }

   // Get and process messages
   if (folder != null) {
     try {
Message[] messages =
  folder.getMessages();

for (int i = 0;
  i < messages.length; i++) {
  if (filter(messages[i])) {
    if (delete)
      messages[i].setFlag
        (Flags.Flag.DELETED, true);
  }
}
folder.close(true);
store.close();
     } catch (Exception e) {
e.printStackTrace();
     }
   }
 }

 /**
  * The filter() method tries to fig-
ure out the
   * message type, and recursively
sends
   * multi-part parts through the fil-
ter.
   *
   * @param part The message part.
   *
   * @return True iff successful.
   */
  public boolean filter(Part part) {

    Object o = null;

    try {
      for (Enumeration e = part.getAll-
Headers();
        e.hasMoreElements(); ) {
        Header header =
          (Header) e.nextElement();
        System.out.println(header.get-
Name() +
          ": " + header.getValue());
      }
      System.out.println("\n");
      o = part.getContent();
    } catch (Exception e) {
      System.out.println
        ("Couldn't get content.");
      return false;
    }
    if (o instanceof String) {
      String lMessage = (String) o;
      System.out.println(lMessage);
      return true;
    } else if (o instanceof Multipart)
{
      Multipart lPart = (Multipart)o;
      try {
 int count = lPart.getCount();
 for (int i = 0; i < count; i++)
   return filter(lPart.getBodyPart(i));
      } catch (Exception e) {
 System.out.println
   ("Couldn't access parts.");
 e.printStackTrace();
      }
    } else if (o instanceof Message) {
      return filter((Part)o);
    } else if (o instanceof Input-
Stream) {
      BufferedReader lReader = new
BufferedReader
 (new InputStreamReader((InputStream)
o));
      int c;
      StringBuffer lBuffer = new
StringBuffer();
      try {
 while ((c = lReader.read()) != -1)
   lBuffer.append((char) c);
 lReader.close();
      } catch (Exception e) {
 System.out.println("Read error.");
 e.printStackTrace();
      }

System.out.println(lBuffer.toString());
      return true;
    }
    return false;
  }

}
```

Download the Code!

The code listing for this article can also be located at

www.JavaDevelopersJournal.com

When Bruce Scott (cofounder of ORACLE) started PointBase, Inc. he chose *Java Developer's Journal* as PointBase's exclusive advertising partner!

Base

JAVA DEVELOPER'S JOURNAL

We Know How to Create Success Stories!

VSI

www.vsi.com/breeze

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

JDJSTORE.COM GUARANTEED LOWEST PRICES!

Take a look at our specials this month!

EASTLAND DATA SYTEMS

Internet Shopping with Java Shopping Cart

…Described as the most progressive and interactive form of shopping on the web today…This Java Applet provides a complete user interface package for Internet Shopping Web Sites. Using Java technology we produce a drag-and-drop shopping user interface that is fun and easy to use, encouraging shoppers instead of frustrating them with confusing controls that are hard to follow. And the easier it is to shop, the more you sell. .... 294^{99}

Hybrid Shopping Cart

This Java Applet provides a complete user interface package for Internet Shopping Web Sites. A "Hybrid" is defined as an offspring of two varieties. A blending of the best features from our CGI and Java shopping products, we took the most powerful aspects of Java technology; real-time, on-screen updating and computational capabilities. And combined those with the most desirable features of our CGI shopping Cart, namely it's flexibility and compatibility with web designers with artistic talent. .... 294^{99}

CGI Shopping Cart

The Shopping Cart automates the Shopping Process to make shopping on your site intuitive, straight forward, and enjoyable! It's one of the most affordable Shopping Carts because it was designed for small businesses. Specifically for entrepreneurs who are testing the Internet waters, and can't or don't want to make large investments into bells and whistles for their site. But simply want to make shopping on their site easy for the customer. .... 294^{99}

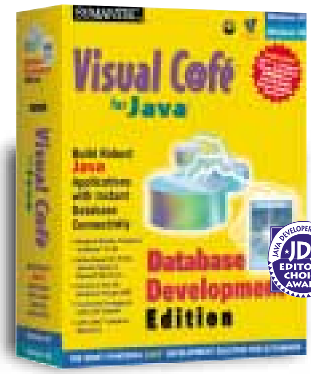
Guaranteed Best Prices

JDJ Store Guarantees the Best Prices.
If you see any of our products listed anywhere at a lower price, we'll match that price and still bring you the same quality service.

Terms of offer:
- Offer good through November 30, 1999
- Only applicable to pricing on current versions of software
- August issue prices only
- Offer does not apply towards errors in competitors' printed prices
- Subject to same terms and conditions

Prices subject to change.
Not responsible for typographical errors.

Attention Java Vendors:
To include your product in JDJStore.com, please contact jackie@sys-con.com



IBM's

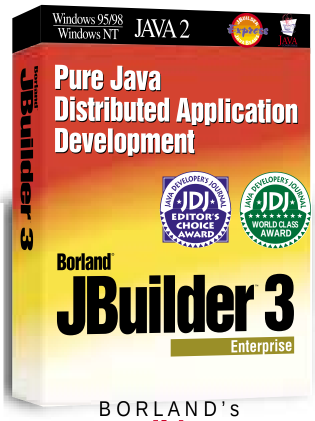
VisualAge for Java 2.0 Enterprise Edition

JDJStore.com Price . . **$2,498^{99}$**

COMPARE...

Programmer's Paradise Price .... $2,499^{00}$

Beyond.com Price................. $2,525^{00}$



SYMANTEC's

Visual Café Professional Edition

JDJStore.com Price . . **258^{99}**

COMPARE...

Programmer's Paradise Price .... 259^{95}

Beyond.com Price................ 299^{75}

BORLAND's

JBuilder 3 Professional New User

JDJStore.com Price . . **499^{99}**

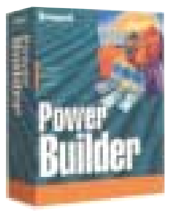
COMPARE...

Programmer's Paradise Price .... 545^{99}

Beyond.com Price................ 548^{99}

SYBASE

PowerBuilder Enterprise v.7

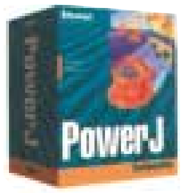


Release 7.0 of the market-leading enterprise development environment offers significant productivity enhancements and broad support for Web-based component standards. With the new HTML DataWindow, you can deploy to all major Web browsers. Tight integration in both development and deployment with EAServer offers highly competitive reliability, availability, and scalability (RAS) for PB applications developed for the Web.

PowerBuilder Enterprise v.7...................... 2789^{99}

SYBASE

PowerJ Enterprise



PowerJ provides a true end-to-end solution for building sophisticated Internet applications, exploiting the benefits of HTML, Java clients, and delivering powerful Java server-side components. PowerJ not only offers powerful database capabilities -- it also integrates seamlessly with Sybase Enterprise Application Server, enabling enterprise-class applications from creation, to testing and debugging, to deployment.

PowerJ .... 304^{99}

PROTOVIEW

JSuite v2.5

JSuite, a low priced bundle of four of the industry's leading JavaBeans component products. It includes: CalendarJ: Calendar Display Component. DataTableJ: The Fastest Grid Component On The Net! TreeViewJ: Feature-Rich TreeView Component. WinJ Component Library: A Series Of UI Components.

The JSuite comes with both JDK 1.02 Classes and JAR files with support for Java 2 and JDK 1.1.x Source Code is also available.

JSuite v2.5................................ 794^{99}

BLUE SKY SOFTWARE

RoboHELP Office 2000

Now You Can Create a High-Quality Custom JavaHelp System with RoboHELP Office. RoboHELP Office provides a user-friendly WYSIWYG authoring environment for creating JavaHelp. RoboHELP guides you through the process so you can create a great JavaHelp system - with point-and-click and drag-and-drop ease. Now you can create JavaHelp systems as easily as you create WinHelp, Microsoft HTML Help, and WebHelp (cross platform Help) from the same source product - all with RoboHELP Office.

RoboHELP Office 2000.......................... 888^{99}

ORDER TODAY! **888-303-JAVA**

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

BUY THOUSANDS OF PRODUCTS AT GUARANTEED LOWEST PRICES!

JDJStore.COM

GUARANTEED BEST PRICES FOR ALL YOUR JAVA-RELATED SOFTWARE NEEDS

SYBASE

SQL Anywhere Studio

JRun is the industry-leading tool for deploying server-side Java. JRun is an easy-to-use Web server "plugin" that allows you to deploy Java Servlets and JavaServer Pages. Servlets form the foundation for sophisticated server-side application development. Java servlets are platform independent, easy to develop, fast to deploy, and cost-effective to maintain.

SQL Anywhere Studio 1 user price $335.99
SQL Anywhere Studio 5 user price $849.99

KL GROUP

JProbe Suite

JProbe Profiler is the most powerful tool available for finding and eliminating performance bottlenecks in your Java code. JProbe Coverage makes it easy to locate individual lines of untested code and reports exactly how much of your Java code has been tested. JProbe Threadalyzer lets you pinpoint the cause of stalls and and deadlocks in your Java applications and makes it easy to predict race conditions that can corrupt application data.

JProbe Profiler w/Standard Support (includes JProbe Memory Debugger) $464.99
JProbe Suite w/Standard Support $934.99

INSTANTIATIONS

JOVE Starter Kit

The JOVE Super Optimizing Deployment Environment lets you create and deploy the world's fastest, most efficient Java applications. JOVE combines aggressive whole-program and object-oriented optimization technologies, native compilation, and a scaleable runtime architecture and deployment environment. The runtime system includes state-of-the-art multithreaded generational garbage collection, native multi-threading, low overhead polymorphism, and a number of other incredible technology pieces. As a result of all this technology, JOVE enables the creation of very high performance, robust executable files for the deployment of very large, complex Java applications.

JOVE Super Optimizing Deployment Environment.... $4494.99

ORACLE

JDeveloper Suite Personal Edition

Oracle JDeveloper Suite 2.0 provides a complete Java development environment for developing and deploying applications ranging from Java and HTML clients to server-based business components for the Internet computing platform. JDeveloper 2.0 supports application development based on industry-standard Enterprise JavaBeans (EJB) and CORBA component models. Using intuitive and easy-to-use wizards, the product provides full support for a rich set of Java standards – servlet, JDBC, SQLJ, InfoBus and JFC/Swing. JDeveloper 2.0 comes with a servlet engine that allows servlets to be developed and tested within JDeveloper.

JDeveloper Suite Personal Edition............ $119.99

IBM

WebSphere Application Server

Whether it's business integration, Web self-service or e-commerce, IBM WebSphere Application Server provides the software and tools for building and deploying Web-based applications. With Standard, Advanced and Enterprise editions, Application Server supports all your e-business needs, from simple Web transaction processing to enterprise-wide Web applications. An open, extensible solution providing the highest levels of performance, security, availability and scalability, WebSphere Application Server leverages your existing IT investment to create new opportunities - now and into the future

WebSphere Application Server Standard Edition v2.02 .... $668.99

INSTALLSHIELD

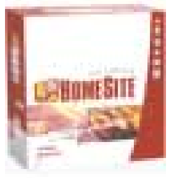
InstallShield Java Edtion 2.5

InstallShield Java Edition 2.5 is the powerful tool developers require to produce bulletproof InstallShield installations with Java versatility. You can target your application for multiple systems with cross-platform distribution. And InstallShield Java Edition 2.5 offers the key features and functionality designed to let developers go further in distribution and deployment.

InstallShield Java Edition $474.99

ALLAIRE

HomeSite 4.0



HomeSite is the award-winning HTML editing tool that lets you build great Web sites in less time, while maintaining Pure HTML. Unlike WYSIWYG authoring tools, HomeSite gives you precise layout control, total design flexibility and full access to the latest Web technologies, such as DHTML, SMIL, Cascading Style Sheets and JavaScript. HomeSite 4.0 is the only HTML editor featuring a visual development environment that preserves code integrity.

HomeSite 4.0 .... $87.99

ALIVE.COM

Alive e-Show 1.1

Alive e-Show is an easy, cost effective desktop application that allows non-technical PC users to create and publish streaming media e-shows to the Web. Using Alive e-Show, employees can create new content, integrate existing media, and publish it to the Web with a single click. It's a great way to integrate slides, digital photography, animation, synchronize audio and video, add hyperlinks, and supplement with closed captions. Create e-shows quickly and easily. You create slides in outline mode, record audio and video, and format the e-show using predefined templates with slide styles and placeholders for closed captions too. Allaire HomeSite® is included free, further simplifying any extra customization you may need using this HTML editor.

Alive e-Show 1.1 .... $558.99

INTUITIVE SYSTEMS

Optimizeit 3.0 Professional

Optimize It!

Optimizeit provides a powerful solution for developers looking to rapidly track down and fix performance issues in any Java programs. Optimizeit offers complete profiling capabilities including memory leak debugging, CPU profiling and monitoring of object allocations. Easy to use, comprehensive and accurate, Optimizeit provides real-time thorough information about CPU and memory usage, presented in easy-to-read graphs and relevant metrics. Using Optimizeit, developers get complete, coherent information on any part of their program down to the responsible line of source code.

Optimizeit 3.0 Professional.................. $448.99

GALILEO DEVELOPMENT SYSTEMS

Intr@Vision Foundation

Intr@Vision Foundation helps bring ColdFusion development to the next level. It provides an out-of-the-box application security architecture for handling your most complex intranet and extranet needs. Instead of spending 30% of your development time adding security to every application you build, it gives you a proven solution with a single line of code. Intr@Vision Foundation allows your developers to focus on building business solutions, not infrastructure.

Intr@Vision Foundation $3499.99

ALLAIRE

ColdFusion 4.0

With ColdFusion 4.0, create Web applications for self-service HR solutions, online stores, interactive publishing and much more. The integrated development environment that has all the visual tools you need to create Web applications quickly and easily. From simple to sophisticated ColdFusion gives you the power to deliver the Web solutions you need-faster, and at a lower cost.

ColdFusion 4.0 .... $354.99
SkillBuilding with ColdFuion Interactive Training CD.... $284.99

CLOUDSCAPE INC.

Cloudscape Single User Developer License

Building on its technology lead as the industry's first embeddable Java database designed for distributed, off-line, and mobile computing, Cloudscape Release 2.0 is designed to support eBusiness applications such as "smart" eCatalogs and supply chains. Enhancements in 2.0 include: *Multi-user concurrency that supports hundreds of users. *Advanced security that enables only authorized applications to read data. *Substantially faster load of initial information into the database Cloudscape 2.0 is bundled with Cloudview (a schema browser utility. Includes one year of support from the Cloudscape services organization.

Cloudscape Single User Developer License..... $894.99

WWW.JDJSTORE.COM

Enterprise Java at Syracuse University

Distributed applications written in Java using the EJB model to build reliable and scalable systems

WRITTEN BY
FRANK GATES

Today's universities are recognizing the desirability of providing many staff functions for their students through a Web interface. Self-service applications allow students to enroll in courses, manage their personal information and examine their class schedules – and save the university time and expense for staff that would otherwise perform these tasks. In April 1999 New York State's Syracuse University went online to accomplish these activities over the Internet with a self-service application based on Enterprise JavaBeans technology. Here's how it was done.

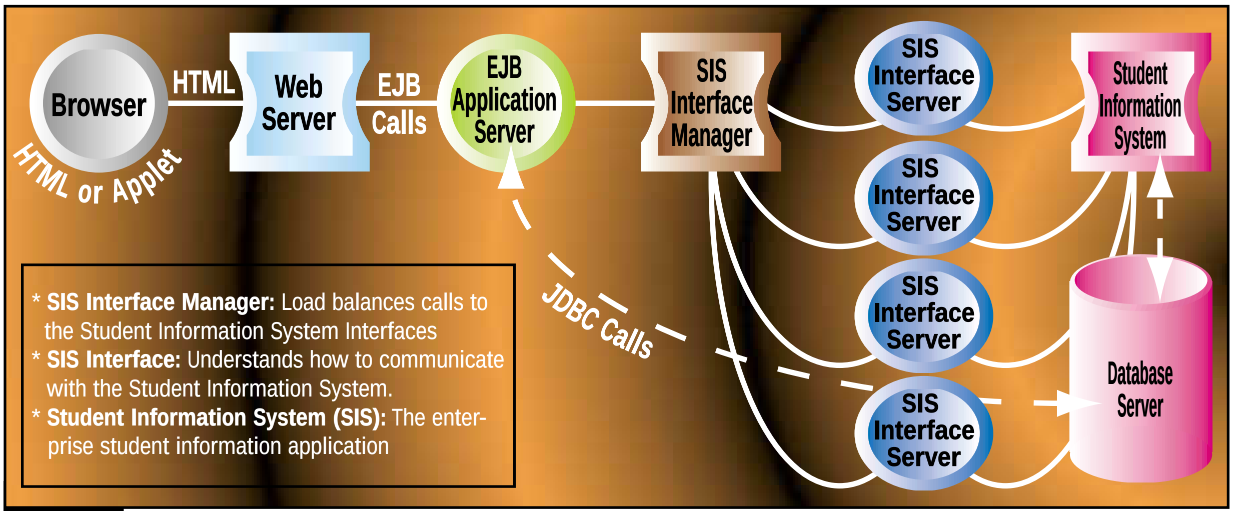

Browser
HTML
Web Server
EJB Calls
EJB Application Server
SIS Interface Manager
SIS Interface Server
SIS Interface Server
SIS Interface Server
SIS Interface Server
Student Information System
Database Server
HTML or Applet
JDBC Calls
* SIS Interface Manager: Load balances calls to the Student Information System Interfaces
* SIS Interface: Understands how to communicate with the Student Information System.
* Student Information System (SIS): The enterprise student information application

FIGURE 1: Online student information environment

The Requirements

Syracuse University had bought a new online student information system (SIS) from a leading software vendor. Unfortunately, the system didn't provide an adequate Web interface, and it was built from the point of view of administrators, not students, complicating how a Web-based solution could be developed.

The university sought a Web-based student self-service interface for its new system and had an extensive list of requirements for the application. The interface had to be easy to use and responsive over modem connections, and it had to work with all popular browsers. The application had to be platform independent and use distributed, scalable technology. It also had to be highly reliable as it would be running 24/7. Further, the application had to be implemented using standards-based technology such as EJB. Most important, the Web interface had to be capable of reliably enrolling all of Syracuse University's 15,000 students during its open enrollment period.

Syracuse also required an application vendor that would provide first-rate technical support.

It became clear that development of an application that adapted the SIS to the Web would require a substantial effort. A clear separation between the business logic and presentation layer would be needed to enable customizations without affecting the application logic. EJB promotes this separation. In addition, standard Java-based technologies such as Java Servlets and Java Server Pages (JSP) would be needed to write the presentation layer. As Java is platform independent (particularly on the server) and largely vendor independent, an application based on standard Java technologies would be the best fit for this distributed Web application.

In September 1998 Syracuse selected Student Affairs, a Web application from Interactive Business Solutions (IBS), to fulfill their requirements. This application provides a scalable Web-based solution through the use of servers that are specialized in communicating with the SIS used at Syracuse. Scalability is achieved through its multilayer architecture, where each layer fulfills a specific responsibility.

Student Affairs' Application Architecture

The application architecture developed by IBS can be used to develop new applications and to adapt, as Student Affairs does, to third-party applications. The outermost layer is the application user interface. For simplicity, the user interface is a commonly available browser (Internet Explorer 3.0 or later and Netscape 3.0 or later). The presentation layer consists of servlets running on a servlet-enabled Web server. This architecture also supports the use of JSP and HTML/Java (JHTML). Applets can also be used. (Student Affairs uses one small applet to facilitate the enrollment process.)

Student Affairs' servlets constitute a very thin presentation layer for the SIS data, which allows the user interface designer to focus on designing displays that communicate SIS content to the

MetaMata

www.metamata.com

students. The resulting presentation layer can be a rich user interface that guides students seamlessly through the enrollment process yet is not overly concerned with how the enrollment is accomplished.

The EJB Application Server is responsible for executing all the business logic of an application. In the case of this application, however, the SIS contains the business logic; the application's EJB is a thin layer that makes calls to the SIS via the SIS Interface Manager with parameters it receives from the application servlets. In some cases EJB makes direct read-only SQL calls to the SIS's database, either to increase performance or simply to access data that may not be available through the SIS interfaces. The results are returned to the servlet, which then formats and displays the results.

The EJB Application Server communicates with the SIS through two additional layers. EJB communicates directly with the SIS Interface Manager, which load-balances calls from the EJBs to multiple SIS Interface Servers that perform the actual API calls to the SIS.

With the exception of the user interface layer, which is an Internet browser, a specific server manages each layer in the application. For instance, the presentation layer runs on a Web server, and the business logic is in EJBs on the EJB Application Server. These servers can operate on separate computers, and there can be multiple servers for each layer. This obviously increases performance and reliability. Because the servers don't have to run on the same kind of computer (thanks to Java), a university can choose the equipment that's best for each server.

Syracuse and Student Affairs

Student Affairs was installed in October 1998. Syracuse chose to install the Web server, EJB Application Server, SIS Interface Manager and SIS Interface Servers on separate multiprocessor NT servers. While the Web server and EJB Application Server could be installed on any computer that supports Java, the SIS Interface Manager and SIS Interface Servers required installation on NT servers due to the SIS API server requirements.

During the six months before Syracuse did its first online registration, the university spent its time customizing the appearance of the application and testing the application components.

Because Student Affairs is designed to function at any university using the student information system, it has a large set of items that are customizable, including color preferences for pages, links, text and tables. The banner title and logo are also customizable, and a university can readily customize the footer of each page in the application. Each item can be modified without changing any source code.

Syracuse University modified the appearance of the Web application extensively, supplying the university's logo, using school colors for the menu buttons and providing thorough informational text for the customized footers on each page. The university also produced a companion Web site that provided online instruction on how to use the new application.

"The cooperative spirit between the university and IBS accelerated the testing process and significantly improved the product."

The university's tests were in two general arenas: functional testing of the application and performance testing. During the former they sought to determine whether the application worked (a validation test); during the latter, whether it worked in a desirable manner (a usability test). Any bugs were reported to IBS.

Usability issues were also reported. Items judged useful for any university were performed at no charge; items deemed unique to Syracuse were considered chargeable and were performed with the university's approval. All work items – bug fixes and usability items – were then scheduled for repair according to how critical the defect was.

Syracuse also spent several months doing performance testing of the application components, which demonstrated that testing in a production-like environment is essential. The SIS was the first to be tested. The rationale was that if there was a bottleneck in the back end, testing from the Web server could hide or exacerbate the problem. Back-end testing exercises the SIS, the database server and the IBS SIS Interface Servers.

Back-end testing simulated thousands of concurrent enrollments per hour. The system was severely overtaxed and performed poorly. Initially, the SIS Interface Servers weren't running in a full production environment. Additional computers were installed, and the tests were repeated. No further problems were discovered with the SIS Interface Servers during these tests. However, intermittent communication problems between the SIS Interface Server and the SIS Interface Manager were discovered independently by IBS, and the servers were updated at Syracuse.

The SIS and the database servers were still underperforming during these tests. Syracuse spent two months testing and fine-tuning these servers to achieve adequate performance. Once sufficient performance began to be achieved by the back end, front-end testing of the Web server and EJB Application Server was started. Several problems were discovered and remedied at this time.

- The heap size of the Web server and EJB Application Server needed to be expanded. The default maximum heap size, 16MB, was expanded to 128MB.
- Following one update of the EJB Application Server, the server was found to be leaking memory, which it had not been doing before. A seemingly minor change in the server uncovered a memory problem in which circular references between different EJB components prevented garbage collection of EJBs and many resource classes. The server logic was corrected and a new update was quickly made.
- Two deadlocks were discovered between threads competing for server or EJB resources.
- Unlimited users would saturate the Web and EJB servers. Configurable limits were added to control the number of users accessing the system.

Summary

The Syracuse University experience successfully demonstrates that distributed applications written in Java using the Enterprise JavaBeans model can build reliable and scalable systems. The multilayer EJB architecture was essential in enabling the university to go online on time. In addition, the cooperative spirit between the university and IBS accelerated the testing process and significantly improved the product.

fgates@interactivebusiness.com

AUTHOR BIO

Frank Gates, a software developer for 20 years, currently works for Interactive Business Solutions. The senior consultant on the team that developed the IBS Enterprise Application Server and the Student Affairs application on which this article is based, he is also on the team developing the next version of the EAS.

Riverton

Object

www.objcetdes

Design

sign.com/javlin

A Return to Reflection

Adding advanced features to the CodeDocument class à la Borland's CodeInsight

WIDGET FACTORY

WRITTEN BY
JIM CRAFTON

When we last talked, I promised to finish up the CodeDocument class I'd so abruptly left behind back in July (***JDJ*** Vol. 4, issue 7). Now, due to millions of desperate letters from fans around the globe, I've decided to finish off the series in this article, tackling reflection once again and ending with a text component that supports syntax highlighting and a simplified version of something similar to Borland's Code Insight. Along the way we'll see a few more tricks that JTextArea can do, and in general we'll just go hog wild in code!

A Refresher on Reflection

Those of you who read my last article can just skim through this part. For the others I'll try to explain, quickly, what reflection is and why we need it for this article.

Reflection allows object A, which is interested in object B, to interogate object B for any information object A is interested in at runtime – without having the source code on the machine. Interested in what attributes the object has? No problem, just use reflection. Wondering if the object has a particular method? Again, no problem, just use reflection. Now that you know the object has a particular method, you want to know what parameters the method requires. Just use reflection! Now you know what the parameters are for the method, but you want to know what your spouse is thinking? Just use...oh, wait, that doesn't work.

Well, anyway, reflection is one of the cooler features in the Java language (and oh, how I wish C++ had the same thing!). Used properly it offers some great benefits. Obviously, for development tools that need to know all sorts of details about the objects being manipulated, this is great. Other uses might be to generate SQL automatically based on some metadata scheme (this was a technique one of the developers used in a project I was on recently). When you use tools like Borland's JBuilder, the ability of the Inspector to "know" exactly what properties to display for a given UI component is based completely on reflection. So how do we use it? To start, you need to get the Class object from whatever object you're interested in. Let's take a look at the code below.

```
Vector v = new Vector(); //this is the object we want to
                         //learn more about through
                         //reflection
Class aClass = v.getClass();
```

Any object could have been chosen (Vector, JComponent or any other valid Java object), and once we have a valid instance we call the getClass() method to get the object's Class instance. The Class object serves as the starting point for obtaining all sorts of information, such as all the constructors for that object or all the declared methods – or even all the fields (attributes) of the object. Let's look at what we do once we get a method from the list of declared methods for the object:

```
Method[] methods = aClass.getDeclaredMethods();
for (int i=0;i<methods.length;i++){
  Method aMethod = methods[i];
  System.out.println( aMethod.getName() );
}
```

All we have to do, once we have a Class object, is call the getDeclaredMethods() function, which returns an array containing all the declared methods of the class (this includes methods declared as private!). Once we have this array we can iterate through all of the elements in the array calling the Method class's getName() method. If we're interested in determining the accessibility of a particular method, we can use the getModifiers() function:

```
int mods = aMethod.getModifiers();
System.out.println( Modifier.toString(mods) + " " +
aMethod.getName() );
```

If we're interested in the parameters a method took, we can do the following, using the Method class's getParameters(), which returns an array of Class objects representing the arguments to the method:

```
Class[] params =  aMethod.getParameters();
```

We can also invoke methods dynamically on an object by using reflection. First we retrieve the desired method as we did above or by using the Class class's getDeclaredMethod (which is useful if you already know the name and parameters of a particular kind of method – for example, say you were trying to determine if a series of objects has a method called "setText" that takes a single parameter of type String). Then we create an array of Object and call the Method class's invoke() method.

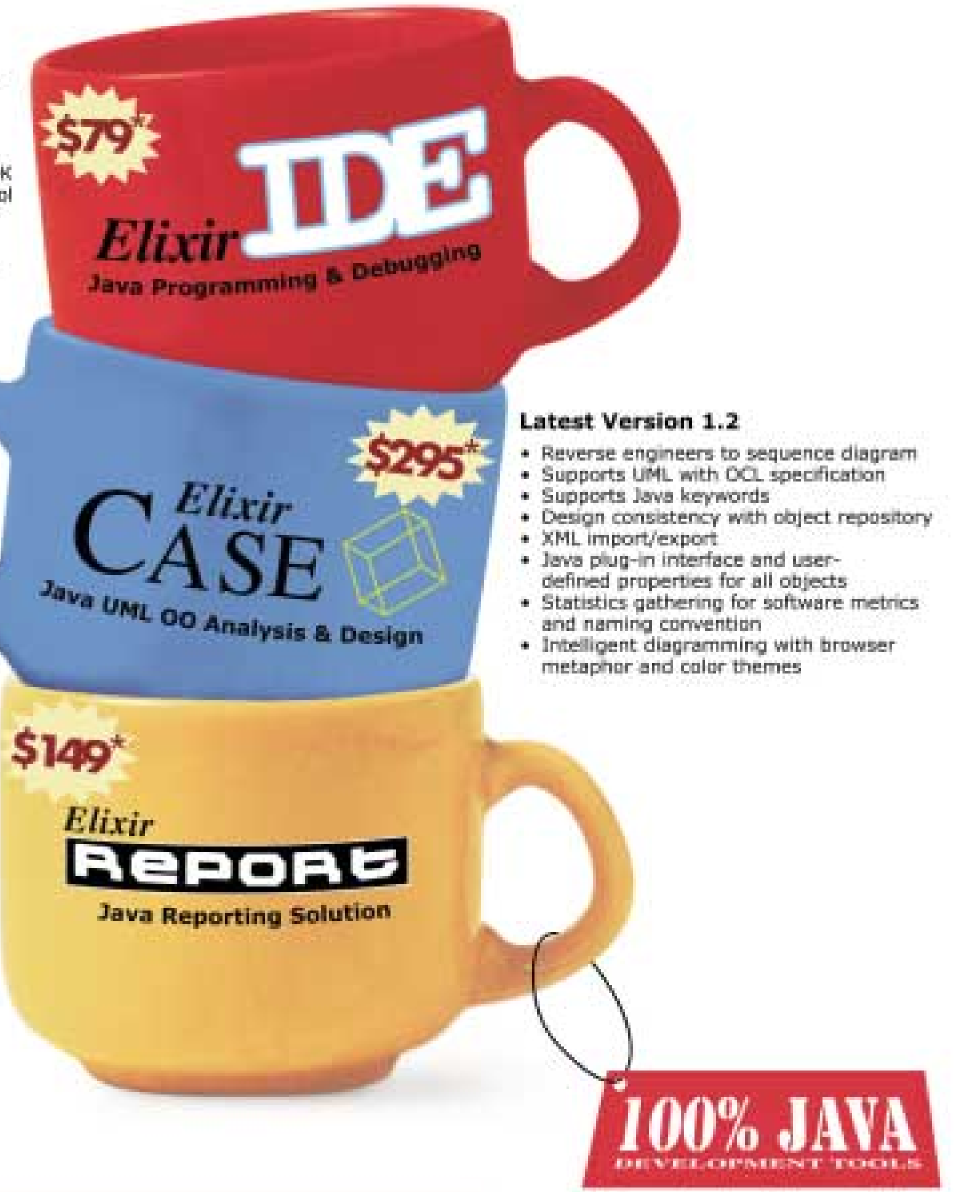
```
Class[] argParams = new Class[1];
argParams[0] = Object.class;
aMethod = aClass.getDeclaredMethod( "addElement", arg-
Params );

Object[] argVals = new Object[1];
argVals[0] = new String( "A New String Object !" );
aMethod.invoke( v, argVals );
```

The first part of the code gets a single method from the aClass variable and then prepares to invoke the method. The argVals array has its first element set to a new instance of a String, and then the invoke method is called: the first parameter is the object that the method is being invoked on (in this case our Vector variable from before), and the second parameter is the object array of method arguments (methods that take no arguments could just pass in null).

So we've looked very quickly at a simple example of how to retrieve information about an object, all of it done at runtime, using reflection. Remember to import the java.lang.reflect package into your code to do your experiments. Now we'll move on to the CodeDocument and how we'll start hooking this all together.

Three Cheers!

All Elixir Java™ development tools are coded entirely in Java. We believe developers should be able to build, rebuild and debug their application on any machine they intend to deploy on. If a problem is encountered on Solaris, Macintosh or Linux, why be forced back to Windows to correct the problem?



Latest Version 2.2

- Multi-threaded Java 2 JPDA debugger
- Build cycle automation including preprocessing, obfuscation, JAR-ing
- Incremental obfuscation
- Multiple compilers (JDK 1.x, Java 2, JDK 1.3B, Jikes), scripting and version control
- Syntax coloring for Java, XML, C++, HTML, and Scheme
- Open file formats in plain-text or XML
- Java plug-ins, class inspector, etc

Latest Version 1.2

- Reverse engineers to sequence diagram
- Supports UML with OCL specification
- Supports Java keywords
- Design consistency with object repository
- XML import/export
- Java plug-in interface and user-defined properties for all objects
- Statistics gathering for software metrics and naming convention
- Intelligent diagramming with browser metaphor and color themes

Latest Version 1.1

- Royalty-free report runtime engine
- Multiple data source types (Java objects, JDBC, text, crosstab)
- Applet reporting
- Dynamic image loading
- WYSIWYG report designer
- Wizards for report and data source
- Project manager, scripting, dynamic user interaction in reporting

* 1 license per developer for all supported platforms

EXPRESS DELIVERY

of Latest Java Technologies

Do your tools give you access to the latest Java technologies, or are you stuck with last year's JDK along with all the bugs? Early access products may just give you the competitive advantage you need.
Elixir tools have consistently provided earliest access to the latest Java advances including **JFC/Swing, Java 2, JPDA, Blackdown JDK 1.2 pre-releases, and JDK 1.3Beta.**

UNLIMITED FREE TASTING

of Lite Versions

Drink all you can! No time limit, no registration required! Just less files you can open, create, or print. Check it out at

http://www.elixirtech.com/download/.

Elixir Technology

http://www.elixirtech.com/

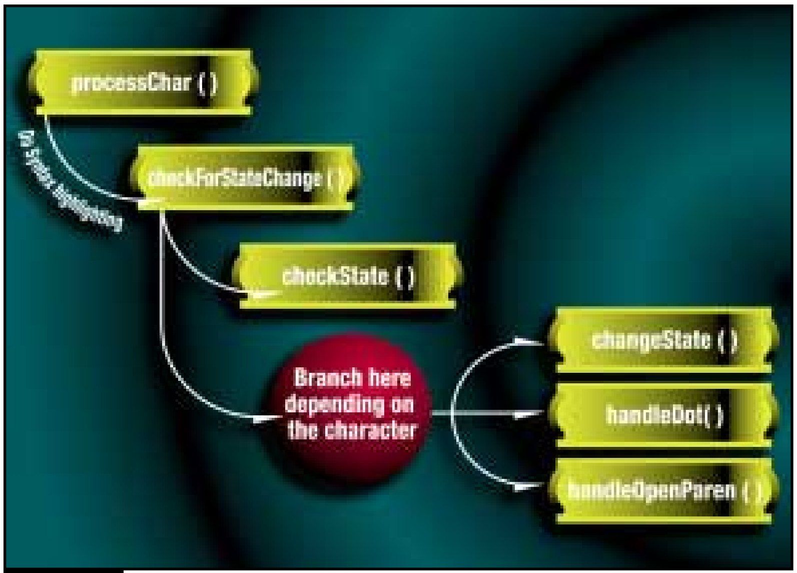


FIGURE 1

When a State Isn't a State (and Other Dumb Plays on Words...)

For the CodeDocument to do its work it has to know its current state – in other words, not only what word you've just typed in (or inserted), but whether or not you're just typing in the package name, typing in imports, creating a class declaration or creating variables in code within a method. The document also has to know what type the variable is as well as whether you've worked with it before so it can look it up if necessary. Obviously, this could get quite complex, and to make this function as a production tool you'd end up with a sophisticated state machine and parser to work all this out. In the interest of time (yours) and energy (mine), I'm going to develop a simplified version of this that should serve as a possible model or at least inspire you to feats greater than mine.

So, back to State. To keep track of the state of what you're typing in, I'll introduce several new constants as well as a number of new variables to keep track of it all.

To save the information for future use, we'll create several small classes to hold the information for us, namely, class CodePackage to hold information about the package just created; class CodeClass for the outer public class that's normally created; and class ClassElement, which will hold all the dynamically discovered fields and methods we display in our drop-down listbox. To simplify all the reflection code we'll use a class called ClassLister, which will do all the reflection work whenever we encounter a possible need to get at the information.

The following code shows the possible states we'll keep track of in the CodeDocument:

```
private static final int STATE_TEXT_INPUT = 10;
private static final int STATE_CLASS_INPUT = 11;
private static final int STATE_VARIABLE_INPUT = 12;
private static final int STATE_PACKAGE_INPUT = 13;
private static final int STATE_IMPORT_INPUT = 14;
private static final int STATE_VARIABLE_TYPE_INPUT = 15;
```

The generic default state (STATE_TEXT_INPUT) represents typing in text, as I'm doing right now. STATE_PACKAGE_INPUT defines when the CodeDocument determines that the user is typing in a package name. This is needed for the reflection to pick classes just created in the package. STATE_IMPORT_INPUT is used to describe when the user is typing in the name of an imported package, like "java.lang.reflect.*", for instance. STATE_CLASS_INPUT signifies the creation of the new outer class. STATE_VARIABLE_TYPE_INPUT and STATE_VARIABLE_INPUT are used to determine when a variable or variable type is entered. As the user types, the state will also be changed, which is handled by the checkForStateChange() method inside the processChar() method. Inside checkForStateChange() the method looks at what kind of character has been passed in and then makes a call to checkState(), which either ouputs diagnostic information or adds the appropriate data to model being built up. Then checkForStateChange() calls the changeState() method, which actually changes the CodeDocument's state. Depending on the character passed in, checkForStateChange() may also call other methods to handle things like a "." or a "(", both of which can cause reflection to take place. Figure 1 shows this in diagrammatic form.

The handleDot() and handleOpenParen() methods will be discussed in more detail in the next section, as these are responsible for triggering events.

Events

The CodeDocument now both triggers and listens to events in this new version. Every time the handleOpenParen() method is called, the MethodEvent is fired, via the fireMethodEvent method. This allows outside controls to be notified whenever the CodeDocument is ready to display the arguments of a method, similar to what happens in JBuilder when you type in a method and then type the "(" character and the tool tip pops up with all the method's arguments. Outside controls interested in this event can use the CodeDocument's addMethodListener() method to register themselves as listeners to the documents event, assuming they implement the MethodListener interface given in Listing 1.

The document also listens to events, which is how the drop-down list with the available methods for a given object works. If the handleDot() method is called, the CodeDocument gets the current position and extracts the appropriate text (see the earlier ***JDJ*** articles pertaining to the CodeDocument for exactly how this is done) until the class name is found for the variable in question. This is currently the Achilles heel of the whole thing, and those of you interested in making this more sophisticated will want to start improving this part, but it works for our purposes. Once the class name is found, a new set of attributes is created and the classLister attribute's setClassName() method is called, which sets the classLister's class name attribute. To retrieve all the info, the method then calls the classLister's listAll() method, which returns a Vector containing a list of ClassElement objects (which are simply convenient ways to store all the method or attribute data).

Now that all the data is put together, we can go ahead and create our drop-down list. But wait, you say. Where on earth are you going to put it? And how? Patience, my young one, all in good time!

One of the neat features of the Document model class is the ability to insert not only text of different formats, but also visual components, like drop-down listboxes. To do this we use the SimpleAttributeSet class and create a new instance. Then, using the StyleConstants class's static method setComponent(), we pass in our newly created list box and the attribute set we just created. Let's look at the code in Listing 2.

The setComponent() method attaches the newly created JComboBox to inputAttributes object. To make this show in the document, we use the insertString() method at the current position plus one, with a space (" "), and passing in the attribute set. Notice that we're calling the super classes insertString() method. Otherwise we'd end up cycling through all our code, which we don't need to do at this point.

Force 5

www.force5.com

The other thing that's important is to register the CodeDocument as a listener to the JcomboBox. This is done so the CodeDocument knows when the user has selected an item from the list and can safely remove the JComboBox and insert the selected text back into the document. Which is why, as you'll see in Listing 3, the declaration for the CodeDocument class has also changed, and yet another method has been added.

The ClassElement stores the information about any of the methods or attributes in the list. The Element value, which stores the actual name of the attribute or method, is retrieved. After this, the combo box is removed and the string retrieved from the ClassElement is put in its place.

The ClassLister

The ClassLister class is at the heart of all the reflection that goes on in the CodeDocument. Its primary method is the listAll() method, which uses the ClassLister currentClassName attribute to attempt to return a Vector containing a complete listing of all methods and attributes belonging to the class named from the currentClassName attribute. The first thing the listAll() method tries to do is obtain a Class object from the currentClassName String. This is done via the Class class's static method classForName(), which takes a fully qualified class name and attempts to return a valid Class instance. A fully qualified classname would be something like "java.util.Hashtable" or "com.sun.java.swing.JListBox". Since the String parameter to the listAll method may not be a fully qualified name, the ClassLister keeps track of a list of packages it knows about. Thus, if the first try of the class match fails, the listAll() method catches the exception and tries prepending the package names it knows about to the currentClassName until a Class instance is successfully created or it runs out of package names, in which case the method fails. If the method does create a Class instance, it uses reflection, as discussed above, to add all the method and attribute names to a Vector, packaging each method or attribute into a ClassElement object that is added to the Vector list and then returned.

ClassLister has several other methods:

- *listMethods()*, which does something similar to listAll(), except it only returns methods
- *listMethodArgs()*, which, given a method name, tries to list out the arguments for the method
- *isNameAClass()*, which returns true if the supplied String is a class (like listAll(), the String passed in to isNameAClass() does not have to be a fully qualified class name), or false if the String is not a class name.
- *addPackage*, which adds a package name to its internal list (a Vector) of known packages

Tying It All Together

Let's make a simple test application to try this out. Listing 4 shows how it works. Although it looks a bit long, it's actually pretty simple. The main() creates new instances for a frame (class TestFrame – we'll get to that in just a bit), an editor (the infamous JTextArea component) and the doc (our beloved CodeDocument). A keywords list, of type Vector, is created and all the Java keywords are added to it (this is what takes up the bulk of code). The CodeDocument setKeywords() method is called to set the CodeDocument's keyword list. The frame is then registered as a listener to the CodeDocument by the addMethodListener() method. The editor's Document model is set with a call to setDocument(), which sets the CodeDocument as the editor's Document model. The frame size is then set and...we're done!

If you run this, remember to type a few nonkeyword characters (like "//" followed by the actual text). Otherwise you'll encounter a known bug, which can be looked up on the Swing Web site (for more information look at http://developer.java.sun.com/developer/bugParade/bugs/4128967.html).

Conclusion

We've now covered how to make our CodeDocument, highlight use, defined syntax, control, numbers, strings and comments. Along the way we've added code completion (à la JBuilder's Code Insight) and method hints using the magic of Java's reflection classes. We've learned how to use the power of reflection to build the start of a really useful tool as well. I hope you found this as interesting a topic as I did !

Author Bio

Jim Crafton is part of the research and development team at Improv Technologies (www.improv-tech.com), helping to develop a new production-quality animation tool. He also develops advanced graphics software that can be seen at www.one.net/~ddiego/.

ddiego@one.net

Listing 1

```
public class MyLabel extends JLabel implements MethodListen-
er{
  ...//code
  public void methodAction(ActionEvent e){
    this.setText( e.getActionCommand() );
  }
}

...//more code
MyLabel aLabel = new MyLabel();

JTextArea editor = new JTextArea( new CodeDocument() );
CodeDocument doc =  (CodeDocument)control.getDocument();
doc.addMethodListener(aLabel);
```

Listing 2

```
SimpleAttributeSet inputAttributes = new SimpleAttribute-
Set();
..//more code...
try{
  JComboBox c = new JComboBox();
  c.addActionListener(this);

  StyleConstants.setComponent(inputAttributes, c);
  super.insertString(currentPos + 1, " ", inputAttributes);
}
catch (Exception ex){
  ex.printStackTrace();
}
```

Listing 3

```
public class CodeDocument extends DefaultStyledDocument
implements ActionListener{

..//more code

  public void actionPerformed(ActionEvent e) {
    try{
      JComboBox comp = (JComboBox)e.getSource();
      ClassElement elem = (ClassElement) comp.getSelect-
      edItem();
      String item = elem.getElementValue();
      this.remove(listOffs, 1);
      super.insertString(listOffs, item, null);
      currentMethod = item;
    }
    catch (Exception ex){
      ex.printStackTrace();
    }
  }
```

American Cybernetics

www.multiedit.com

Listing 4

```
package CodeEditor;

import com.sun.java.swing.*;
import java.util.*;
import java.awt.event.*;
import java.awt.*;

public class Application1 {


 public static void main(String[] args) {
  TestFrame frame = new TestFrame();
  TextPane editor = new JTextPane();
  CodeDocument doc = new CodeDocument();

  Vector keywords = new Vector();

  keywords.addElement("abstract");
  keywords.addElement("boolean");
  keywords.addElement("break");
  keywords.addElement("byte");
  keywords.addElement("byvalue");
  keywords.addElement("case");
  keywords.addElement("cast");
  keywords.addElement("catch");
  keywords.addElement("char");
  keywords.addElement("class");
  keywords.addElement("const");
  keywords.addElement("continue");
  keywords.addElement("default");
  keywords.addElement("do");
  keywords.addElement("double");
  keywords.addElement("extends");
  keywords.addElement("else");
  keywords.addElement("false");
  keywords.addElement("final");
  keywords.addElement("finally");
  keywords.addElement("float");
  keywords.addElement("for");
  keywords.addElement("future");
  keywords.addElement("generic");
  keywords.addElement("if");
  keywords.addElement("implements");
  keywords.addElement("import");
  keywords.addElement("inner");
  keywords.addElement("instanceof");
  keywords.addElement("int");
  keywords.addElement("interface");
  keywords.addElement("long");
  keywords.addElement("native");
  keywords.addElement("new");
  keywords.addElement("null");
  keywords.addElement("operator");
  keywords.addElement("outer");
  keywords.addElement("package");
  keywords.addElement("private");
  keywords.addElement("protected");
  keywords.addElement("public");
  keywords.addElement("rest");
  keywords.addElement("return");
  keywords.addElement("short");
  keywords.addElement("static");
  keywords.addElement("super");
  keywords.addElement("switch");
  keywords.addElement("synchronized");
  keywords.addElement("this");
  keywords.addElement("throw");
  keywords.addElement("throws");
  keywords.addElement("transient");
  keywords.addElement("true");
  keywords.addElement("try");
  keywords.addElement("var");
  keywords.addElement("void");
  keywords.addElement("volatile");
  keywords.addElement("while");

  doc.setKeywords(keywords);
  doc.addMethodListener(frame);

  editor.setDocument(doc);

  frame.setSize(400,400);
  frame.getContentPane().add(editor);
  frame.show();
 }
}

 class TestFrame extends JFrame imple-
 ments MethodListener{
  public void methodAction(ActionEvent
  e){
    this.setTitle( e.getActionCommand()
    );
  }
 }
```

Download the Code!

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

SlangSoft

www.slangsoft.com

Insignia

www.insignia.com

The Business Advantage of EJB PART 1

Developing portable EJB applications

WRITTEN BY
JASON WESTRA

What's all this hype about portability? Portability has been a hot topic since Java's arrival just a few years ago, so I'm going to devote some space toward understanding portability issues centered around the Enterprise JavaBeans architecture and development. This month I'll discuss the various types of portability and Java's relationship to each; then I'll touch on the portability goals of the EJB specification and where EJB portability lacks maturity (and why not to worry). Next month I'll provide tips on EJB portability as well as code examples depicting how you can help achieve the promise of EJB portability through solid design and coding practices.

Competitive Advantage of Portability

Portability is a true concern in the minds of many IT executives who see the need to develop applications with faster times-to-market and across more platforms than ever before. The portability hype is fostered by the need to bring new products to market at ever-increasing rates – products that are as easily compatible with legacy systems as with Web-enabled systems. This mission is heavily reliant on the interoperability of software across a number of areas, including platform/operating systems, resources such as databases and transaction management services, and multiple development languages/component models – most recently, Enterprise JavaBeans components.

Types of Portability

As previously mentioned, there are different types of portability. Applications that are portable across these areas can be quickly modified to meet changes in your business or technology and will be more easily maintainable than nonportable, "stovepipe" applications. For instance, rather than making upgrades to multiple versions of an application for each specific platform, one upgrade is made to one application.

Depending on the focus of your software, you may worry about one or two types of portability while shifting the burden of the others to a middleware provider. Let's take a look at each type of portability and understand which ones will directly impact a Java/EJB developer.

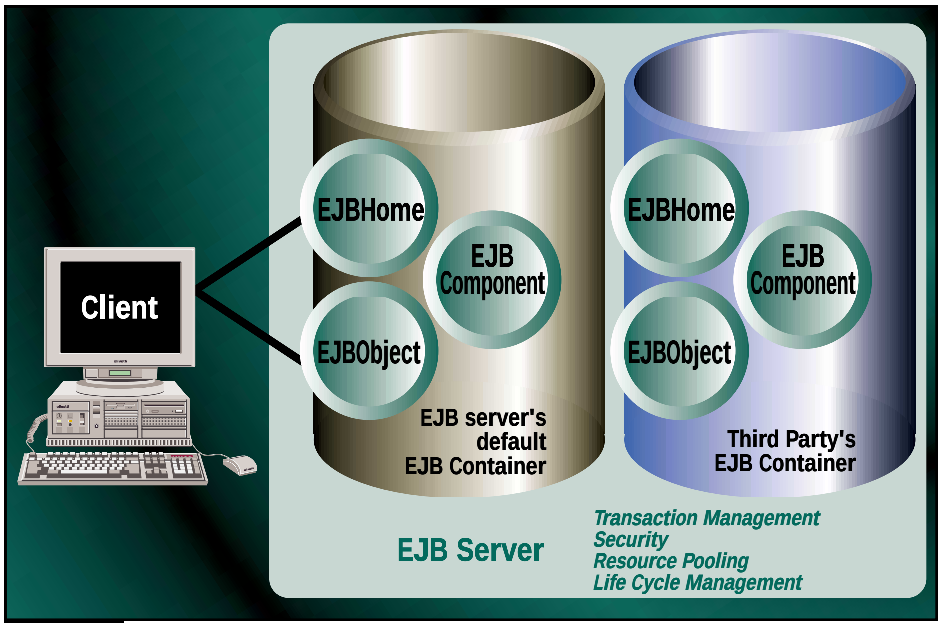


FIGURE 1: EJB component execution environment

Platform Portability

There's a real need to have operating system portability. The World Wide Web had a profound impact on the need for cross-platform (OS) software because you couldn't tell what platform an Internet user was surfing on. Java combated this problem by providing OS portability via the Java Virtual Machine, which translates operations into the correct OS platform's API. This common concept is called wrappering.

Before I started a career in software development, I thought a wrapper was simply packaging for my chewing gum; however, wrappers have revolutionized more than the candy industry. They're a prime enabler for the interoperability of heterogeneous components in the software industry. Interoperability through a wrapper or "adapter" design pattern is achieved by providing an acceptable interface that a client component expects and can communicate with effectively (see *Design Patterns: Elements of Reusable Object-Oriented Software* by Gamma et al.). Thus an existing component with an incompatible interface may be called by other software via a wrapper, achieving the reuse we've come to expect in component-based development. An example we should all be familiar with: a JVM wraps multiple operating systems in an acceptable common interface, obviating the need for developers to build multiple OS-specific versions of their software for each platform.

The key point introduced here, and one I'll refer to often: the wrapper design pattern is important to software portability (not just a convenient place to put old chewing gum).

SL Corp

www.sl.com

Resource Portability

A flexible application is built with an awareness of resource portability. Examples of resources might include transaction management services, naming services, communication protocols or a database management system. For instance, a commercial product built to access a storage resource such as a database should remain as neutral as possible with regard to its persistence mechanisms. Designing flexibility into this product might include providing a layer of insulation between your business objects and how they're stored. This will allow your customers to decide which database they wish to use with minimal impact to your product. With so many types of storage mechanisms, including RDMSs, object-relational databases, embedded databases and pure ODBMSs, you have to take database portability seriously.

Java provides some degree of insulation against platform-specific database code through its JDBC API. However, JDBC is flexible and powerful enough to allow vendor-specific code to be written through its database metadata features. You still have to use sound design patterns and wrappering techniques to ensure minimal impact when changing storage schemas. As we'll see, EJB specification 1.0 provides further guidance on resource portability.

Language/Component Model Portability

Language portability is the ability to support multiple development languages. This form of portability is provided to the development community through two widely regarded standards: Common Object Request Broker Architecture and Component Object Model. CORBA and COM allow components coded in multiple languages to communicate with each other through an interface definition language. IDL provides mappings between disparate languages such as C, C++, Java, Visual Basic (VB) and even COBOL. These unique languages are wrapped (once again, that valuable wrapper pattern) in recognizable, acceptable interfaces that foster reuse not only of business logic but also of human resources by way of current IT staff. For instance, language portability means you can use an existing COBOL or VB programmer to code a piece of functionality for your application that can be wrapped to look like a CORBA component. As you can see, portability allows quicker time-to-market for your products through the reuse of existing assets, even human assets.

Java provides platform portability, but you're tied to one language, Java. While CORBA and COM alleviate this single-language portability issue, they're still quite different from one another. CORBA objects may be deployed on any platform, while COM components are currently tied to the Windows operating system. Thus CORBA and COM have potentially different markets. While CORBA fits well in heterogeneous environments, COM is ideal for pure Microsoft shops. The EJB specification seriously addresses component model interoperability with CORBA; however, it leaves COM integration up to the EJB server vendor's imagination.

Career

Central

www.careercentral.com/java

Enterprise JavaBeans Portability

One of the primary goals of Enterprise JavaBeans is to provide a component model for building portable server-side components that addresses Java's lack of language portability and more. For the remainder of this article I'll focus on how the EJB specification lays the groundwork for Java's portable, server-side component model.

Portability Goals in the EJB Specification

"Enterprise JavaBeans will make it easy to write applications: application developers will not have to understand low-level transaction and state management details; multi-threading; resource pooling; and other complex low-level APIs...." —EJB specification 1.0, section 2.1: Overall Goals

Though not stated explicitly, the net result of this goal is the promise of interoperability between third-party EJB vendor products. This goal implicitly introduces the notion of a component execution environment, the central ingredient in providing EJB portability. In my first "EJB Home" column I described this concept briefly, but I'll review it again as a refresher.

A component execution environment typically consists of an EJB server and one or more EJB containers (see Figure 1). While JavaBean client components typically run within a visual container, EJBs execute within server-side containers. The container shelters your EJB component from its runtime platform by managing all of its interactions with the operating system. Together, an EJB server and its containers provide your components with access to distributed runtime services such as state management, distributed transaction management, multithreading and resource pooling, as stated above.

The Enterprise JavaBeans specification provides guidelines to encourage EJB server and container providers to build their products in a portable manner (see section 17 of specification). Figure 1 shows how an EJB server is making use of a third-party EJB container. Because the third-party container was built according to the EJB specification, it's portable to any EJB server and runs seamlessly within this component execution environment. EJB's component execution model is a prime example of how portability cuts time-to-market for products. An EJB server provider lacking skills in certain areas, such as object databases, could license a third-party EJB container that maps entity beans to an object database and *viola!* – it's in the EJB server business!

"Enterprise JavaBeans applications will follow the 'write-once, run anywhere' philosophy of the Java programming language. An enterprise bean can be developed once, and then deployed on multiple platforms without recompilation or source code modification." —EJB specification 1.0, section 2.1: Overall Goals

This EJB portability goal is easy! It simply reiterates Java's approach to providing OS portability from the standpoint of EJB. The very nature of EJB means it should run on any Java-compatible platform via the JVM. The benefit of "write-once, run anywhere" on the server is the ability to easily scale your application from a lowly workstation to a high-end enterprise server as volume demands. Compare that to the time it would take to rewrite a nonportable application from one server platform to another! By the time you finished, your dissatisfied cus-

IAM

www.iamx.com

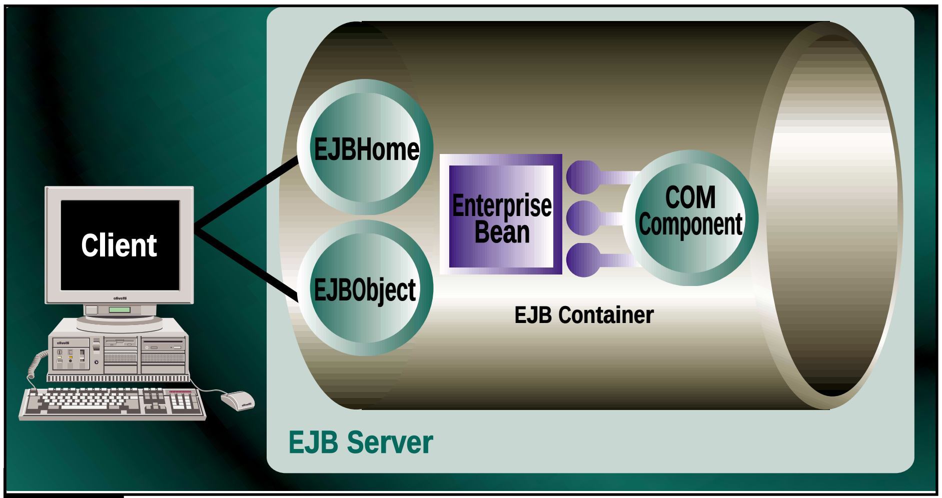


FIGURE 2: COM component wrapped as an EJB

tomers would have given their business to a company that could meet their business needs sooner.

"The Enterprise JavaBeans architecture will define the contracts that enable tools from multiple vendors to develop and deploy components that can interoperate at runtime." —EJB specification 1.0, section 2.1: Overall Goals

EJB specification 1.0 defines two types of contracts: between enterprise beans and their client and between enterprise beans and their containers. The enterprise bean-client contract ensures that the enterprise bean provider (e.g., enterprise bean developer) and container provider must collaborate to offer unique object identity, method invocation capabilities and an EJB home (e.g., factory interface and class). In a perfect world upholding this contract provides client-side portability with respect to various EJB server vendors.

A second component contract is defined between the enterprise bean and its container. By upholding this contract, an enterprise bean will be deployable within multiple vendors' tools, and will be able to use the runtime services of any vendor's EJB server. This lists a number of services, such as lifecycle management, that the container must offer an enterprise bean, as well as the interfaces that allow the container to manage the bean in a component execution environment.

"The Enterprise JavaBeans architecture will provide interoperability between enterprise Beans and non-Java programming language applications. The Enterprise JavaBeans architecture will be compatible with CORBA." —EJB specification 1.0, section 2.1: Overall Goals

Portability across multiple languages is important in developing software applications. The creators of the EJB specification, understanding the importance of portability, defined the Enterprise JavaBeans component model to allow for the interoperability with other component models such as COM/DCOM (Distributed COM) and CORBA. A separate specification describes the mapping of EJB to CORBA. It is available for download at www.javasoft.com/products/ejb/docs.html.

"In Part 2 I'll cover tricks, traps and techniques in designing and coding portable Enterprise JavaBeans"

The implementation of component model portability in EJB is vendor specific. An example implementation might have a COM or CORBA component "wrapped" in an EJB container, providing a client with EJB interfaces just like a normal enterprise bean (see Figure 2). You can see the business advantage of EJB's component model portability – existing and new business logic written in other component models will be reusable from your EJB application.

EJB Portability: Myth or Reality?

While the EJB specification 1.0/1 has laid the foundation for building portable enterprise beans, there is much left to be specified before true interoperability will be a reality. The EJB specification is vague in a number of areas important to EJB portability, including a container provider's responsibilities, multiple-vendor EJB server integration, CORBA and EJB security models, distributed/asynchronous event notification and mappings for COM integration. Without addressing these areas in future drafts of the EJB specification, EJB portability will be compromised. Am I worried? No. Fortunately, JavaSoft and a contingent of EJB proponents are working on these issues as I write.

Summary

Portability offers tremendous business advantages by enabling quicker time-to-market for new products that don't need to be rewritten for each platform in question. Portability also reduces maintenance costs of your applications and promotes reuse across disparate technologies, allowing you to use existing hardware, business logic and even software development talent.

Enterprise JavaBeans will soon be trusted to help build portable, server-side components and applications, providing the business advantage IT managers seek. The EJB specification addresses key issues around the integration of disparate component models and portability between multiple EJB vendors, providing early EJB adopters with a competitive advantage in the marketplace. Furthermore, EJB vendors are offering increasingly powerful tools to build EJB applications.

However, even though the recently released EJB specification 1.1 addressed issues around entity bean compatibility (i.e., entity beans are now mandatory), there is still much to be fulfilled concerning EJB's portability promise. It is lacking in critical areas, including security, system management and EJB server compatibility. While the specification defines the need for compatibility on numerous fronts, it doesn't qualify the means to this end. As vendors pump out new EJB products, how can you be sure of the portability promised by the specification?

In Part 2 I'll cover tricks, traps and techniques in designing and coding portable Enterprise JavaBeans. Expanding on what you've learned in this article, the code examples in Part 2 will give you an understanding of how to apply the wrapper design pattern to your development repetoire to ensure EJB portability. Until next time....

jwestra@uswestmail.net

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

9Net

www.9netave.net

SYS-CON RADIO INTERVIEWS WITH... GRANT WOOD & DANIEL BERG, AND MARTIN HARDEE

Broadcast live from San Francisco at JavaOne

JDJ: ***Joining me from Cyrus Intersoft is Grant Wood, an engineer, and Daniel Berg, the chief technology officer. What is your involvement in the Java industry?***

Berg: We're a software start-up based in Minneapolis, Minnesota; we have an Internet application platform that allows any Java-based software application and applet to be distributed anywhere. It's anytime, anywhere computing, utilizing the Internet in new ways.

We're delivering the capability to make your digital presence (e.g., your preferences, file systems, the applications you like to access most) meet up with your physical presence at any given point, any place on the Net. I could be at the airport using a kiosk, a PDA or whatever, and after making sure that whenever I go in and establish authentication, all my applications are available there, including all my file systems, security services and all the stuff that go along with providing OS-type services on the Internet.

JDJ: ***It seems like it's a very broad category of application servers. Would you fall under that category?***

Berg: I would say no because we don't offer the same things an application server would offer as far as services to the application. What we do is provide a transparent platform on which to deploy the applications. For example, if you were to take a regular Java-based program, say a word processor, most likely you developed access to a file system (in order to read or write a file); since it's probably an application, what we can do is run that application. You click on an icon and it comes into our environment; it's a run-time environment without installation. It almost behaves like an applet – you can go to a site or to our URL, and the application starts execution.

Now when I go in there and do a file dialogue or file open, I not only see local file systems but also any remote file systems I'd have access to. We can dynamically offer some of these services to applications.

We can take applications that have been developed with multi-tiered environments that have app servers and all the different pieces of the equation, and they will still work on top of our platform as long as whatever you are deploying is Java.

JDJ: ***Who are some of the people who are using your technology and in what ways are they using it?***

Berg: We're initially going after ISPs and service providers and giving them new ways of deploying services. We're also going after Java developers and letting them know that here is a new channel for them to deploy their application. And it's really a "no brainer." Grant could write an application and run it on a platform allowing a user to access that software from anywhere, and he doesn't have to change anything in his application. There are no APIs. It's all transparent through the VM. It offers developers new opportunities and channels.

Wood: A developer could download Speiros and log into a server somewhere that would not only give him access to an open source project that he may be working on or a project for work, but the tools he uses to develop that could be handed to him from the server as well. And those tools could be updated and changed on a daily basis.

What we have is a vehicle for doing real time deployment of software as well. As a developer, I could log into a server. I would have my files and any shared files I was working on. They could put it all on a server, and now when people login, maybe that's how they check out their software. They just click on the program, write it, compile it and they're done. You don't have to learn the way they check it out. They could be using CVS, JC...whatever they want. It's really an excellent platform for distributing, computing and development. It takes a lot of the thought out of the back end of it and allows you to hand out tools or the tools of choice to whoever is using the links.

JDJ: ***Two things I've always heard from Java developers that are the most important are the ease of use and speed.***

Wood: Well, can you click an icon?

JDJ: ***That's all I really need to know. Give us a little idea of that. Tell us how it works.***

Wood: You know how the Web is right now; every site is different. How do you find your way through a site? You have to claw through it. Since we're not doing content anymore only applications, we're doing things much more familiar to users. When someone turns on their cable box and it logs into a Speiros server somewhere, it gives them a little menu with icons for their applications. You didn't have to browse anywhere. You could search and say I want to see word processors, and it might bring up 50 word processors; you can click on them, get information and bring up the Web page of the guy who developed it. But all you do to launch it is double-click on it. And that could be coming from anywhere on the Web.

All you need to know is point and click – it's that simple. Now you know how to use that kiosk in the mall, that set-top box, or your MAC, it really doesn't matter. We're really delivering on the promise of Java, "write once, run anywhere." You could do that before, but now you can actually get to it anywhere. You could sit down anyplace because now your application is a full-blown application. It's out on the Net. Your files are something we deliver as well. As you login, here's your home directory and maybe you're connected to 150 different servers out there, and because your files and your applications are on the Internet, it doesn't matter where you are. You don't have to install it on your box or keep your files with you when you leave your home machine. Why would you even keep anything in a local box when you can just keep them all out on the Internet?

Berg: We have a developer's release available on our Web site, CyrusIntersoft.com, and we're looking for developers to come download it and try it out.

Worldwide Internet

www.wipc.net

***JDJ:* Joining me right now is Martin Hardee, manager of the java.sun.com Web site. Since we're at JavaOne, tell us a little bit about the JavaOne Web site. What do you think is its coolest part?**

Hardee: The JavaOne Web site is actually a subsite of java.sun.com – java.sun.com/JavaOne. I think probably the coolest thing our site does is we aspire to get the community together. You see it on the Java developer connection, and I think we've seen it at JavaOne where we have voting and things for the applications, but it is really the attendees that are building things and putting them together.

***JDJ:* Let's get to the Java Sun site. It looks a little different.**

Hardee: We redesigned the site slightly. We thought about all sorts of radical redesigns as you always want to do. We do surveys continuously and look at all the Web feedback. We probably get thousands of comments a year. We did some remote usability tests on the phone where we'd call people in Australia or some other country and ask them to look at some mock-ups we had. We learned people really like the look of the current site. We pretty much stuck with something that looks the same, but the technology has grown so much.

Last year we did almost 250 different discrete full technology releases on the site, which is basically one every working day. There's an explosion of technology and with the Java community process there's going to be even more. We restructured things so you could find what you needed, from discussion topics to industry solutions and products. Based on the responses we've gotten so far, I think we hit the nail on the head. We'll continue to evolve it of course.

***JDJ:* Tell us about some of the feedback you've gotten and if it influenced your redesign?**

Hardee: We've actually done a lot of these improvements incrementally. About six months ago we did a survey on the site; users said the technology has grown tremendously so they couldn't find anything anymore. The first thing we did was reorganize the products in the API's page. It was organized both by APIs and alphabetically. We also built an A-to-Z index of the important items on the site, which has been very popular. In the first week, it was one of our top 20 URLs on the site. We get about a million to two million page views a day.

The hits range is up to 6–7 million hits a day. We've also tried to marry java.sun.com and the Java developer connection subsite together for a more graceful transition. And the JDC – a few months ago you had to login for everything. We've taken a lot of that off so there's more free and easy access. We've left on login for the discussions and other things where you have to be part of the community and have to be known. But primarily we've made access to the JDC easier to find and easier to access once you get there.

***JDJ:* What can we expect in the future from the Web site?**

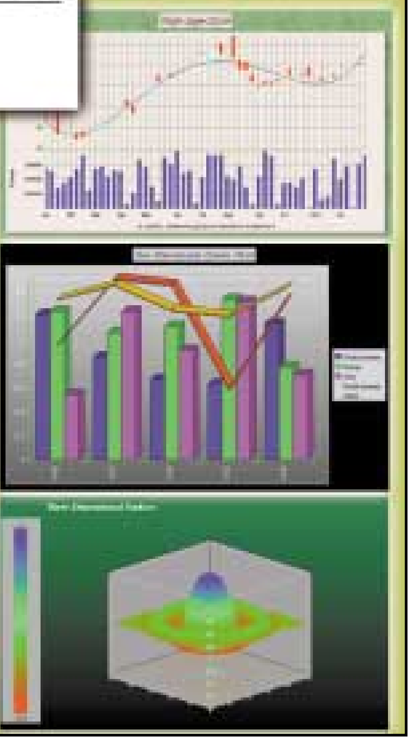
Hardee: We did community source this year and have really ramped up the Java community process, which we're trying to make more of a Web-based activity. One thing I'm interested in, our site's been around for a while and while we use a lot of Java technology, including Java Web Server, parts of the site are still running Apache. We're really interested in using Java Server Pages. We're doing a lot of servlet work right now. Our commerce system and the JDC are all servlet based, so under the hood where you don't really see anything, we're definitely having a lot of fun playing with servlets and JSPs. If you have any comments, go to the feedback links, and all your comments go into a database. We read them, route them and everything else.

***JDJ:* You actually read them?**

Hardee: Somebody has the job of reading them and figuring out where they go, which is a tough job. We track all the comments in the database and make sure they get answered.

VISUALIZE INC
GET THE PICTURE

Founded in 1996, Visualize, Inc. is the premier data visualization provider with over 20 years total of Java Development expertise.



Our products offer:
- 2D & 3D Graphics
- User Interaction
- Analytical Capabilities
- Rapid Deployment

VantagePoint:
A class library for embedding graphical functionality in a Java application

DataVista Pro:
An applet designed to graphically display data within a web page

For a free evaluation copy of our software visit our web site
www.visualizeinc.com
or contact
JDJ@visualizeinc.com

100% PURE JAVA

1819 E. Morten, Suite 210
Phoenix, AZ 85020
602-861-0999

Geek Cruises

www.geekcruises.com

KL Group

www.klgroup.com/pagelayout

What's Coming in CORBA 3?

Three major categories of specifications enhance CORBA integration with Java and the Internet

WRITTEN BY
JON SIEGEL

The next release of the CORBA specification will be a major one, CORBA 3.0. The last time the major release number was incremented – to CORBA 2.0 – it signified the standardization of interoperability. What's new and different enough in this version for OMG to increment the major release number this time?

Despite its compact designation, CORBA 3 isn't a single specification – instead, it's the collection of specifications, adopted individually, that will be added to the current CORBA 2.3 to form the CORBA 3.0 release. Although we use "CORBA 3" as shorthand to refer only to the new parts, the official designation CORBA 3.0 refers, formally, to the entire CORBA specification book. With the posting of final submissions for the CORBA Component Model (CCM), Scripting and the Persistent State Service (PSS) on the OMG Web site in early August, all parts of CORBA 3 are finally available. Votes on these last few parts will be underway when you read this, and are expected to be completed by the end of 1999; check OMG's Web site for the latest word.

There are about 10 new specifications in CORBA 3, and I'll touch on all of them here. Java programmers will be particularly interested in the reverse Java-to-IDL mapping and the CCM's embracing of Enterprise JavaBeans (EJBs), so I'll put some extra detail into these sections.

I'll give you the URL for every specification we discuss here. Documents under consideration for all of OMG's works in progress are available to members and nonmembers alike at www.omg.org/schedule. Find the process or future specification you're interested in and click on it. This will bring up a new page with URLs for all related documents. For specifications, go to www.omg.org and look down the left-hand side of the page for "The OMA"; all specifications are available under that heading. You can download any of these OMG documents, specifications, or works in progress without charge.

Introduction to CORBA 3

The specifications included in the designation CORBA 3 divide neatly into three major categories that we'll cover in the following order:

- Java and Internet integration
- Quality of service control
- CORBA component architecture

Java and Internet Integration

The three specifications discussed below enhance CORBA integration with Java and the Internet.

Java-to-IDL Mapping

CORBA 3 adds a Java-to-IDL mapping to the "normal" IDL-to-Java mapping you're familiar with if you've done any Java/CORBA programming. To use it, you start out by writing RMI objects in what the specification refers to as the "RMI/IDL subset of Java." It's a pretty full subset; restrictions affect things such as multipath inheritance of overloaded methods, name case collisions, and private types in interfaces. (Java allows private types in distributed interfaces. CORBA can't seem to figure out why anyone would want to keep a distributed variable private.) Objects must extend java.rmi.Remote, and exceptions must inherit from java.lang.Exception. Obviously, CORBA objects passable by value play a large role in the CORBA-side implementation of this specification.

Once you've written the objects, two things happen: first, by compiling through rmic with the proper options set, your objects generate CORBA stubs and use IIOP instead of RMI protocol. Second, the RMI compiler will output the IDL for your object into a file that you can then compile in any programming language, on any ORB, allowing you (or your friends) to write CORBA clients in any language, on any IIOP-speaking ORB, that can invoke your Java object.

This does a number of things. It turns Java object programmers into CORBA object programmers, and lets Java objects play in CORBA's multilanguage environment. (Unfortunately, it doesn't do anything for Java client programmers!) According to the first draft of EJB 1.1, it's a future requirement for Enterprise JavaBeans interoperability and will ensure that EJBs can play in this multilanguage environment as well.

It's not a round-trip mapping, quite intentionally. Since out and inout parameters don't occur in Java, the reverse mapping has no element that corresponds to the Holder classes in the IDL-to-Java mapping, and some IDL types will never occur in the IDL output from rmic. In some ways, though, the mapping is pretty clever: variables that appear only in Java set-and-get operations will be mapped to an IDL Attribute.

This specification is available from the Web at www.omg.org/corba/clchpter.html#jilm.

Firewall Specification

The CORBA 3 firewall specification defines transport-level firewalls, application-level firewalls and (perhaps most interesting) a bidirectional GIOP connection useful for callbacks and event notifications.

Palm Computing

www.palm.com

Transport-level firewalls work at the TCP level. By defining (courtesy of IANA) well-known ports 683 for IIOP and 684 for IIOP over SSL, the specification allows administrators to configure firewalls to cope with CORBA traffic over the IIOP protocol. There is also a specification for CORBA over SOCKS.

In CORBA, objects frequently need to call back or notify the client that invoked them; for this the objects act as clients and the client-side module instantiates an object that's called back in a reverse-direction invocation. Because standard CORBA connections carry invocations only one way, a callback typically requires a second TCP connection for this traffic heading in the other direction – a no-no to virtually every firewall in existence. Under the new specification, an IIOP connection is allowed to carry invocations in the reverse direction under certain restrictive conditions that don't compromise the security at either end of the connection. The firewall specification comprises two documents: www.omg.org/cgi-bin/doc?orbos/98-05-04 and an erratum, www.omg.org/cgi-bin/doc?orbos/98-07-04 .

Interoperable Naming Service

The CORBA object reference is a cornerstone of the architecture. Because the computer-readable IOR was (until this service) the only way to reach an instance and invoke it, there was no way to reach a remote instance – even if you knew its location and that it was up and running – unless you could get access to its object reference. The easiest way to do that was to get a reference to its naming service, but what if you didn't have a reference even for that?

The interoperable naming service defines one URL-format object reference, iiopoc, that can be typed into a program to reach defined services at a remote location, including the naming service. A second URL format, iiopname, actually invokes the remote naming service using the name that the user appends to the URL, and retrieves the IOR of the named object.

For example, an iioploc identifier, iioploc://www.omg.org/NameService, would resolve to the CORBA naming service running on the machine whose IP address corresponded to the domain name www.omg.org (if we had a name server running here at OMG). The URL for the interoperable naming service specification is www.omg.org/cgi-bin/doc?orbos/98-10-11.

Quality of Service Control

Asynchronous Messaging and Quality of Service Control

The new messaging specification defines a number of asynchronous and time-independent invocation modes for CORBA, and allows both static and dynamic invocations to use every mode. Results of asynchronous invocations may be retrieved by polling or callback – the choice is made by the form used by the client in the original invocation.

Policies allow control of quality of service of invocations. Clients and objects may control ordering (by time, priority or deadline); set priority, deadlines and time-to-live; set start and end times for time-sensitive invocations; and control routing policy and network routing hop count. The specification also defines CORBA routers and store-and-forward agents for IIOP invocations. Routers with the highest defined quality of service will pass invocations with a transaction-like handshake and store invocations persistently, providing messaging-like network transmission. The routing specification defines IDL interfaces to the marshaling engine, an interesting piece of work in itself.

The URL for the messaging specification is www.omg.org/cgi-bin/doc?orbos/98-05-05.

Minimum, Fault-Tolerant and Real-Time CORBA

Minimum CORBA is intended primarily for embedded systems. Embedded systems, once they are finalized and burned into chips for production, are fixed, and their interactions with the outside network are predictable – they have no need for the dynamic aspects of CORBA, such as the DII or the IR that supports it, which is why these features are not included in minimum CORBA. The URL for the minimum CORBA specification is www.omg.org/cgi-bin/doc?orbos/98-08-04.

Real-time CORBA standardizes resource control – threads, protocols, connections and so on – using priority models to achieve predictable behavior for both hard and statistical realtime environments. Dynamic scheduling, not a part of the current specification, is being added via a separate RFP. The URL for the real-time CORBA specification is www.omg.org/cgi bin/doc?orbos/99-02-12; an erratum is www.omg.org/cgi-bin/doc?orbos/99-03-29.

Fault tolerance for CORBA is being addressed by an RFP, also in process, for a standard based on entity redundancy and fault management control. The URL for all information on this RFP is www.omg.org/techprocess/meetings/schedule/Fault Tolerance RFP.html.

CORBA Components Package

CORBA Objects Passable by Value

Termed valuetypes, objects passable by value add a new dimension to the CORBA architecture that previously supported passing (and invocation) only by reference. Like conventional CORBA objects, these entities have state and methods; unlike CORBA objects, they don't (typically) have object references and are invoked in-process as programming language objects. It's only when they're included in parameter lists of CORBA invocations that they show their talent by packaging up their state in the sending context, sending it over the wire to the receiving context, creating a running instance of the object there and populating it with the transmitted state. Frequently used to represent nodes in binary trees or cyclically linked lists, valuetypes have been specified and implemented to faithfully represent these important constructs. This specification gives CORBA the capability of the Java serializable, and is used extensively in both the reverse mapping and component model. The valuetype specification may be downloaded from URLs www.omg.org/cgi-bin/doc?formal/99-07-09 and www.omg.org/cgi-bin/doc?formal/99-07-10. Many aspects of valuetypes show only in the language mappings; for these go to www.omg.org/library/clangindx.html.

CORBA Components and CORBA Scripting

The CCM takes the key services you use most regularly – persistence, transactions, security and notification – combines them with the POA's servant-handling capability, and wraps all these tools in higher-level interfaces corresponding to the patterns that experienced programmers use to code enterprise and Internet server applications.

This means that:

- CCM applications are very compact. They use little code, and the little that's required is devoted almost totally to business programming.
- Infrastructure functions – storing data, activating and deactivating servants – are done automatically and coded automatically as well.
- CCM implementations will be built around an industrial-strength infrastructure written by specialists and tuned for optimum performance in stressed environments, including the enterprise and Internet. When this infrastructure runs your CCM application in its tuned environment, you automatically get the benefits – high throughput, great performance,

The Theory Center

www.theorycenter.com

robustness – even though you don't have to write any infrastructure code, or even code to POA and CORBA services interfaces.

The four major parts of the CCM are:

- A model that presents common functionality – transactions, security, event handling, persistence – to the programmer at a high level using declarative languages and, optionally, visual tools
- A container environment that packages (at the basic conformance level) transactions and security, adding (at the extended conformance level) persistence and event handling
- A software distribution format that enables a CORBA component software marketplace
- Integration with Enterprise JavaBeans

Component model functions are packaged and presented to programmers at a higher level of abstraction than are the bare CORBA services. Component interfaces are declared using newly standardized additions to OMG IDL. Components declare their persistent state using Persistent State Definition Language (PSDL, a superset of OMG IDL), defined in the new PSS (a new CORBA service not presented here). Programmers then use Component Implementation Definition Language (CIDL, an extension of PSDL) to declare some of the component's behavior; CCM products use these declarations to generate code for parts of the implementation. Finally, an XML-based configuration language ties components together in assemblies and configures them for individual installations. Because these new languages work at higher levels of abstraction than CORBA and the CORBA services, business programmers can write enterprise- and Internet-level applications with less help (perhaps no help!) from infrastructure experts. In addition, some of the languages were designed to be generated by visual tools.

One objective of CCM is to allow you to write distributed applications that mix CORBA components running in CORBA component servers with EJBs running in EJB-technology-based servers. This allows programmers to create an application by reusing existing components of either kind. To accommodate EJBs the CCM defines two levels of containers and conformance: the basic level container is transactional and secure, and provides simple persistence; its definition corresponds, almost feature for feature, to EJB 1.1. To this the extended-level container adds container-managed (and -implemented) persistence for multiple segments, event handling, multiple interfaces and navigation. The specification covers all of the various ways that CORBA clients can interact with EJBs and how Java clients can interact with CORBA components. We'll go over details of these CCM/EJB interactions in a future column. Since every container presents its services to components through the same set of standardized interfaces, component applications are portable from any vendor's container to another's. A basic-level container packages up subsets of the CORBA Transaction Service, CORBA security, and simple persistence in a new set of interfaces. At level 2 the container adds more of the PSS and a subset of notification service. It's not necessary for a container implementation to furnish its own persistence service; the

"CORBA does as much as it can to support every programming language and platform"

specification assumes that CCM products will use any standard PSS and that your site administrator will buy and install one. Of course, vendors are free to package PSS and CCM together if they want. Level 1 containers must include Transaction and Security support, however, and level 2 containers have to add their own event handling capability.

Extended CORBA components support multiple interfaces. The CCM defines interfaces, provided by the container, that allow a component-aware client application to navigate among them. To component-unaware clients – including all clients written in a CORBA 2 environment – each interface appears to be an individual CORBA object; these clients can't take advantage of the implementation's component nature but can invoke it as a normal CORBA object without any trouble.

To help build a market in components, the CCM defines a software distribution format with several notable features. The distribution format contains executables for all of the platforms a vendor wants to support, in a type of zip file. Installer code, built into the container, extracts and installs the proper executable for the platform it's going to run on. Following installation, component executables can be configured. This overcomes another resistance point – that it's rare to find two installations that need exactly the same piece of functionality. By building and selling flexible modules to wider markets, vendors can attain the volumes they need and the component market will build to critical mass.

Configuration files contain information on events emitted and consumed by components, at least at the extended level. Containers construct channels for the events and transmit them at runtime. Since CCM applications typically consist of several component types, the container will also have to connect up an invocation by one type to an interface on another; this also is specified in the configuration file and install-time configuration process. By the way, configuration files use XML format.

There's also a scripting language specification that will map various scripting languages to the CCM. This will add another way to assemble components into applications, in addition to the XML-based assembly tools described in the CCM. We'll devote a column to this sometime in the future too.

The CCM and scripting specifications hadn't completed their final votes when this column was written, but may be a done deal by the time you read it. Check out www.omg.org/techprocess/meetings/schedule/CORBA_Component_Model_RFP.html and www.omg.org/techprocess/meetings/schedule/CORBA_Scripting_Language_RFP.html for details, and to download your own copy of the 700-page CCM document.

Summing Up...

CORBA does as much as it can to support every programming language and platform. Because Java's object model aligns closely with CORBA's, and Java's Virtual Machine is platform-portable, CORBA is able to do more with it than with other languages, and this shows in the specifications we've reviewed in this month's column. We think this bodes well for productivity and popularity on both the CORBA and Java sides of the aisle, and I may muse more about it in a future column.

siegel@omg.org

Author Bio

Jon Siegel, Object Management Group's director of technology transfer, presents tutorials, seminars and company briefings around the world. The author of CORBA 3 Fundamentals and Programming, *he has extensive experience in distributed computing, object-oriented software development and geophysical computing. Jon holds a Ph.D. in theoretical physical chemistry from Boston University.*

Protoview

www.protoview.com

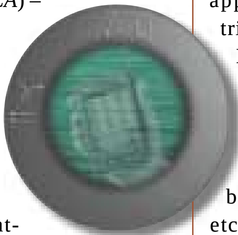
Mindbridge.com Introduces IntraSmart

(*Fort Washington, PA*) – Mindbridge.com has released IntraSmart, a comprehensive, ready-to-go intranet that includes powerful business applications and the software components necessary to deploy an enterprise-level intranet. IntraSmart's business applications include Document Sharing/Management, Group Calendar/Scheduler, Group Address Book/Company Rolodex and Employee Directory. www.intrasmart.com

HOB Offers HOBLink J-Term 2.2

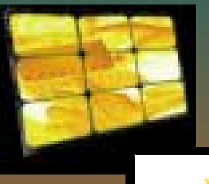
(*Dallas, TX*) – Web-to-host connectivity is available using HOBLink J-Term version 2.2, a Java-based terminal emulation software that offers 3270, 5250 and VT525 connectivity among mainframe, midrange, UNIX and personal computers. HOBLink J-Term provides secure, multiplatform, remote data access to mainframe, AS/400, UNIX and DEC computers via the Internet, intranets or extranets. www.hob.de

PalmSource '99 Conference



(*Santa Clara, CA*) – Palm Computing, Inc., will host its third annual PalmSource conference for the Palm Computing platform on October 19–22, 1999, at the Santa Clara Convention Center. The conference will provide attendees with platform technology presentations, industry perspectives and product showcases. Sponsors include Aether, AvantGo, BellSouth, Handspring, IBM, JP Systems, Motorola, QUALCOMM, Sun Microsystems and Symbol Technologies. Participants can register online, via fax or by mail. www.palmsource.com

Introducing the First SQL DB Written in Java that Runs on Handheld



(*San Mateo, CA*) – PointBase, Inc., and Psion Computers announced success running PointBase Mobile Edition on a Psion Series 5mx handheld computer – the first time a SQL database written in 100% Pure Java has run on a handheld device of any type. The Psion-ready PointBase Mobile Edition is available now with full developer support. www.pointbase.com and www.psionusa.com

ObjectSwitch Offers EJB Interface

(*San Francisco, CA*) – ObjectSwitch Corporation announced its new EJB interface for ObjectSwitch 3. The ObjectSwitch J-Adapter allows CLECs, Wireless Carriers and alternative network providers to create intelligent mediation and provisioning applications using EJB servers from multiple vendors, and to combine the resulting applications into a single seamless service offering. www.objectswitch.com

JavaCC Now Supported by Metamata

(*Fremont, CA*) – Java Compiler Compiler (JavaCC), a parser generator for use with Java applications, is now being distributed and supported by Metamata. In addition to the parser generator, JavaCC provides other standard capabilities related to parser generation such as tree building, actions, debugging, etc. www.metamata.com

ServletExec 2.1 and ServletExec Debugger 2.1

(*Alpharetta, GA*) – New Atlanta Communications, LLC, announces Servlet Exec 2.1, a free upgrade to its servlet and JSP engine, and the ServletExec debugger 2.1, a free tool for developing and debugging servlets within popular Java IDEs. www.newatlanta.com

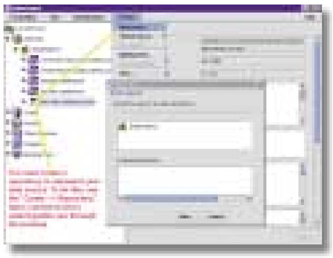
Jeode Platform Shines in Performance Testing

(*Fremont, CA*) – Insignia Solutions announced that its Jeode Embedded Virtual Machine (EVM) tested faster than other virtual machines evaluated in performance benchmarks conducted by Pepsan Inc., a third-party organization. www.insignia.com

JDJ Editor Gives Keynote Speech at Hi-Tech Job Fair

(*Meadowlands, NJ*)– Sean Rhody, editor-in-chief of ***Java Developer's Journal***, was the keynote speaker at the career event at Giants Stadium on September 15, 1999. ***JDJ*** was the exclusive media sponsor for this conference.

Earlier this year ***JDJ*** announced their Java career opportunities services. http://www.sys-con.com/java/jobs/javajobs.cfm



Cerebellum Launches 1.3

(*Pittsburgh, PA*) – Cerebellum Software, Inc., released its next upgrade, which enables a wider variety of applications to easily access, integrate and update data located in an increased number of data source types. The new version also provides greater flexibility in the types of applications that can use Cerebellum's application programming interface. www.cerebellumsoft.com

Java Developer's Journal Launches *JDJ* Consulting Services Division

(*Pearl River, NY*) – In September 1999, ***Java Developer's Journal*** announced its new consulting services division, which will provide high-end, enterprise-wide Java technology solutions to mid-sized and Fortune 500 corporations. As the world's leading Java resource, ***JDJ*** has been providing cutting-edge Java solutions for over four years and is poised to use its pool of expert Java developers to undertake mission-critical Java projects. www. JavaDevelopersJournal.com

JDJ Consulting Services
Your Key to Java Experts

SilverStream Announces J2EE Seminars

(*Burlington, MA*) – SilverStream Software, Inc., has launched a worldwide educational seminar series designed to educate Java application developers and architects on J2EE. Beginning in October, the free half-day "J2EE for the Real World" sessions will include software, educational materials, technical demonstrations and instruction from industry experts. For seminar information and registration go to www.silverstream.com/website/SilverStream/Pages/events_f.html.

InetSoft

www.inetsoftcorp.com

PRODUCT REVIEW

SQL 2000 v7.5

by Pervasive Software

REVIEWED BY JIM MILBERY

AUTHOR BIO

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

jmilbery@kuromaku.com

SQL 2000 v7.5: Pervasive Software
Web: www.pervasive.com
Phone: 800.287.4383

Test Environment:
Client/Server: Gateway Solo 366, 256MB RAM, 10 Gigabyte disk drive, Windows NT 4.0 (Service Pack 4)

Pricing: $149.00

Requirements:
Intel 486, Windows NT 4, 95 or 98, 29 MB hard Disk 6 MB RAM

JAVA DEVELOPER'S JOURNAL JDJ ★★★★★ WORLD CLASS AWARD

Pervasive Software has released version 7.5 of its ubiquitous database engine and software development kit. I got the chance to take a look at the database and the various tools using an evaluation copy of the software for Windows NT 4.0.

Installation and Configuration

I installed the software from the PervasiveSQL 2000 SDK CD-ROM. Pervasive automatically searches for the presence of a working copy of their workgroup database engine when you install the SDK kit. If you don't have a running copy of PervasiveSQL, the SDK will install the database server kit first. The install routine for the database server gives you the choice of accepting a typical installation or selecting a custom install option. Selecting the custom install for the database server appears to make no difference; the installation program seemed to go ahead and install all the options anyway. All in all the installation went smoothly. After the installation completes, the server runs a small test to ensure that everything was installed properly.

Once the database is running, the SDK kit can be installed. Selecting the custom option for the SDK gives you a number of choices. PervasiveSQL 2000 provides a comprehensive set of tools for interfacing with the database, such as plug-ins for Inprise, Microsoft Visual Studio and Java. The database server is available for Windows NT and Novell NetWare, and Pervasive has announced versions for Solaris and Linux. Pervasive even makes a version of the engine available for various smart-card environments.

Architecture

The database server uses a small footprint and starts up easily and quickly as an NT service. The underlying technique that serves as the foundation for PervasiveSQL 2000 is an engine that Pervasive calls the MicroKernel Database Engine (MKDE). The MKDE provides a core layer for all low-level processing, such as index management, caching, transaction control and lock management. The PervasiveSQL engine is built on top of the MKDE with a transaction layer based on Pervasive's Btrieve architecture, and a relational layer based completely on ODBC. Pervasive's SQL interface is abstracted from the MicroKernel by a layer that's called the SQL Relational Database Engine (SRDE) (see Figure 1).

Part of Pervasive's differentiation from the other database players in the marketplace is their support of relational and transactional interfaces based on their Btrieve layer. Pervasive's relational interface fully supports the ODBC standard (and by extension JDBC), and programming against the MKDE using ODBC is easy and quick. Pervasive has essentially built its relational database-access layer around the ODBC standard, making it a straightforward process for embedded developers to work with the relational layer. Although you're free to develop all your applications using the relational layer, the folks at Pervasive make no bones about the fact that one of its advantages is the ability to use the transactional interfaces to access the database at a much lower level. On their Web site, Pervasive provides a detailed description of their product in a paper called "Pervasive Products and Services," where they describe what they believe to be some of the major benefits of the transactional interface, e.g., speed, data integrity and scalability. Although the original Btrieve API is over 15 years old, it continues to evolve. Thus it supports modern RDBMS concepts such as transaction support and on-line backup, which are more commonly identified with relational database engines. A single "logical" database can be accessed by the lower-level transactional APIs as well as by ODBC and SQL. The Pervasive Control Center provides you with tools to add the necessary data dictionary information to a set of Btrieve files that form a logical database. If you wish to access the database using the transaction interface, they provide a suite of ActiveX controls for this purpose, as well as interfaces for Visual Basic, Delphi and C++. PervasiveSQL also provides an interface from Java that's based on JDBC, but they've added some extensions to account for the fact that JDBC is based upon SQL, which isn't a requirement for the Btrieve API layer. This allows a Java program to access the Btrieve API directly, without relying on column information stored in the relational layer. Current Btrieve programmers will find that the Java interface hides many of the implementation details when compared with the older non-Java API.

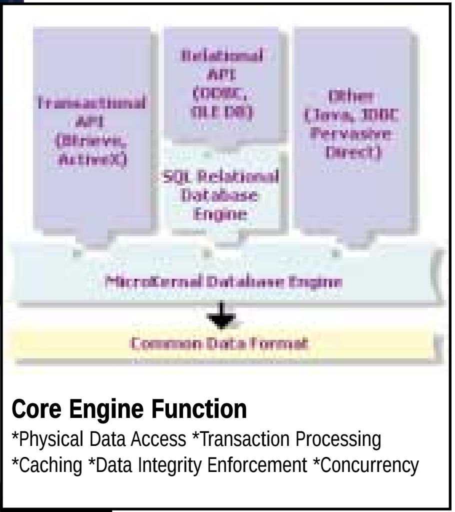

Transactional API (Btrieve, ActiveX)
Relational API (ODBC, OLE DB)
SQL Relational Database Engine
Other (Java, JDBC Pervasive Direct)
MicroKernel Database Engine
Common Data Format
Core Engine Function
*Physical Data Access *Transaction Processing
*Caching *Data Integrity Enforcement *Concurrency

FIGURE 1 Pervasive MicroKernel Architecture

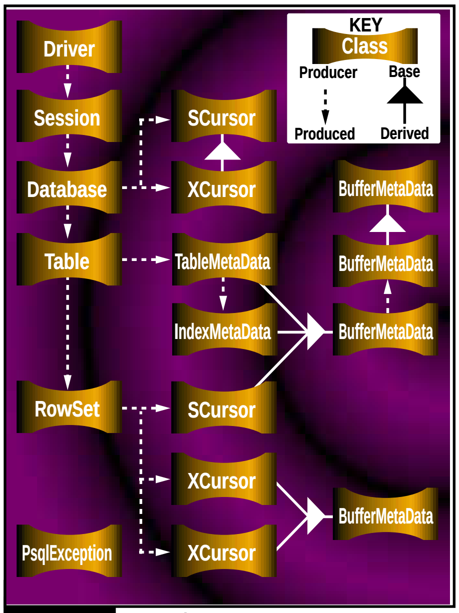


FIGURE 2 Pervasive Java API

Java Buyers Guide

www.javabuyersguide.com

Pervasive SQL and Java

I found the Java interface reasonably easy to use. Pervasive provides documentation for working with their sample applications in either Inprise's JBuilder or Symantec's Visual Cafe. However, I was able to work with the sample projects using my copy of Oracle JDeveloper. Pervasive makes use of the factory concept for their Java API, as shown in Figure 2.

There are class objects for the major tasks you'll need to undertake in order to work with your PervasiveSQL database and Java. What makes Pervasive different from working directly with JDBC is you can also access the data using the transactional API. If you have a logical database that includes a relational data dictionary layer, you can use the DATABASE class to work with the metadata, as in a classical relational design. However, you're not required to use this interface in order to access the Btrieve layer. The Java API will allow you to work directly with data files in a loosely coupled database. The standard installation of Pervasive includes a number of sample databases and programs for you to work with. As usual, I bypassed the samples and went directly to creating my own database using the Pervasive Control Center, as shown in Figure 3.

The control center provides a one-stop-shop for accessing all the major tools and interfaces for creating and managing Pervasive databases. There are lots of hidden gems within the various tools that are shown in the outline control on the left-hand panel of the control center. I was easily able to create a new database with a relational layer using the ODBC interface from the control center. I modified a few existing table-and-load scripts and used them with the SQL Query window to create a new table and populate it with data. The control center provides a classic text display as well as a grid display for records, as shown in the panel on the right-hand side of Figure 3. Pervasive provides plenty of tools and utilities for working with the server engine as well as the databases managed by the server. Pervasive's API supposedly allows you to control and tune your databases as well as access records, and I suspect that the development team made use of this API when constructing the various utilities that are called by the control center. There are an abundance of tools, but the user interface between them is somewhat inconsistent. Many of the utilities provide for making detailed modifications at the lowest levels of the database, and the utility interface expects you to know what you're doing. For example, the Maintenance utility gave me low-level access to the UGRADS file that I created inside my new database. However, the utility assumed that I knew what I wanted to do to the file, and there were no wizards to guide me through the modification process.

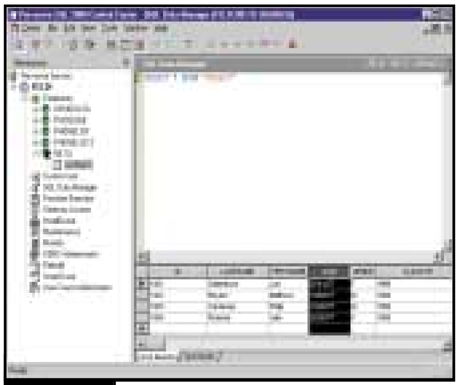


FIGURE 3 Pervasive Control Center

Once my database was created, it was a relatively simple process to use the Java API to access my UGRADS table through a Java program (although I'll admit that I stuck to using the relational layer with the DATABASE class).

Summary

The Pervasive Software Web site offers an Aberdeen research paper that compares the total cost of ownership of PervasiveSQL to other database engines on the marketplace. Given the fact that you can control the entire database and server through the API interface, I believe you can build an application and database that's easy to maintain and manage. If you plan on embedding a database inside an application and are willing to control the database in this manner, Pervasive offers a compelling product. Pervasive will appeal to the developer who wants to exert a fine degree of control over the data while still allowing end users to access the underlying data using friendly SQL interfaces over ODBC.

Cyscape

www.cyscape.com

Get Your Own Subscription to the Finest Technical Journals in the Industry!

1-800-513-7111
www.sys-con.com

SYS-CON PUBLICATIONS

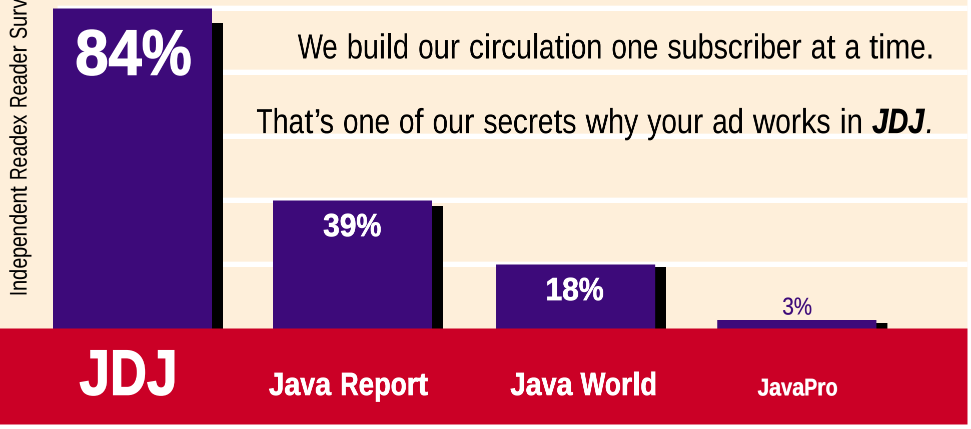
Only *Java Developer's Journal* Readers are 100% Pure Java

PUBLICATIONS REGULARLY READ BY JAVA PROFESSIONALS

Your ad in ***Java Developer's Journal*** reaches only 100% Java professionals who make the decisions to purchase Java related products and services, not Visual Basic programmers who never requested the publication you advertise in!

We build our circulation one subscriber at a time.

That's one of our secrets why your ad works in ***JDJ***.



Before you advertise in a publication, please ask how many real Java readers you're actually reaching!

Carmen Gonzalez
Vice President, JDJ Advertising Sales

JAVA DEVELOPER'S JOURNAL
The World's Leading Java Resource

www.JavaDevelopersJournal.com or call *914-735-0300*

Sigs Java C

www.javad

Conference

evcon.com

Where in the world can you find the most qualified Java experts for your enterprise Java technology solutions?

JDJ Consulting Services
Your Key to Java Experts

www.javadevelopersjournal.com
jdjconsulting@sys-con.com

ADVERTISING INDEX

ADVERTISER	URL	PH	PG
9NETAVENUE, INC.	WWW.9NETAVE.NET	888.9NETAVE	63
AMERICAN CYBERNETICS	WWW.SOFTEXPORT.COM	800.899.0100	55
AVANTSOFT, INC.	WWW.AVANTSOFT.COM	408.530.5705	83
BEA WEBLOGIC	WWW.WEBLOGIC.BEASYS.COM	800.817.4BEA	2
BLUE SKY SOFTWARE	WWW.BLUE-SKY.COM	800.559.4423	23
CAREER CENTRAL	WWW.CAREERCENTRAL.COM/JAVA	888.946.3822	60
CAREER OPPORTUNITY ADVERTISERS		800.846.7591	85-93
CEREBELLUM SOFTWARE	WWW.CEREBELLUMSOFT.COM	888.862.9898	37
COMPUWARE NUMEGA	WWW.COMPUWARE.CON/NUBEGA	800.4.NUMEGA	6
CYSCAPE	WWW.CYSCAPE.COM/FREE4J	800.932.6869	78
DEVELOPMENTOR	WWW.DEVELOP.COM	800.699.1932	83
ELIXIR TECHNOLOGY	WWW.ELIXIRTECH.COM/	65 532.4300	51
ENTERPRISESOFT	WWW.ENTERPRISESOFT.COM	510.742.6700	11
FIORANO SOFTWARE, INC.	WWW.FIORANO.COM	408.354.3210	33
FORCE 5 SOFTWARE, INC.	WWW.FORCE5.COM	408.735.0665	53
GEEK CRUISES	WWW.GEEKCRUISES.COM		67
IAM CONSULTING	WWW.IAMX.COM	212.580.2700	61
INETSOFT TECHNOLOGY CORP	WWW.INETSOFTCORP.COM	732.235.0137	75
INSIGNIA SOLUTIONS, INC.	WWW.INSIGNIA.COM	800.848.7677	57
INSTANTIATIONS INC.	WWW.INSTANTIATIONS.COM	800.808.3737	26
JAVA BUYER'S GUIDE	WWW.JAVABUYERSGUIDE.COM	914.735.0300	77
JDJ CONSULTING SERVICES	WWW.JAVADEVELOPERSJOURNAL.COM	800.713.5111	84
JAVA DEVELOPER'S JOURNAL	WWW.JAVADEVELOPERSJOURNAL.COM	914.735.0300	79
JDJ STORE	WWW.JDJSTORE.COM	888.303.JAVA	42-43
KL GROUP INC.	WWW.KLGROUP.COM/PAGELAYOUT	888.328.9599	67
KL GROUP INC.	WWW.KLGROUP.COM/SWINGSUITE	888.328.9596	21
KL GROUP INC.	WWW.KLGROUP.COM/COLLECT	888.3289597	96
METAMATA, INC.	WWW.METAMATA.COM	510.796.0915	45
NEW ATLANTA	WWW.NEWATLANTA.COM/	678.366.3211	29
OBJECT DESIGN	WWW.OBJECTDESIGN.COM/JAVLIN	800.962.9620	48-49
OBJECT INTERNATIONAL SOFTWARE	WWW.OI.COM	919.772.9350	39
OBJECTSWITCH CORPORATION	WWW.OBJECTSWITCH.COM	415.925.3460	35
PALM COMPUTING, INC.	WWW.PALM.COM		69
POINTBASE	WWW.POINTBASE.COM/DEVLIC/JDJ	877.238.8798	27
PROTOVIEW	WWW.PROTOVIEW.COM	800.231.8588	3
PROTOVIEW	WWW.PROTOVIEW.COM	800.231.8588	73
QUICKSTREAM SOFTWARE	WWW.QUICKSTREAM.COM	888.769.9898	30
RIVERTON SOFTWARE CORPORATION	WWW.RIVERTON.COM	781.229.0070	47
SD 99 EAST	WWW.SDEXPO.COM	800.441.8826	82
SEGUE SOFTWARE	WWW.SEGUE.COM	800.287.1329	17
SIGS CONFERENCE FOR JAVA DEVELOPMENT	WWW.JAVADEVCON.COM	212.242.7515	80-81
SILVERSTREAM SOFTWARE, INC.	WWW.SILVERSTREAM.COM	888.823.9700	95
SL CORPORATION	WWW.SL.COM	415.927.1724	59
SLANGSOFT	WWW.SLANGSOFT.COM	972.375.18127	16
SLANGSOFT	WWW.SLANGSOFT.COM	972.375.18127	56
SOFTWIRED INC.	WWW.JAVAMESSAGING.COM	(41) 1.445.2370	7
SUN MICROSYSTEMS INC.	WWW.SUN.COM/SERVICE/SUNED/JAVA2	800.422.8020	4
SYBASE INC.	WWW.SYBASE.COM	800.8.SYBASE	25
THE THEORY CENTER	WWW.THEORYCENTER.COM	888.843.6791	71
TIDESTONE TECHNOLOGIES	WWW.TIDESTONE.COM	800.922.9665	31
UNIFY CORPORATION	WWW.EWAVECOMMERCE.COM	800.GO.UNIFY	13
VISICOMP, INC.	WWW.VISICOMP.COM	831.335.1820	15
VISUALIZE INC.	WWW.VISUALIZEINC.COM	602.861.0999	66
VSI	WWW.VSI.COM/BREEZE	800.556.4VSI	41
WORLDWIDE INTERNET PUBLISHING	WWW.WIPC.NET	800.785.6170	65

Advantsoft

www.avantsoft.com

Develop Mentor

www.develop.com/courses/ijava.htm

Java
Business Conference

www.javabusinessconference.com

Career Opportunities

Career Opportunities

Career Opportunities

Career Opportunities

Career Opportunities

Career Opportunities

Career Opportunities

Career Opportunities

IMI Systems Inc.

www.imisys.com

Java OOP Means OODBMS—*"Not"*

WRITTEN BY BRUCE SCOTT

There are many reasons for Java's success. Although heavily debated and discussed, the "Write Once Run Anywhere" aspect of Java is one of the reasons. A sometimes less-heralded reason is Java's superior object-oriented implementation. I don't view myself as an OO expert, but I've observed the rapid market acceptance of Java as compared to other OO languages. In the past these languages took many years to enter the mainstream (C++) or remained in a niche market (SmallTalk). The creators of Java were able to stand on the shoulders of others that had created OO languages; they were able to capitalize on the tremendous strength of OO programming and avoid the pitfalls. With Java, they created a language and platform that's allowing OOP to rapidly enter the mainstream.

With my background in database technology, I've seen similar sea changes in databases. Before relational databases there were network and hierarchical databases. Relational databases were the new kid on the block, and it took Oracle's success to show that these early database models could be replaced. In the mid to late eighties, some believed that relational databases would be replaced by another model – object-oriented databases. In Silicon Valley hundreds of millions of dollars were invested in OODBMS start-up companies. In the last ten years the market has made an overwhelming choice. Today RDBMS is a multibillion-dollar ($10 billion+) market and OODBMS is only a $200–300 million market. OODBMS companies have found themselves relegated to a niche market providing very specialized solutions.

Now with Java and OOP entering the mainstream the proponents of OODBMS are hanging their hat on promoting the belief that OOP requires an OODBMS. They're hoping that Java programmers will believe that the same advantages that are derived from OOP will also come from using an OODBMS.

What's missing here is that one of Java's strengths is its ability to connect the old world with the new. The new world includes application servers, mobile computing and Internet devices. The old world contains a vast amount of relational data, among other things. Sun has spent a lot of effort on JDBC because they understand the importance of providing Java programmers with the ability to access the vast amount of relational data available. With JDBC Java programmers can access all the world's relational data, and as they move from project to project, or job to job, the skills they acquire in working with an RDBMS are easily transferred. Contrast this with working on a project that uses an OODBMS. The likelihood that the next project or job will use an OODBMS are about 100 to 1 based on the relative market sizes. Since OODBMSs are in a niche market, the Java programmer who spends time on an OODBMS application will be too.

The question to be asked is: "Why has RDBMS maintained a dominant position for over 20 years? Is it because Larry Ellison is a tremendous marketeer and has managed to fool us all?" Larry is good but not that good. The answer may be too simple to believe. The relational model has stood the test of time simply because it's good. Most of the world's data isn't terribly interesting or complex and fits quite nicely into simple rows and columns. The complexity comes from the ever-evolving use of the data. RDBMS with SQL allows the use of the data to evolve in unpredicted ways. Yes, there are special situations where an OODBMS approach works best, such as parts explosions, CAD or mapping. These special cases can now be solved with OO extensions to the relational model found in the SQL '99 standard or with object-to-relational mapping tools.

> "Why has RDBMS maintained a dominant position for over 20 years? Is it because Larry Ellison is a tremendous marketeer and has managed to fool us all?"

There are many reasons for the success of RDBMS over OODBMS. Unlike OODBMS the data relationships don't have to be decided ahead of time. As the uses of data evolve the RDBMS can adapt through the support of joins and other dynamic aspects of SQL, and by easily altering the relational schema. SQL provides an industry-wide standard for accessing and managing data. There's no equivalent industry-accepted query language for OODBMS. One must write a procedural program for every new use of the data. The RDBMS market is supported by a large aftermarket that includes application development tools, report generators, design tools, etc. No such aftermarket exists for the OODBMS market. The list could go on.

We all want Java to succeed and it will. Java's success will grow as we bridge Java to the highly successful RDBMS market. Java OODBMS applications will find themselves in the OODBMS niche. These applications will only detract from the momentum of Java because they'll be isolated.

At PointBase we strongly believe in this, so we want to help every Java programmer learn how to write Java applications that use an RDBMS. On our Web site is a free developers' version of our 100% Pure Java database. A Java programmer can get up and running in minutes and try their hand at developing an RDBMS application. It's compatible with Oracle but doesn't require the complexities or expense of buying and installing Oracle. We encourage Java programmers to download this free PointBase developers' version so they can learn how to use RDBMS with Java. On our Web site we also host *Developers Central* to help answer questions on how to build RDBMS applications in Java. In our own way we're doing our part to help Java succeed by connecting Java to the lucrative world of relational data.

AUTHOR BIO

Bruce Scott, president, CEO and founder of PointBase, is a leader in the area of enterprise and embedded database architecture and product development. A cofounder of Oracle in 1977, Bruce cofounded Gupta Technology in 1984, pioneering the notion of the small footprint database server for Intel-based platforms.

bruce.scott@pointbase.com

Silverstream

www.silverstream.com

KL Group

www.klgroup.com/collect