JAVAEDGE CALL FOR PAPERS DEADLINE APRIL 16 P.93

JAVA™ DEVELOPER'S JOURNAL

The World's Leading Java Resource

March 2001 Volume: 6 Issue: 3

JAVADEVELOPERSJOURNAL.COM

Sept 23–26, 2001 Santa Clara, CA
JAVAEDGE conference&expo 96

Oct 22–25, 2001 New York, NY
XMLEDGE conference&expo 105

RETAILERS PLEASE DISPLAY
UNTIL MAY 31, 2001
$4.99US $6.99CAN

SYS-CON MEDIA

BASED E-BUSINESS MESSAGING PROVIDERS

WRITTEN BY DAVE CHAPPELL AND BILL WOOD PAGE 8

JAVA DEVELOPERS JOURNAL

EDITORIAL ADVISORY BOARD
JEREMY ALLAIRE, TED COOMBS, ARTHUR VAN HOFF, JIM MILBERY, GEORGE PAOLINI, KIM POLESE, SEAN RHODY, RICK ROSS, AJIT SAGAR, BRUCE SCOTT, RICHARD SOLEY, ALAN WILLIAMSON

FOUNDING EDITOR/CHIEF CORPORATE EDITOR:	SEAN RHODY
EDITORIAL DIRECTOR:	JEREMY GEELAN
EDITOR-IN-CHIEF:	ALAN WILLIAMSON
EXECUTIVE EDITOR:	M'LOU PINKHAM
ASSOCIATE ART DIRECTOR:	LOUIS F. CUFFARI
MANAGING EDITOR:	CHERYL VAN SISE
EDITOR:	NANCY VALENTINE
ASSOCIATE EDITORS:	JAMIE MATUSOW
	GREGORY LUDWIG
EDITORIAL INTERN:	NIKI PANAGOPOULOS
TECHNICAL EDITOR:	BAHADIR KARUV, PH.D.
PRODUCT REVIEW EDITOR:	JIM MILBERY
INDUSTRY NEWS EDITOR:	LEE PARSEGHIAN
J2EE EDITOR:	AJIT SAGAR

WRITERS IN THIS ISSUE
BILL BALOGLU, ERIK BERGENHOLTZ, DAVE CHAPPELL, KHANH CHAU, SREEDHAR CHINTALAPATY, TIM DEBOER, STEVE FRANCISCO, MICHAEL LACY, TONY LOTON, LARRY MCCAY, CRAIG MCCLANAHAN, JAMES MCGOVERN, ELLEN MCKAY, JOE MITCHKO, BILLY PALMIERI, SEAN RHODY, RICK ROSS, ARTHUR RYMAN, CHRIS SIEMBACK, JASON WESTRA, ALAN WILLIAMSON, JOE WINCHESTER, BILL WOOD

SUBSCRIPTIONS
FOR SUBSCRIPTIONS AND REQUESTS FOR BULK ORDERS, PLEASE SEND YOUR LETTERS TO SUBSCRIPTION DEPARTMENT
SUBSCRIPTION HOTLINE: SUBSCRIBE@SYS-CON.COM
COVER PRICE: $4.99/ISSUE
DOMESTIC: $69.99/YR. (12 ISSUES)
CANADA/MEXICO: $79.99/YR. OVERSEAS: $99.99/YR.
(U.S. BANKS OR MONEY ORDERS). BACK ISSUES: $10/EA., INTERNATIONAL $15/EA.

PUBLISHER, PRESIDENT, AND CEO:	FUAT A. KIRCAALI
VICE PRESIDENT, PRODUCTION & DESIGN:	JIM MORGAN
SENIOR VICE PRESIDENT, SALES & MARKETING:	CARMEN GONZALEZ
VICE PRESIDENT, SALES & MARKETING:	MILES SILVERMAN
VICE PRESIDENT, SYS-CON EVENTS:	CATHY WALTERS
SALES EXECUTIVE, EXHIBITS:	MICHAEL PESICK
ADVERTISING ACCOUNT MANAGERS:	ROBYN FORMA
	MEGAN RING
	RONALD J. PERRETTI
ASSOCIATE SALES MANAGERS:	CHRISTINE RUSSELL
	CARRIE GEBERT
	ALISA CATALANO
VICE PRESIDENT, CIRCULATION:	AGNES VANEK
NEWSSTAND SALES MANAGER:	CHERIE JOHNSON
ART DIRECTOR:	ALEX BOTERO
ASSOCIATE ART DIRECTORS:	CATHRYN BURAK
	RICHARD SILVERBERG
ASSISTANT ART DIRECTORS:	ABRAHAM ADDO
	AARATHI VENKATARAMAN
WEBMASTER:	ROBERT DIAMOND
WEB DESIGNERS:	STEPHEN KILMURRAY
	GINA ALAYYAN
WEB DESIGNER INTERN:	PURVA DAVE
JDJSTORE.COM:	ANTHONY D. SPITZER
ASSISTANT CONTROLLER:	JUDITH CALNAN
CREDIT & COLLECTIONS:	CYNTHIA OBIDZINSKI
ACCOUNTS PAYABLE:	JOAN LAROSE

EDITORIAL OFFICES
SYS-CON MEDIA 135 CHESTNUT RIDGE RD., MONTVALE, NJ 07645
TELEPHONE: 201 802-3000 FAX: 201 782-9600
JAVA DEVELOPER'S JOURNAL (ISSN#1087-6944) is published monthly (12 times a year) for $69.99 by SYS-CON Publications, Inc., 135 Chestnut Ridge Road, Montvale, NJ 07645. Periodicals postage rates are paid at Montvale, NJ 07645 and additional mailing offices. POSTMASTER: Send address changes to: JAVA DEVELOPER'S JOURNAL, SYS-CON Publications, Inc., 135 Chestnut Ridge Road, Montvale, NJ 07645.

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

SEAN RHODY, FOUNDING EDITOR / CHIEF CORPORATE EDITOR, JDJ

Sunset on the Evil Empire

Maybe I don't understand marketing concepts. It seems to me that advertising one of your products as 13 times more reliable than another of your products is not the optimal approach. In my mind, you'd want to emphasize its reliability compared to your competitors' products. But hey, nobody hired me to do marketing for Microsoft, so what do I know?

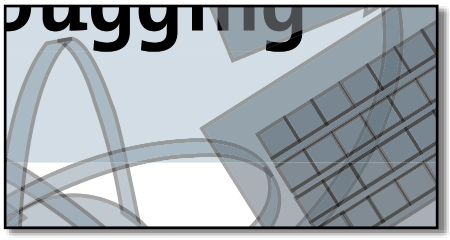
It's just that I have to laugh every time I see their two-page spread in a magazine, advertising how Windows 2000 Server is that much more reliable than Windows 98. That's not progress – my toaster is more reliable than Windows 98. If Ford advertised two cars, one of which was 13 times less likely to crash, which would you buy? Me, I'd buy a different brand of car. I'd want to know that my car is safer and more reliable than all the others out there, not just a Yugo.

And where do they get off being the world's number one software vendor and having the least reliable operating system? I'm missing something. I can't go a day without rebooting my Millennium Edition machine. Please, Microsoft, get Whistler right so I can get off this horrible platform and still play interesting PC games. Because that's the only reason I run Windows Me at home. It's certainly not for the exquisite joy of watching a brand-name machine lock up every few hours because it runs out of system resources with 96MB of RAM and only two open applications. Sheez.

Since this column has turned into one big Microsoft rant, allow me to continue to rail away at the resolution of the Sun–Microsoft lawsuit. Like some of you, I use Microsoft products every day. My company mandates that, but in most cases the products are good enough that I use them at home as well. Like it or not, they are de facto standards for much of the world.

So it pains me to see the door slammed on any chance of Java integration with the Microsoft platform. It shortchanges millions of users who could have benefited from the integration of the language into the product.

Microsoft, not to be caught flatfooted, has



come out with their own "migration" strategy, called JUMP to .Net, to ensure that people everywhere who used Java on Microsoft will not be left in the lurch. Sure, just move off Java. JUMP stands for Java User Migration Path, and it outlines several different approaches to move Java code to the .Net platform. None of these includes running Java code. Given its portability, perhaps many will simply JUMP to UNIX. Or continue to ignore the Microsoft Internet products and run the Java-based products on NT or 2000, as many shops do.

In the meantime, Microsoft will continue to become even more irrelevant to the Java community. We'll still use the basic operating system software, but it's a darn shame we couldn't get real support from Microsoft for the language. Oh well.... Time to reboot my toaster. No, wait, I mean my Windows box.

sean@sys-con.com

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

ALAN WILLIAMSON, EDITOR-IN-CHIEF

Do Magazines Have Version Numbers?

You may have noticed a slight change in this month's editorial: namely, me. Don't be alarmed; it's all under control. Like many a good piece of software, the trick is never to assume that what you have will continue to be what you need. Reinvention is paramount to keep ahead in what can only be described as a lightning-paced industry.

Accordingly, your beloved ***JDJ*** is about to get a version increase. At the moment, we're going into version 1.1, with a view to a complete move to version 2.0 in three to four months' time.

JDJ, now in its sixth year of serving you, the Java developer, has grown from strength to strength in those years. We have followed the trends in this corner of the industry and reported them to you as best we could. Considering the speed at which Sun was churning out new versions of the API, it was a task that ensured we never had time to sit on any laurels. With reference to that, I'd like to give special thanks to the man who steered the ship so well through those waters and brought us to the point we are at today: our founding editor-in-chief, Sean Rhody. I inherit a great legacy.

As you know, Java has moved on significantly since its advent just over six years ago. The world is starting to get serious about this new kid on the block, noting its move into areas that traditionally have never been threatened by such a surge. I hope that, with your help, we can continue to bring you the information and news you require to assist you in shaping the industry in which we all work and operate.

Our readers are evolving. No longer are we exclusive to the Java development community; a new wave of people is pouring over this monthly digest of Java news, looking for ideas, looking for assistance, looking for direction. We want to be able to serve them – you – better. We are in this Java universe as a team, and to that end we have to work like a team to ensure that the success of Java is secured.

Sun Microsystems recently announced their new strategy, Sun ONE. (An interview regarding this announcement appears elsewhere in this issue.) Sun is striving for a more open and communicative industry. They want us to start using tools and standards that won't lock us in to a particular vendor. They recommend Java and XML, among other things. The very fact that you are already using Java to deploy your solutions is a bit like preaching to the choir. So hurrah to you, that has chosen the open route.

I am fortunate in the position that I hold with respect to n-ary. We are at the forefront in deploying server-side Java solutions to our client base. We can see what clients want, and, more important, get a firsthand reaction to their concerns and fears. These concerns are the ones we want to continue to address each month in ***JDJ***. The magazine will be going through a number of transformations over the next few months to better address *all* the areas of Java, not just those at the server side.

I invite all of you to be part of that change. I want you to tell us what we can do better. I want you to feel part of this new version change. So help us help you by joining our mailing list – http://myjdj.sys-con.com/mailman/listinfo/myjdj – and by letting us know what you think. In addition to adding your name to the mailing list, please bookmark my e-mail address. And don't hesitate to e-mail me directly. I promise to get back to you. If I personally can't solve your problems or respond to your concerns, our excellent team here at **SYS-CON Media** will be able to assist you.

Next month I'll be introducing you to some old and new faces that will be helping us shape this version change to better serve you.

alan@sys-con.com

AUTHOR BIO

Alan Williamson holds the reins at n-ary (consulting) Ltd, one of the first companies in the UK to specialize in Java at the server side. Rumor has it that he welcomes all suggestions and comments.

FEATURE

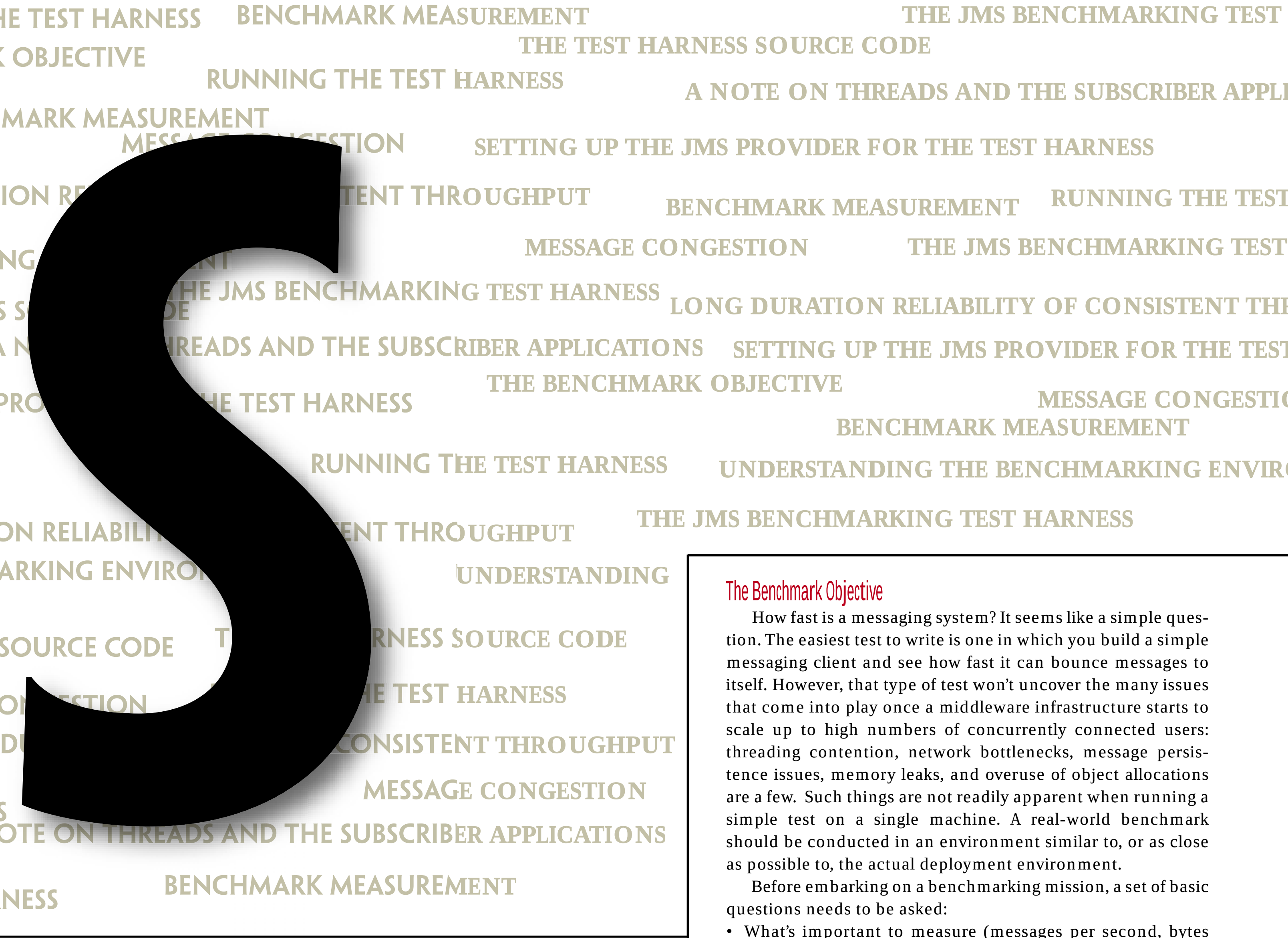
BENCHMARKING JM

BASED E-BUSINESS MESSAGING PROVIDERS

Benchmarking any distributed computing middleware product is a complex task. Knowing how well a distributed infrastructure will perform under heavy load with a large number of concurrently connected users is a key factor in planning a development and deployment strategy.

With the advent of Java Message Service (JMS) as the standard for a global class middleware infrastructure, development organizations can enjoy the luxury of building distributed applications using a common set of APIs and message delivery semantics. At the same time they can pick and choose from a variety of JMS-compliant vendor implementations.

However, the science of testing global-class messaging middleware is uncharted territory as there are no industry-accepted benchmarks. There are many vendor implementations to choose from, all of which claim to be the fastest. Some also claim to be highly scalable. To date, there are no publicly available multiconnection, multitopic,



S
THE JMS BENCHMARKING TEST HARNESS
BENCHMARK MEASUREMENT
THE TEST HARNESS SOURCE CODE
RUNNING THE TEST HARNESS
A NOTE ON THREADS AND THE SUBSCRIBER APPLICATIONS
MESSAGE CONGESTION
SETTING UP THE JMS PROVIDER FOR THE TEST HARNESS
LONG DURATION RELIABILITY OF CONSISTENT THROUGHPUT
THE BENCHMARK OBJECTIVE
UNDERSTANDING THE BENCHMARKING ENVIRONMENT

WRITTEN BY DAVE CHAPPELL AND BILL WOOD

and multiqueue benchmarking test tools to validate these claims.

This article outlines a methodology for conducting a large-volume benchmark using any JMS implementation, focusing on the issues involved in properly measuring performance throughput and scalability. It's intended to help you build your own tests that will provide some objective validation of messaging systems for Java programmers. Source code for the test harness is presented and explained in Listings 1- 5 on the *JDJ* Web site.

The Benchmark Objective

How fast is a messaging system? It seems like a simple question. The easiest test to write is one in which you build a simple messaging client and see how fast it can bounce messages to itself. However, that type of test won't uncover the many issues that come into play once a middleware infrastructure starts to scale up to high numbers of concurrently connected users: threading contention, network bottlenecks, message persistence issues, memory leaks, and overuse of object allocations are a few. Such things are not readily apparent when running a simple test on a single machine. A real-world benchmark should be conducted in an environment similar to, or as close as possible to, the actual deployment environment.

Before embarking on a benchmarking mission, a set of basic questions needs to be asked:

- What's important to measure (messages per second, bytes per second)?
- How are the results to be interpreted?
- Should I measure send rates? Receive rates? Both?
- How often should measurements be sampled?
- What's the minimum necessary duration?
- How many JMS senders and receivers do I need to use to get a realistic view of my expected deployment scenario? What's the ratio of senders/receivers? Many to one? One to many, or many to many?
- What are the diverse scenarios (message size, persistent, nonpersistent, pub/sub, point-to-point)?

The last two issues are application specific, and usually not crisply defined early in the development process. Even if the deployment topology and size of the user community is well known at the onset of a project, it's likely to grow over time. Therefore, a key criterion when benchmarking middleware is its ability to scale beyond your initial deployment requirements. The number of senders and receivers, the diversity of message sizes, and the number of pub/sub topics and point-to-point queues are likely to increase. The surest way to account for this growth is to test the largest number of senders and receivers that your test hardware environment can handle.

Benchmark Measurement

The goal of the benchmark is to measure the overall throughput of messages through the system under a given load condition. The measurements need to span from the first send to the last receive, as illustrated in Figure 1. Figure 1 shows the time line associated with the large-scale send/receive test. The important figures that will be derived from this test are the overall length of the test and, from that, the average send and receive (throughput) rate the server actually achieves. The test is designed so there's a "ramp-up" period where all the connections, senders, and receivers are established. The measurement begins when the first message is sent. The measurements continue at designated time intervals until the number of designated time intervals has transpired. This effectively measures the JMS provider's "steady-state" throughput while all senders and receivers are actively processing messages.

Message Congestion

Message traffic in a deployed environment is often unpredictable. Bursts in message traffic can occur for extended periods of time. It's important to measure and analyze both the send rates and the receive rates under these high-volume conditions. It's feasible that messages may be produced far faster than they can be consumed (one of the benefits of using a MOM), causing a buildup of messages at the JMS provider layer. If this condition occurs for an extended period of time, message congestion can occur. Such a condition will result in a no-win situation for all messages flowing through the system. Nonpersistent messages will fill up in-memory queues, and eventually reach the limits of the server machine(s) that the JMS provider resides on. Persistent messages will constantly flow into persistent store. The net result of this is thrashing of the server machine, thus further prohibiting the JMS provider from achieving its ultimate goal – delivering messages to their intended destinations.

In such a situation, it's prudent to "throttle" the flow of messages from the message producers through the use of "Flow Control." This ensures a smooth and timely delivery of all messages through the system while still allowing the JMS server processes to behave properly within the physical restraints of the machines they're running on.

Flow control implementations vary from throwing an exception to the sending client (requiring the client code to somehow know when to start sending again), to automatically instructing the senders when to slow down and when to pick up again, to discarding the message altogether.

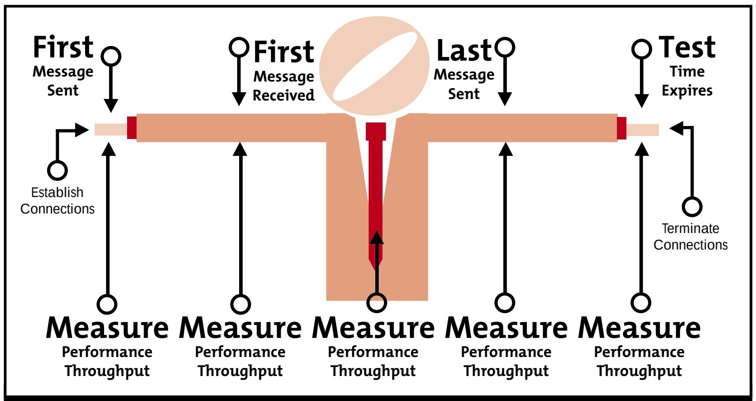

First Message Sent
First Message Received
Last Message Sent
Test Time Expires
Establish Connections
Terminate Connections
Measure Performance Throughput
Measure Performance Throughput
Measure Performance Throughput
Measure Performance Throughput
Measure Performance Throughput

FIGURE 1 Large-scale send/receive test

Long Duration Reliability of Consistent Throughput

When your application goes into production, it's likely you'll be expecting it to operate on a 24x7 schedule, receiving and sending messages for long periods of time. Any benchmark undertaken should be designed to reflect these requirements, and as such you'll need to have your benchmark running over a period of time. Ideally you'd be able to test for months, but usually delivery time scales and resources prevent you from being able to do that. A quick test of a few thousand messages won't be enough to really indicate any long-term trends. Thirty minutes is a good rule of thumb as a bare minimum for any test, whether the goal is to measure raw throughput or to measure long duration reliability.

"THIS DISCUSSION IS NOT INTENDED TO IMPLY THAT **JMS** IS A RISKY PROPOSITION"

It's really important that you at least run some overnight and weekend tests. Any trends such as message congestion, memory growth, or general inconsistency in results are always detectable within that timeframe.

Key things to look for are whether:

- The server continues to perform.
- Performance at the end of such long tests is consistent with the numbers over shorter durations.

This discussion is not intended to imply that JMS is a risky proposition. The fundamental issues discussed here would apply whether it's based on CORBA, RMI, DCOM, or MOM, and should be tested before cementing your deployment schedules.

Understanding the Benchmarking Environment

It's imperative that the hardware environment used in the benchmarking effort be as close as possible to the actual deployment hardware. For example, performing a benchmark test on a notebook computer will yield results that are vastly different from a test run on a set of high-powered server machines. The test results on different types of machines will not be proportionate due to bottlenecks that will occur in different places.

The following checklist is an example of the kinds of concerns to be aware of:

- Running on a single machine won't reveal any variations in test results that could occur due to network latency. It's critical to test on at least two machines.
- Benchmarking with 50 senders/receivers will not yield the same relative results as with 1,000 senders/receivers.
- Running on a 10Base-T network versus a 100Base-T network may create an artificial network bottleneck that would mask the true stress testing of the JMS server (or servers).
- If you're using nonpersistent messages, the speed of the server processor and memory will affect performance.
- Regardless of the efficiency of the JMS provider's persistence mechanism, the speed of the disk is crucial in persistent messaging cases. Even with powerful RAID arrays, results can be skewed 20-fold depending on the speed of the disk.
- When using a caching disk controller, it's important that the write caching be disabled. For true reliability and recovery, absolute disk writes need to happen in the event of a failure.
- Due to factors such as network traffic, time of day, and unpredictable variation in your setup, you should try running key tests many times and comparing the results. You might be surprised by the variation between runs (or, hopefully, not).

The JMS Benchmarking Test Harness

In your actual deployment, all your senders and receivers will usually be running on different machines. It's important that at least two machines be used in order to gain an understanding of network latency issues. However, it may not be reasonable for you to allocate 1,000 machines to perform your test. In recognition of this, the test harness code provided simplifies multimachine complexities by allowing a framework for creating multiple senders or receivers within a single application. These applications can be configured to simulate multiple, separate JMS clients. Each sender and receiver can have its own connection to the JMS server within its own dedicated thread.

For simplicity, two main components of the test harness – the sender application and the receiver application – have been kept separate. This allows the benchmarker to run senders and receivers on different machines, and look at sending rates and receiving rates separately (see Figure 2).

If you have access to multiple machines, you may spread these across as many as you like, as shown in Figure 3. The real goal is to focus on stress testing the JMS server(s). That's where the focal point of the message traffic is likely to be. Be careful not to overload the client machines with too many JMS clients. This will create an artificial bottleneck at the client side that won't exist in the real deployment environment. A good rule of thumb is to monitor the CPU and memory usage on the client machines, and make sure that the steady-state processing of messages doesn't exceed 80% utilization of either CPU or memory consumption.

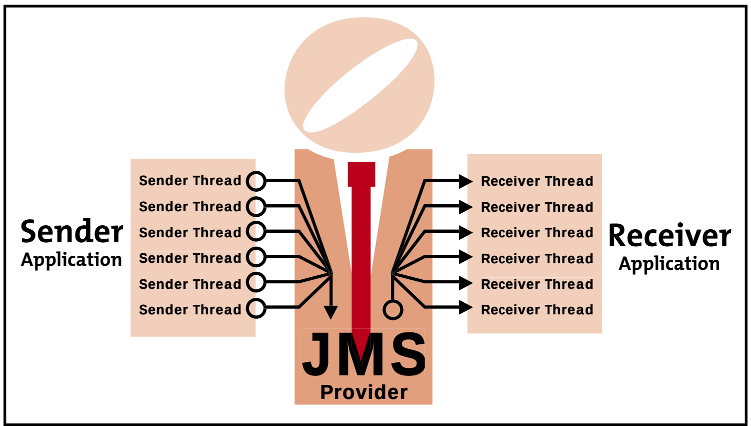


FIGURE 2 Multiple JMS clients can be simulated by a single application

Ideally you should have a full, nonrestricted version of each product you're testing. In the absence of that, note that most of the free developer versions of JMS that you can download from the various vendor Web sites are limited by the number of connections or number of machines that can connect to the JMS server. Upward linear scalability trends, or lack of, don't usually start becoming obvious until you have at least 50 senders and 50 receivers; 200–300 pairs are even better.

In recognition of these issues, this test harness gives the flexibility to specify multiple JMS sessions per connection, so you can go beyond any connection limit that might be imposed. The test harness code is also designed to allow you to specify the number of senders or receivers, how they connect to the JMS server, and the number of senders that should share a given connection.

If you're limited to two machines, the JMS server should be run on one machine, and the sender application and the receiver application should be run in separate JVMs on the other one (see Figure 4).

The Test Harness Source Code

To explain the test harness, we'll focus on the publish/subscribe messaging model using a many-to-many scenario. The coding framework is similar in the point-to-point queuing version of the test harness. The fully functional versions for both models, including any vendor-specific issues, can be downloaded from www.SonicMQ.com.

"BE CAREFUL NOT TO OVERLOAD THE CLIENT MACHINES WITH TOO MANY **JMS** CLIENTS"

For simplicity, code for point-to-point is kept separate from publish/subscribe. The full test suite, therefore, for tests of SonicMQ include the following four Java files:

- PtpSenderApp.java
- PtpReceiverApp.java
- PubSubPublisherApp.java
- PubSubSubscriberApp.java

All the code samples follow the same basic pattern:

1. Basic parameters are read in from the command line.
2. A separate object is created for each sender/receiver. Each object is managed by its own thread.
3. All the JMS objects are started.
4. The application pauses, waiting for the developer to verify the setup.
5. The application sends or receives messages as fast as possible.
6. The application queries each sender/receiver once per interval and computes average messages sent/received per second. This result is sent to the console. (A rolling average is also reported.)
7. After a prespecified number of intervals, the application closes all JMS objects, stops reporting, and exits cleanly.

Let's look at some of the code that implements these steps, focusing on the PubSubPublisherApp, for example.

The help screen in Listing 1 shows the usage of the publisher application. The main() method simply reads in the data from the command line and creates an instance of the PubSubPublisherApp object, where the actual JMS work is performed. Sender application parameters are shown, and similar arguments exist for the receiver applications and the Publish/Subscribe domain.

–continued on page 16

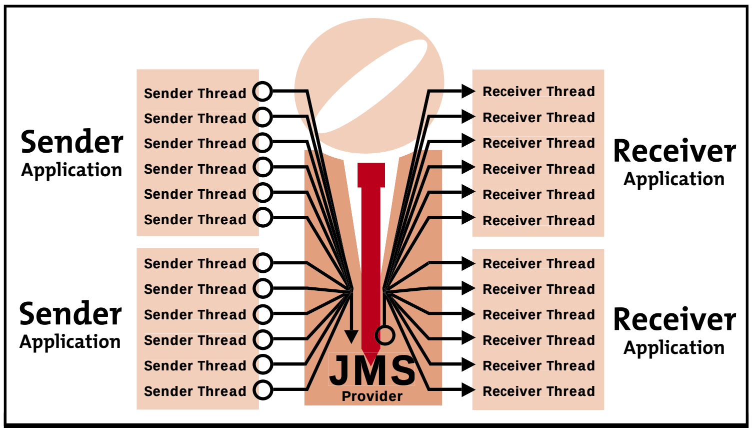


FIGURE 3 Multimachine test scenario

A typical usage scenario would be to run this for 50 publishers publishing 2K messages to 10 topics using a JMS server hosted on a remote machine, ntserver.

```
REM Assume all Java Classpath arguments have been set up.
java PubSubPublisherApp –b ntserver:2506 –npublishers 50
–ntopics 10 –msize 2
```

Once the arguments have been read in, an instance of the sender application object is instantiated. The basic code is shown in Listing 2.

One of the first lines of the construction is the line that finds the javax.jms.ConnectionFactory associated with the particular provider and messaging domain. In this case the factory is for TopicConnections and is retrieved using an InitialContext object that emulates the use of a third-party, external JNDI object store.

Because JMS is a developer API common to all JMS systems, the only provider-specific code you'll need to change is the code that sets up the InitialContext and its environment.

Once the ConnectionFactory has been instantiated, the desired number of connections is created. If possible, this should be equal to the number of actual senders, but due to evaluation limitations, you might be forced to use fewer connections.

The application then creates an array of publisher objects. These are runnable objects and will be associated with a unique thread object. Each of these objects is created and sequentially assigned to one of the connection objects created. Each sender will have its own JMS TopicSession and its own JMS TopicPublisher.

Similarly, the publisher objects are sequentially assigned to the number of queues created. For example, if you have 50 publishers, 25 connections, and 10 topics in the case you're evaluating, each of the 25 connections will have two publishers. The 50 publishers will likewise be uniformly divided among the 10 topics (five per topic).

Listing 3 shows the actual code for the publisher object itself. You'll note that it's runnable, as expected. The bulk of the JMS code is shown in the run() method. This method simply sends messages as fast as possible and records the number of messages sent in a counter, m_msgCounter.

Note that most of the remaining code in the PublisherObj object is related to reporting the results. It's the main() method of the PubSubPublisherApp that's responsible for controlling the start and stop of these objects, and for combining the results from all senders.

Listing 4 shows this portion of the code from the PubSubPublisherApp main() method. This is the code that loops through the array of publisher objects once per interval and queries their performance. At each interval, the application prints one line of the report. This line includes:

- Message sent per second (over the interval)
- Messages sent per second (averaged over the latest five intervals)
- Total messages sent in the interval
- Cumulative messages sent in the test

Similar rates for messages received are reported separately by the PubSubSubscriberApp.

A Note on Threads and the Subscriber Applications

The code for subscriber applications follows the same pattern as the publisher application in this test harness. One difference, however, is that the subscriber application doesn't explicitly manage the threads used for consuming messages. This is because the subscriber objects use the JMS asynchronous MessageListener interface (see Listing 5).

Setting Up the JMS Provider for the Test Harness

Depending on the JMS server you're using, you may have to administratively create the topics and/or queues needed in your test. For example, the default queues used by the PubSubPublisherApp would be TestTopic-1, TestTopic-2, TestTopic-3, and so forth (you can specify the number of topics to use in your test).

Similarly, the queue names used by the point-to-point test harness

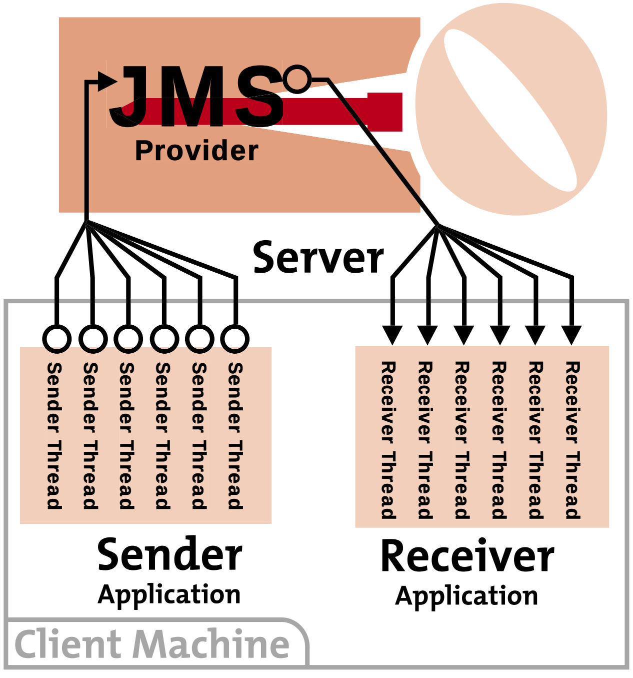
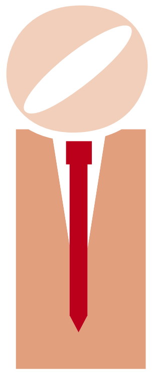


FIGURE 4 Test run on only two machines



```
==========================
==========================
CODE VERSION: 1.06
Broker url: ntserver:2506
Username: Administrator
Password: Administrator
Number of Subscribers: 50
Number of Connections: 25
Number of Sessions: 50
Number of Topics: 10
Topic name prefix: TestTopic-
Number of Reporting intervals: 90
Seconds per interval: 60
==========================
Please wait for PubSubSubscriberApp to finish setting up for test...
***Please press <ENTER> to start test intervals.***

---------------------------------------------------------------------------
Interval #
|
|       Interval Receive Rate (msg\sec)
|       |
|       |       Rolling Receive Rate (msg\sec) [Last 5] intervals
|       |       |
|       |       |       Interval Msg Total
|       |       |       |
|       |       |       |       Cumulative Msg Total
---------------------------------------------------------------------------
Interval 1   647.9   647.9   38874.0   38874.0
Interval 2   675.4   661.7   44577.0   83451.0
Interval 3   687.5   670.3   41252.0   124703.0
Interval 4   709.6   680.1   42579.0   167282.0
Interval 5   696.7   683.4   41803.0   209085.0
...
```

FIGURE 5 Sample output

(PtpSenderApp) are named TestQueue-1, TestQueue-2, TestQueue-3, and so forth. You'd need to administratively create the queues.

Running the Test Harness

On your server machine, make sure the JMS server has been started and all queues and topics needed for your test have been configured. The amount of setup will depend on the JMS server you're using. You should also make sure the settings for your server and its JVM allow for a liberal use of operating system resources.

You're now ready to configure the JMS client applications (e.g., PubSub SubscriberApp and PubSubPublisherApp). Typically, you'll start these applications in two separate console windows, possibly on separate machines.

The receiver application might typically be started first with a very long test duration. It will report the results interval by interval as you start and stop the publisher application or change the send rate. The major configurable options for the subscriber application are:

- Number of connections
- Number of subscribers
- Number of topics
- Duration of test (number of sampling intervals and interval time)

The main setup issue is to make sure the topics you're subscribing to match the name and count of the ones you plan to publish to. The major configurable options for the publisher application are:

- Number of connections
- Number of publishers
- Number of topics
- Message size
- Delivery mode
- Duration of test (number of sampling intervals and interval time)

For example, to set up the receiver application, the following would be a typical command line for a long duration test using 50 publishers, 25 connections, and 10 topics. We'll connect to a server on ntserver (port 2506) and run for 90-minute intervals.

```
java PubSubSubscriberApp –b ntserver:2506 –nsub-
scribers 50 –ntopics 10 –nintervals 90
```

After starting the subscriber application, we can start the corresponding application.

```
java PubSubPublisherrApp –b ntserver:2506 –nsub-
scribers 50 –ntopics 10
```

Both applications will report their setup configuration as well as their interval results. The sample output shown in Figure 5 would come from the subscriber application.

Summary

Remember these key points:

- Plan for burst rates and an overall larger-than-expected deployment scenario in terms of number of JMS clients, and number of topics and queues. Start with a reasonable number of clients (50), then move up to larger amounts in increments of 50–100. Watch for upward trends in throughput.
- Look at both the send rates and the receive rates. Watch out for message congestion, and understand what the flow control behavior is, if any.
- If you're going to deploy on some serious hardware, then test on some serious hardware.
- Run the test long enough to see some realistic results. Stress the server. Analyze CPU, memory, and disk usage.

Experiment! Have some fun with different message sizes and delivery options. By expanding your test criteria and gaining a thorough understanding of the overall performance characteristics of your global class middleware, you have a better chance for success in your deployed application.

Author Bios

Dave Chappell is vice president and chief technology evangelist for SonicMQ, Sonic Software's award-winning JMS e-business messaging server. He is also coauthor of Java Message Service *(O'Reilly).*

Bill Wood, a Sonic Software fellow, is a senior architect for SonicMQ and director of the SonicMQ performance and scalability team.

chappell@progress.com

wood@progress.com

Implementing J2EE Security with WebLogic Server

Use any type of data store as a security policy domain

Part 1 of 2

WRITTEN BY
JASON WESTRA &
CHRIS SIEMBACK

This month's article is the first in a two-part series on J2EE security. In Part 1 we'll discuss basic J2EE security. Part 2 will provide you with a model to set up and deploy a functioning security-enabled application. Resident J2EE security guru, Chris Siemback, has been kind enough to join me in coauthoring this series to contribute in-depth examples of J2EE security at work.

Our discussions will cover securing both Enterprise JavaBeans (EJBs) in the business and data layers as well as JavaServer Pages (JSPs) and servlets in the Web layer of a J2EE application.

J2EE Security Overview

Understanding the strong suits and weaknesses in the J2EE security model will help you determine the right security architecture for your application. A basic overview of J2EE security will help you compare its qualities to your security requirements. Let's come to grips with a few terms we'll see in the discussion that follows:

- ***Principal***
  A principal is an entity that can be authenticated in an enterprise application. The data and the content required to be authenticated may be different from implementation to implementation, but a principal is generally authenticated by a username and password combination.

- ***Security Policy Domain***
  A security policy domain, commonly called a realm or, just, security domain, is simply the implementation of a store containing authentication and authorization information and the enforcement of policies on this information as defined by a security administrator. Security domains are typically repositories where users, groups, permissions, and secured resources are stored. In other words, the realm contains the usernames, passwords, and roles that the container uses to authenticate and authorize user requests. A realm may be based on LDAP, UNIX, database, and NT security stores, or it may simply be XML documents that are read by the application server.

With so many choices in security domains, how do you choose which one to use in your enterprise application? Numerous factors are involved including politics, corporate standards, and features such as the ability to modify the realm and for the realm to cache information.

For instance, politics and technology often govern standards in the organizations with which we work. If an organization has standardized on NT, you may choose to leverage the existing NT security.

For large-scale applications, you should ensure that your implementation of a J2EE security domain has at least the ability to add, remove, and modify users, roles, privileges, and secured resources to and from the realm without having to shut down the server. Some implementations are statically loaded and do not allow reloading of security information. However, this forces downtime to reload modified information, which is generally not acceptable in today's online applications.

Also, a great feature to have in your realm is the ability to cache security information. Although it may change (as mentioned above) this information is generally static, and a high-performing application needs the ability to cache commonly used information at the application server level rather than retrieving it from a persistent store with each request.

- ***Credential***
  A credential either contains security attributes or refers to attributes for a principal. A credential is used to grant access to other secured services; thus it's passed along to each service accessed by the principal. If security attributes are not passed along, they may be retrieved from a trusted third party storing the information required for authentication and authorization.

- ***Authentication and Authorization***
  When you attempt to access restricted resources, the container is responsible for both authenticating and authorizing the request based on your credentials. Authentication is the process of verifying who you are. Authorization is the step made after authentication that verifies whether or not you may use a secured resource.

Now that you have a handle on the security vocabulary we'll be using, let's look at how J2EE security allows the authentication and authorization of principals within a security domain, providing access to secured resources such as Web components and EJBs.

Characteristics of J2EE Security

There are two main characteristics of J2EE security: it's role-based rather than user-based (object level), and declarative rather than code-based. J2EE security is designed to be a role-based, declarative approach to securing applications and resources through the use of security domains.

• ***Role-Based Security***

J2EE security authorizes access to resources based on the role of the user. An example of a role would be a member or system administrator. Once users have been authenticated (e.g., they exist in the security domain), they'll begin to access various resources while performing their business tasks. This will result in any number of security checks by the J2EE server against the credential (e.g., permissions) of the user with respect to the "role" the user assumed at login.

The ability to perform object-level security (security on a per-user basis) is often required in sophisticated enterprise applications. However, role-based security is incapable of performing object-level security because roles relate to a "group" level rather than focusing on an individual. For instance, there's no way to ensure that the user "John Doe" can view/modify his own employee records, but not other employee records. You could declare that users in the department manager role can view/modify all employee records in their department, however. If your application requires object-level security, be sure to architect it into your solution.

• ***Declarative Security***

The J2EE platform strives for portability across compliant servers. To this end, J2EE stresses the use of declarative security in Web applications (.wars) and EJBs, rather than promoting coding of security logic. Declarative security allows the J2EE server to intercept requests to access secured resources and perform authentication and authorization to use the service. No coding is involved on your behalf. However, the deployment of the component involves the extra step of determining which roles have the right to use a resource and then mapping these roles to the appropriate services.

Since mapping is performed at the deployment of the component, the component is portable across compliant J2EE servers such as WebLogic Server (www.bea.com) and iPlanet Application Server (www.iplanet.com).

Not only does J2EE support declarative security, it also provides an API into the security principals in the domain allowing programmatic checks. This feature is neither as portable, nor as configurable as declarative security, however, and should be used with caution.

Securing Web Resources

All J2EE products are required to support a single sign-on in the Web tier, so you shouldn't have to log in multiple times for different applications on a site. Furthermore, it specifies that authentication occurs only if a user attempts to access restricted resources. This is described as "lazy authentication," which means you may freely access Web-based resources if no security restrictions are placed on them. If you later attempt to access restricted portions of the site, the server must use one of three methods to authenticate the user:

• ***HTTP basic authentication***

HTTP basic authentication is the security method supported by the HTTP protocol. This is the familiar browser pop-up window that requests a username and password. For instance, you may have encountered it the last time you tried to hack your friend's secured site. Upon receiving the username and password, the Web server authenticates the username against the security domain. Note that basic authentication is not a secure method of sending cre-

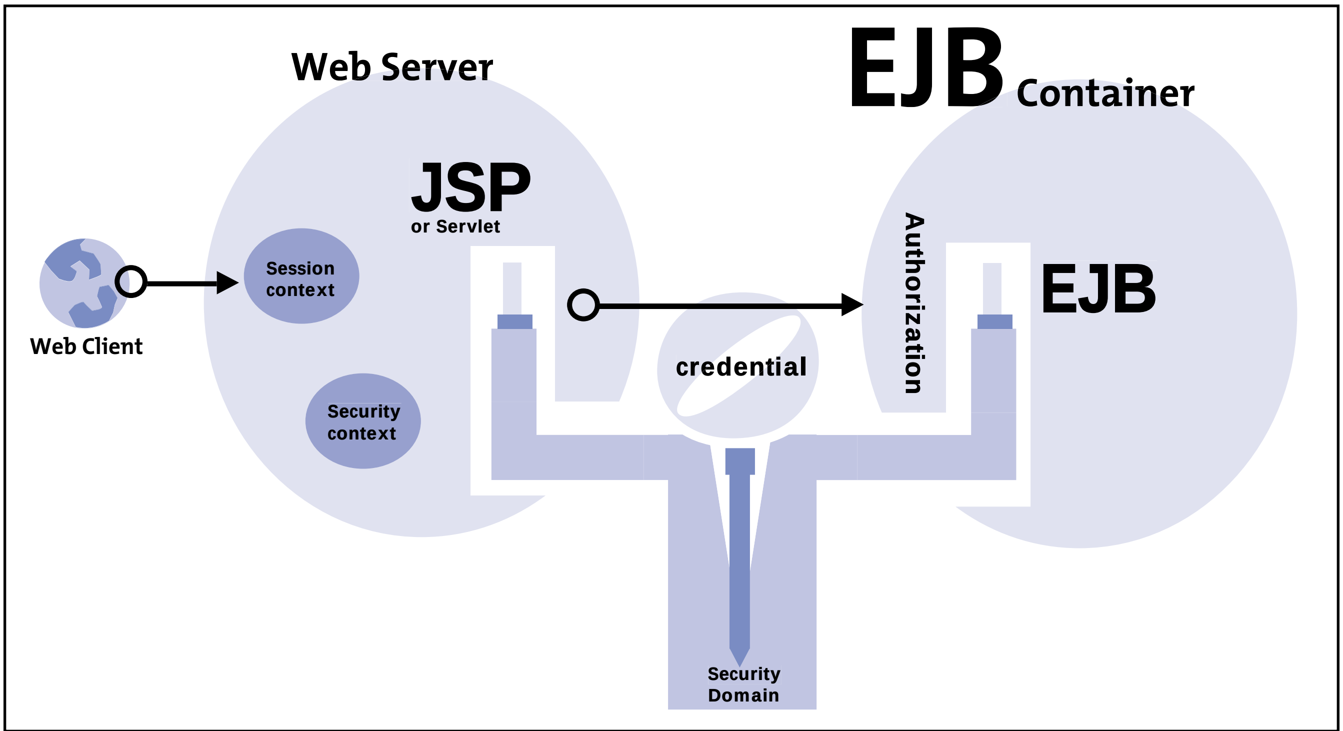


FIGURE 1 J2EE Security Model

dentials. Adding SSL can alleviate this concern, but there's no control of the look and feel of the browser-generated pop-up window.

• ***HTTPS Authentication***

HTTPS is HTTP over Secure Sockets Layer (SSL). SSL encrypts the data being sent between the client and the Web server using a "public/private" digital key scheme. This means that although someone may be able to intercept the data being transmitted, he or she will not be able to decrypt the information without the private key. This method of security is often used with applications that require credit card numbers, for example, purchasing a book at Amazon.com. This is all done automatically by the client and the server and doesn't require any input from you, the user.

• ***Form-Based Login***

Form-based authentication allows developers to create customer login pages for their users. When you attempt to access restricted resources, the custom login page is displayed. The credentials you provide are sent back to the server (preferably over SSL) and verified against the domain. If you have authorization, you're granted access to the resource, otherwise, a custom "failed login" page is presented.

For more information on Web-based security, see the Servlet 2.2 specification. It can be downloaded at www.javasoft.com/products/servlet/download.html.

EJB Security

In the spirit of J2EE's declarative security, the EJB architecture discourages (but does not prevent) the practice of hard-coding security logic in EJBs. Instead, EJB's security model allows developers to place restrictions on bean methods contained within the remote and home interfaces through its deployment descriptor.

During the assembly of an EJB, the application assembler is responsible for defining security roles for an application. This entails bundling sets of permissions into logical groups that restrict component access at the method level. Next, the deployer is responsible for mapping the defined security roles to groups within the enterprise's security domain. In many cases, the same individual performs these two roles.

During method invocation on an EJB component, the container intercepts the method call, enforcing the security restrictions defined during deployment for the EJB. The EJB container will allow a user to execute a method only if the caller's principal is contained within one of the defined security roles. Furthermore, as EJBs invoke methods on other EJBs in the system, the security context is passed along to each component. While this enables seamless security checks across the EJB layer, there's a performance penalty encountered while checking security on each method call.

For more information on EJB security, see the EJB 1.1 specification. It can be downloaded at www.javasoft.com/products/ejb/docs. html.

J2EE Security: Putting It All Together

In the previous sections, we reviewed how each layer in a J2EE application can be secured, but how do Web components and EJBs work together to provide a unified, single sign-on across all layers of your architecture? Figure 1 depicts a Web client accessing restricted resources such as JSPs, servlets, and EJBs. As you can see, each layer in the J2EE architecture (Web server and EJB container) provides its own authentication against the security domain.

The Web server first authenticates the user against the security domain, then confirms that the credentials supplied match to a principal (user) within the security domain. If the user doesn't exist, the authentication fails and the request for the specified resource is denied.

If authentication is successful, the server authorizes the request to access the secured resource, verifying that the user has been given adequate permission to use the resource. If the user has the required permissions for the resource, then the request is granted access to utilize its services; otherwise the server will deny the request. Notice that the credentials provided to the Web server are propagated to the EJB, providing seamless integration. In both cases, the container will enforce the security restrictions based on the deployment descriptor for the component.

Conclusion

J2EE security can be used to secure resources in a portable fashion. Because it's declarative in nature, no coding is necessary to implement it in either the EJB layer or the Web tier, providing transparent security for the developer. This keeps both your Web component and EJB code focused on presentation or business logic, and enhances portability across different servers. Furthermore, it lets businesses use any type of data store as a security policy domain, allowing them to leverage any existing security implementations they currently have.

J2EE security was designed to provide a portable and flexible solution. However, it's important to be aware of some of the limitations. If your application(s) requires a very fine-grain level of security, you may need to add programmatic security. This obviously reduces the portability of your code and opens up possible security holes.

Next month in **EJB Home**, we'll walk through a sample application based on BEA's WebLogic Server that demonstrates the concepts described here. We'll create a relational database security domain, then construct a secured Web-based application that communicates with EJBs. See you next month!

AUTHOR BIOS

Jason Westra is CTO at Verge Technologies Group, Inc., a Java consulting firm specializing in e-business solutions with Enterprise JavaBeans.

Chris Siemback, an Enterprise Java consultant at Verge Technologies Group, Inc., specializes in Web-based EJB applications, various development methodologies, and distributed architectures.

westra@sys-con.com

csiemback@vergecorp.com

FEATURE

Using the Java Platform Debugger Architecture

written by tony loton

Develop a new breed of debugging and profiling tools

The Java Platform Debugger Architecture (JPDA) provides a standard set of protocols and APIs at three levels that facilitate the development of a new breed of debugging and profiling tools. The inclusion of JPDA in the Java 2 SDK enables individual developers as well as commercial vendors to find novel ways of analyzing Java applications as they run even remotely across a network.

In this article I provide a quick-start guide to developing with the new APIs, with my own novel use of the JPDA as an example. Hopefully, this pragmatic approach will help you build your first debugger application quickly, making the prospect of wading through the comprehensive JPDA documentation less daunting.

The Example

Object systems are not static; however, judging from the limited reverse-engineering capabilities offered by leading UML CASE tools, you'd think they were. In most cases you're limited to specifying a set of Java ".class" files. What you get in return is just one of the many diagrams offered by UML, a class diagram.

What if you could reverse engineer a Java application as it runs? You could capture the important dynamic behavior of your applications and complete the UML picture with both sequence and state diagrams.

I propose a new kind of tool that bridges the gap between reverse engineering and application profiling, a tool that presents the runtime behavior of your application, as shown in Figure 1.

Note that the left-hand column of the sequence diagram view in Figure 1 shows the thread on which each object interaction has occurred. Note also that the dependencies view shows dynamic dependencies, for example, where an object instantiates and uses another object within a method, rather than the static dependencies that traditional reverse-engineering tools deduce from member variables.

My first attempt at developing a tool such as this was based on using the original command-line Java Debugger, jdb. The idea was to use the input and output streams to drive debugging instructions through the command-line interface, and to parse the resulting text output to extract information on classes, threads, and method invocations. Looking back, I wouldn't have taken this approach; however, after several frustrating attempts, along came the JPDA as a breath of fresh air.

The JPDA Distribution

To begin this project we must first locate the JPDA distribution and understand how its components fit together. The distribution comes bundled with the Java 2 SDK v1.3, and is available to v1.2.2 developers via a separate download at http://java.sun.com/products/jpda. I'll first describe the separate download as the lowest common denominator.

By unpacking the distribution JAR file, several directories are created:

- ***\bin directory:*** Contains dynamic link libraries (for the Win32 distribution) called dt_shmem.dll, dt_socket.dll, and jdwp.dll. These DLLs provide the native communications support for JPDA and must be included in the environment PATH. Equivalent shared libraries are provided with the Solaris distribution.
- ***\lib directory:*** Contains the JPDA Java Archive, jpda.jar, which must be included in the environment CLASSPATH.
- ***\examples directory:*** Contains, among other things, source code for a version of the original Java Debugger, jdb, that's now based on JPDA. This class, TTY.java, is contained in the examples.jar file.
- ***\bin directory:*** Contains the dt_shmem.dll, dt_socket.dll, and jdwp.dll files. It's for the distribution that comes bundled with the Java 2 SDK v1.3, the SDK installation. There's no separate jpda.jar file because the JPDA Java classes are now included in the tools.jar file within the \lib directory, and the examples.jar file is now located in the \demo\jpda directory.

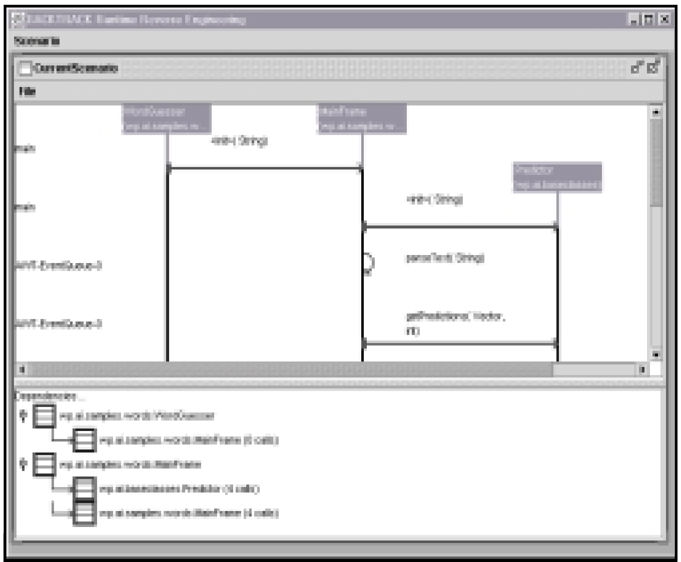


FIGURE 1 Runtime reverse engineering tool

"JDWP provides the communication protocol through which the front and back ends exchange messages"

Three parts constitute the JPDA architecture: the Java Virtual Machine Debug Interface (JVMDI), the Java Debug Wire Protocol (JDWP), and the Java Debug Interface (JDI). Figure 2 shows how these aspects of the JPDA fit together.

A JVM that's supporting the JVMDI, such as the Java 2 SDK, provides the debugger back end. This back end interrogates and controls the VM, and communicates through shared memory or over the network via sockets with the debugger front end.

JDWP provides the communication protocol through which the front and back ends exchange messages, irrespective of the transport mechanism, thus opening up the possibility of remote debugging across a network.

The JDI is a 100% Java interface implemented by the front end that defines information and requests at user code level. Although developers can make direct use of the JVMDI and JDWP, the accompanying documentation recommends this JDI layer for all debugger development.

A Step-by-Step Example

Now I'll provide a step-by-step guide to using the JPDA to develop a front-end application. Code examples are based on real application code that runs; however, the examples have been simplified for clarity. In some cases the exception handling and prior declaration of variables have been removed. In all cases, the full package names for the classes have not been used, so you should be aware that all the new classes introduced here belong in one of the following packages:

```
com.sun.jdi
com.sun.jdi.connect
com.sun.jdi.event
com.sun.jdi.request
```

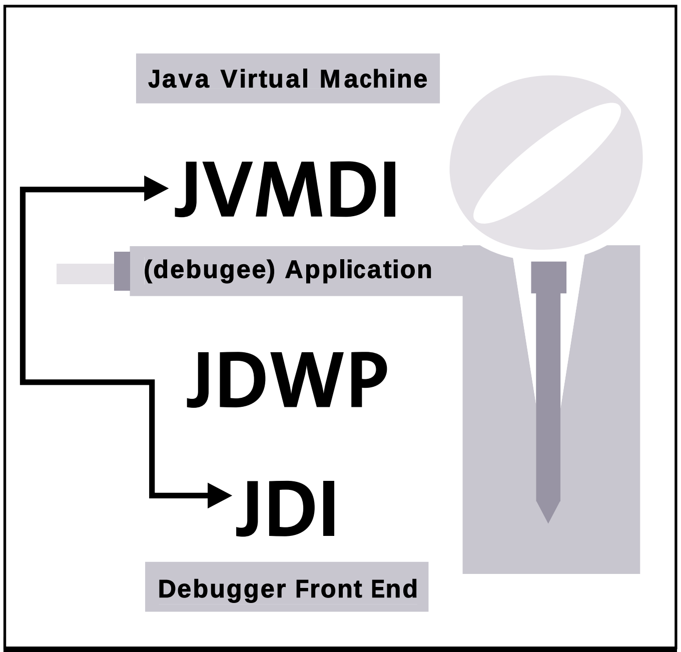


FIGURE 2 JPDA architecture overview

Running the Target Application

A target application is made available for debugging by invoking it with a special combination of command-line options. This is all you need to do to give the application debugee status, with no changes required within the application code. A typical command to invoke an application to be debugged within the Sun classic VM looks like this:

```
java -Djava.compiler=NONE -Xdebug -Xnoagent
-Xrunjdwp:transport=dt_socket,server=y,address=8000
MyApplication
```

The following describes the options in the command:

- ***Djava.compiler=NONE:*** Disables the JIT compiler
- ***Xdebug:*** Enables debugging
- ***Xnoagent:*** Disables the old debugging agent that predates the JPDA
- ***Xrunjdwp:transport=dt_socket,server=y,address=8000:*** Specifies sockets as the connection mechanism, the type of application to be a server and listen for a front-end debugger application to attach, and the transport address on which to listen

There are many other combinations of options described in the JPDA documentation, too many to describe here in detail. For example, it's possible to run a target application that attaches to an already running debugger front end (-Xrunjdwp:server=n), or one that communicates with the debugger front end through shared memory (-Xrunjdwp:transport=dt_shmem) rather than sockets.

Attaching the Front-End Debugger to the Target Application

The simplest way to demonstrate the connection of a front-end debugger to a target application is by using the version of the Java Debugger, jdb, that comes with the JPDA distribution.

The command:

```
jdb -attach targetHostName:8000
(shorthand for jdb -connect
com.sun.jdi.SocketAttach:host=target
HostName,port=8000)
```

attaches the front-end debugger to the (remote) target application listening on machine targetHostname, port 8000.

I recommend using jdb to test the basic JPDA operation prior to developing your own front-end debugger applications. However, for your own debugger applications, and for my example, we need to know how to write the equivalent Java code to establish a connection to a target application.

To use the JDI APIs to establish a connection to a target application, it's necessary to understand the notion of a connector. A JPDA connector is similar to a JDBC driver that separates the implementation details from the API specification. The connectors provided with the JPDA distribution are:

- ***Sun Command Line Launching and Raw Command Line Launching:*** Both connectors launch an application in a new JVM and attach to it.
- ***Socket Attaching:*** Attaches to a currently running, possibly remote, target VM through the socket transport.
- ***Shared Memory Attaching (Win32 only):*** Attaches to a currently running target VM through the shared-memory transport.
- ***Socket Attaching and the Shared Memory Attaching:*** Each connector has a listening version that allows the front-end debugger to be invoked first, with the target application then attaching to it. Although the API for each connector is identical to that for every other connector, the individual capabilities of each one are accessed and controlled through a series of name-value properties.

My example uses the Socket Attaching Connector that attaches to a target application initiated using the typical invocation command shown earlier:

```
java -Djava.compiler=NONE -Xdebug -Xnoagent
-Xrunjdwp:transport=dt_socket,server=y,address=8000
MyApplication
```

Now onto the Java code. First I look for the required connector by name as shown in Listing 1.

Next I obtain the arguments or name-value pairs for this connector and set their values according to my needs (see Listing 2).

Finally, I create an instance representing the target virtual machine and attach to it (see Listing 3).

Implementing the Front End Using the JDI APIs

For my Runtime Reverse Engineering Tool I'm interested in trapping each method invocation as it occurs in the target application. This allows me to build up the UML sequence diagram and the dependencies view as shown in Figure 1.

The JDI API provides an EventRequestManager class, the singleton instance of which may be obtained from the VM. This may be configured to trap events such as method invocations as they happen. For each set of method invocations that we wish to intercept, a MethodEntryRequest must be added to the EventRequestManager.

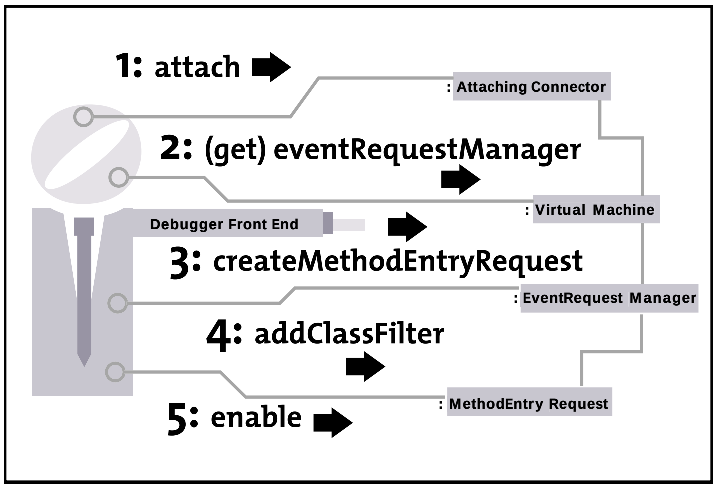


FIGURE 3 Collaboration diagram for attaching and adding a MethodEntryRequest

The Method-EntryRequest may be restricted by adding a class filter, which is important for my example because I want to display interactions between application objects (e.g., mypkg.*), not between Java library classes (java.*). Listing 4 demonstrates how to obtain the singleton EventRequestManager and how to add a MethodEntry-Request to it.

For clarity Figure 3 presents a collaboration diagram showing the steps for attaching to the target VM in the first place, and then adding a MethodEntryRequest.

Each time the EventRequestManager traps an event corresponding to my MethodEntryRequest, it places an event on the EventQueue for my attention. Thus the main body of any front-end debugger application will almost certainly consist of an endless loop in which events are popped from the EventQueue, as shown in Listing 5.

The "// process this event" comment indicates where the code should go to take some action for the event. For my example the action is to call a method to add a new object interaction to the UML sequence diagram shown in Figure 1.

To add a new object interaction I need to know three things: the method that's been invoked, the callee (receiver) class, and the caller (sender) class. The steps involved in getting these three pieces of information are:

1. Get the method from the event object.
2. Get the thread on which the event occurred.
3. Get the list of stack frames for this thread.
4. Look at the top stack frame for the callee class.
5. Look at the next stack frame for the caller class.

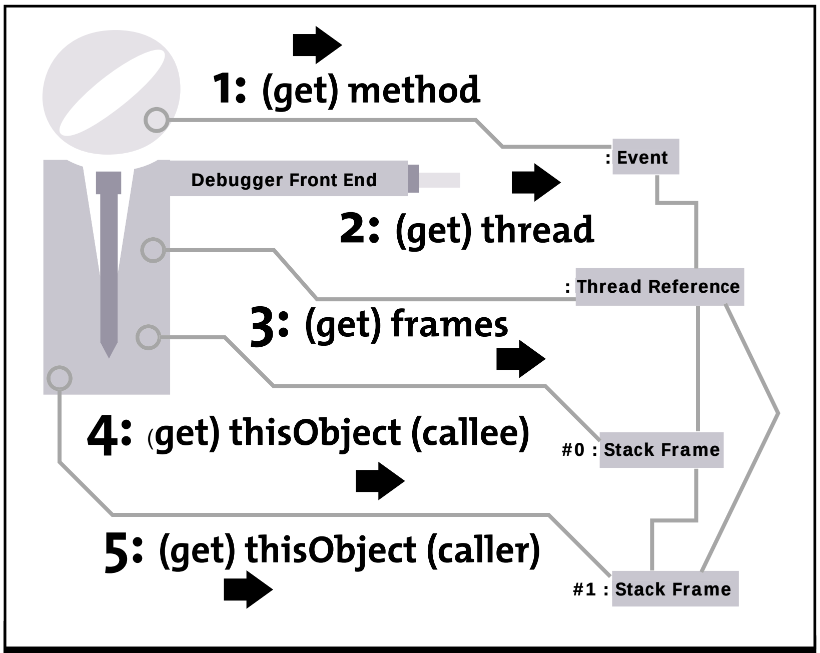


FIGURE 4 Processing an event

Listing 6 shows how to implement these five steps, which are also presented as a collaboration diagram in Figure 4.

For my application it gets a little more complicated. Since I'm interested only in interactions between application objects, not Java library objects, the caller class isn't necessarily the one on the second stack frame. I therefore work down the list of stack frames until I find one with a caller class that's not in one of the java.* packages. This identifies the caller application object that interacts with the callee application object, albeit possibly through several levels of Java library objects.

A Variation Using the Command Line Launching Connector

For many applications you may not want to attach to an already running, possibly remote, application. Instead, you might allow the debugger front end to launch the target application on demand. In this case you'll need to use the Command Line Launching Connector rather than the Socket Attaching Connector.

The name of the connector to search for in the list of available connectors (see Listing 1) is "com.sun.jdi.CommandLineLaunch". The arguments for this connector, which can be set by modifying the code shown in Listing 2, are "options" (options, in addition to the debug options, with which to invoke the JVM), "main" (the main class and command-line arguments for the target application), and "suspend" (true if the target VM is to be suspended immediately before the main class is loaded).

A version of Listing 2, modified for the Command Line Launching Connector, is shown in Listing 7.

You'll notice that in this example I obtain the InputStream and ErrorStream from the VM process and pass each to a display-RemoteOutput() method. The output from the invoked application must go somewhere, and for my application "somewhere" means the standard output of the debugger front end. A more elegant solution would be to provide both an "output" and "error" window as part of the debugger front-end GUI, and channel the target application's output streams to these windows as appropriate.

Conclusion

As stated in the beginning of this article, the inclusion of the JPDA in the Java 2 SDK should facilitate the development of a new breed of debugging and profiling tools.

Using the JPDA I succeeded in developing a prototype of my proposed Runtime Reverse Engineering Tool, which allows any running – local or remote – application to be visualized as a UML sequence diagram and a (dynamic) dependency diagram as shown in Figure 1. Without this addition it wouldn't have been so easy, if at all possible, to realize my own debugger front end.

The listings in this article have been simplified for publication, but they're based on code that actually runs as part of my prototype Runtime Reverse Engineering Tool. The complete code has taken a significant amount of time to develop, but I'll consider genuine e-mail requests for access to the full binary and source code for this project.

"The inclusion of the JPDA in the Java 2 SDK should faciliate the development of a new breed of debugging and profiling tools"

AUTHOR BIO

Tony Loton has several years' experience as a consultant and course trainer for Enterprise Java, CORBA, and UML. He presents new technology seminars nationally and internationally and recently set up an independent company, LOTONtech (www.lotontech.com).

tony@lotontech.com

Listing 1

```
Connector connector=null;
List connectors=Bootstrap.virtualMachineManager().allConnectors();
Iterator iter=connectors.iterator();
while (iter.hasNext())
{
  Connector thisConnector=(Connector)iter.next();
  if (thisConnector.name().equals("com.sun.jdi.SocketAttach"))
    connector=thisConnector;
}
```

Listing 2

```
Map arguments=connector.defaultArguments();
for (Iterator itr=arguments.keySet().iterator(); itr.hasNext();)
{
  Connector.Argument argument=(Connector.Argument)
  arguments.get(itr.next());
}
Connector.Argument host=(Connector.Argument) arguments.get("host-
name");
Connector.Argument port=(Connector.Argument) arguments.get("port");
host.setValue("targetMachine");
port.setValue("8000");
```

Listing 3

```
AttachingConnector attacher=(AttachingConnector) connector;
try { vm=attacher.attach(arguments); }
catch (Exception e) {//error}
```

Listing 4

```
EventRequestManager em=vm.eventRequestManager();
MethodEntryRequest meR=em.createMethodEntryRequest();
meR.addClassFilter("mypckg.*");
meR.enable();
```

Listing 5

```
EventQueue eventQ=vm.eventQueue();
while (running)
{
  EventSet eventSet=null;

  try { eventSet=eventQ.remove(); }
  catch (Exception e) { // handle the error }

  EventIterator eventIterator=eventSet.eventIterator();
  while (eventIterator.hasNext())
  {
    Event event=eventIterator.nextEvent();

    if (event instanceof MethodEntryEvent) { // process this event
}
    vm.resume();
  }
}
```

Listing 6

```
String methodString=event.method().toString();
ThreadReference thread=event.thread();
List stackList=thread.frames();
int level=0;
for (Iterator it=stackList.iterator(); it.hasNext();)
{
  StackFrame stackFrame=(StackFrame) it.next();
  ObjectReference thisObject=stackFrame.thisObject();
  if (level==0)
    { calleeClass=thisObject.referenceType().toString(); }
  else if (level==1)
    { callerClass=thisObject.referenceType().toString(); }
}
```

Listing 7

```
Connector.Argument opts=(Connector.Argument)
arguments.get("options");
Connector.Argument main=(Connector.Argument) arguments.get("main");
Connector.Argument susp=(Connector.Argument) arguments.get("sus-
pend");

main.setValue("MyApplication arg1 arg2");
susp.setValue("false");

LaunchingConnector launcher=(LaunchingConnector) connector;

try
{
  vm=launcher.launch(arguments);
  process=vm.process();
  displayRemoteOutput(process.getErrorStream());
  displayRemoteOutput(process.getInputStream());
}
catch (Exception e) { // error }
```

Download the Code!
www.JavaDevelopersJournal.com

The Java Message Service

The newest addition to the J2EE platform

Part 1 of 3

WRITTEN BY
DAVE CHAPPELL

The Java Message Service (JMS) is an enterprise-capable middleware component based on message-oriented middleware (MOM) fundamentals. Since its introduction as a Java software specification in November 1998, vendor implementations have brought JMS forward as a first class, e-business messaging communications platform suitable for exchanging critical business data over the Internet.

This article is the first in a series of three that explain the application program interfaces (APIs), the message delivery semantics, and the deployment environments that are well suited to JMS applications.

What Is JMS and Where Did It Come From?

JMS defines the first and only standard for asynchronous MOM implementations. The specification defines a common set of APIs and message delivery semantics. A developer can write an application using the JMS APIs and enjoy the freedom of choosing among many vendor implementations (JMS providers) on the market today.

The JMS specification was an industry effort, and JavaSoft worked closely with the messaging vendors to produce it. The intent of the effort was to standardize on a common set of APIs and message delivery semantics across all messaging products. That intent changed over the course of the work to provide a specification for a first-class middleware on equal footing with RPC-like solutions such as RMI, CORBA, and EJB. The result is not a least common–denominator compromise but a powerful list of features. These features are a superset of the capabilities provided by the messaging vendors who participated in the effort. Since then, a new breed of JMS-based products, such as SonicMQ, have been built from the ground up with the goal of delivering high volumes of messages across the Internet in a secure, reliable fashion.

JMS As Part of the J2EE Strategy

Thus far JMS has been an optional part of the J2EE platform. Until recently both EJB and JMS servers were not explicitly related or tied to one another. With the introduction of the upcoming 1.3 J2EE specification, JMS will become a required component of the J2EE strategy. The recent renditions of the EJB 2.0 specification introduce the notion of a MessageDrivenBean. The MessageDrivenBean is a special kind of EJB that exists solely for producing and consuming JMS messages in an EJB server environment. More details on this new model will be discussed in an upcoming article.

Standards Enable Best-of-Breed

Application server vendors have been clamoring to meet this new J2EE requirement through building, buying, and partnering in order to round out their J2EE offering and provide a JMS server in the box. The good news is that regardless of what JMS offering is provided in the box of an EJB vendor, the integration points between an EJB server and a JMS provider are fairly well defined. You may choose your own best-of-breed JMS vendor offering for stand-alone messaging applications and still enjoy the convenience of integrating with the EJB server when EJB and messaging need to be used together.

“JMS defines the first and only **standard for asynchronous** **MOM implementation**”

The JMS Specification

The JMS specification defines such things as multiple messaging models in terms of publish and subscribe (pub/sub) and point-to-point queuing. While largely intended to be an asynchronous form of communication between applications, there's also a set of design patterns and helper classes to perform a synchronous and an asynchronous request/reply model on top of pub/sub and point-to-point. There's also a rich and flexible definition of what a message is composed of, and strict rules governing store-and-forward mes-

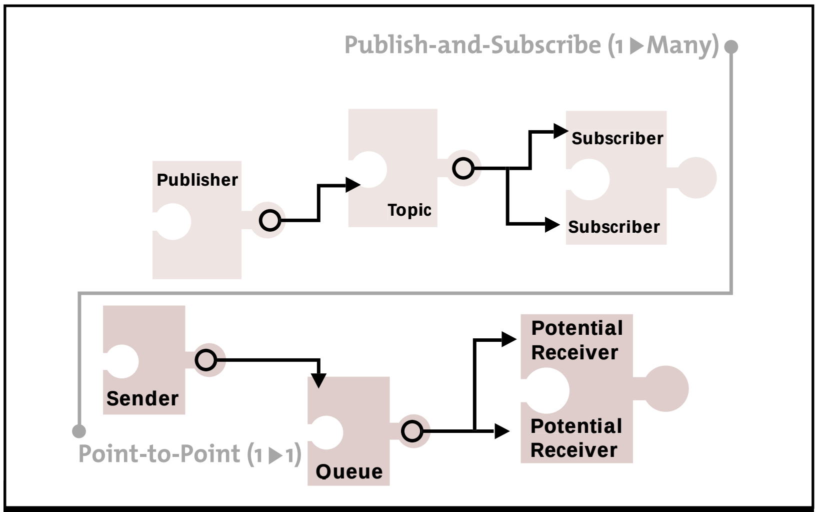


FIGURE 1 JMS messaging domains

saging, guaranteed delivery, message acknowledgments, transactions, and recovery from failures.

JMS applications use the JMS API to communicate with one another using producers and consumers. Senders produce messages and receivers consume them. An application may be both a producer and a consumer at the same time. For instance, a manufacturer may broadcast a request for price quotes to interested suppliers (consumers). At the same time, suppliers may be responding with proposals in the form of XML documents.

Producers and consumers are loosely coupled with each other (they're not directly tied to each other). They're abstractly connected through virtual channels that are administratively defined and accessed programmatically via a JNDI lookup().

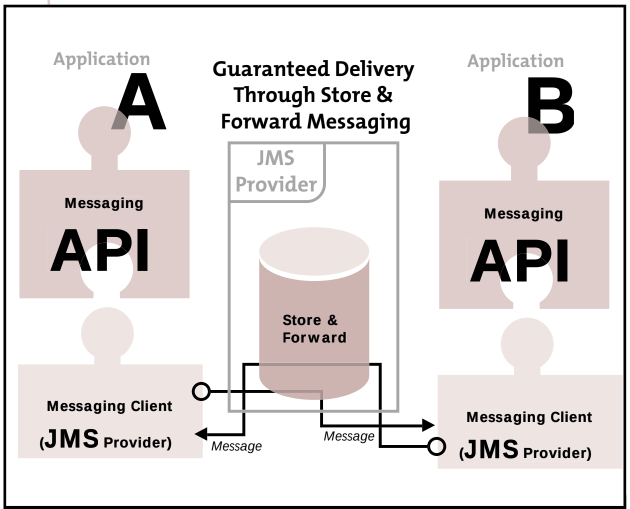


FIGURE 2 Guaranteed once-and-only-once delivery of messages

Messaging Domains

JMS defines two messaging models – *publish-and-subscribe* and *point-to-point* queuing. In JMS terms these are referred to as *messaging domains*. As shown in Figure 1, the pub/sub domain is intended for a one-to-many broadcast of information, while the point-to-point domain is intended for a one-to-one communication between two specific applications.

In the pub/sub model, multiple consumers may register an interest with, or subscribe to, a virtual channel called a *Topic*. A producer uses the publish() method to send a message on that Topic. Each subscriber receives a copy of that message.

In the point-to-point model, only one consumer may receive a message that's sent to a queue. As shown in Figure 1, a point-to-point queue may have multiple consumers listening on a queue for the purposes of load-balancing or "hot backup"; still only one receiver may consume each individual message.

Quality of Service (QoS)

The message delivery semantics cover a range of once-and-only-once to at-most-once delivery. In the once-and-only-once delivery mode, a message is guaranteed by the JMS provider to always arrive at the intended destination no matter what, and it's sent only once. Even in the pub/sub model in which multiple receivers may consume a copy of a broadcasted message, the rules still apply within the relative view of each consumer. Once-and-only-once delivery guarantee is accomplished by the JMS provider through the combination of a store-and-forward mechanism and a rigidly defined set of message acknowledgments (see Figure 2).

At-most-once delivery is a less stringent QoS setting on a message – the JMS provider is allowed to occasionally lose a message. A classic example I like to use is a stock feed application. If the broadcast of a particular ticker symbol doesn't reach its intended destination, another one will be along shortly.

Whether it's once-and-only-once or at-most-once, the key word is *once*. Regardless of the guaranteed-ness of the delivery mode, the JMS provider is responsible for ensuring that the messages are delivered in the exact order in which they are sent.

What's in a Message?

A message is composed of three basic parts: headers, properties, and the message payload. The headers are used by the JMS provider and the application developer to provide information about such things as the destination, a reply-to destination, the message type, and the message expiration time.

The property section contains a set of application-defined name/value pairs that are accessible via an extensive assortment of setters and getters on the Message object.

The message payload can contain any type of application data and is accessible in a number of forms. There are five prescribed message types:

- ***BytesMessage:*** A stream of bytes intended for any arbitrary data, binary or otherwise. Typically useful for data that's opaque to the messaging system (not Java accessible), yet structured in some native format that's known to the sending and receiving application.
- ***ObjectMessage:*** A set of objects accessible by Java programs.
- ***MapMessage:*** A set of name/value pairs in a variety of data types.
- ***StreamMessage:*** Similar to BytesMessage and ObjectMessage, with the added value that a message may be read from and written to using a familiar file stream metaphor.
- ***TextMessage:*** Plain text.
- ***XMLMessage:*** Not part of the JMS standard. However, many JMS vendors now include an XMLMessage type. It allows an application to construct and deconstruct a message using a DOM tree.

JMS and XML — An Ideal Marriage

XML is fast becoming the lingua franca for describing business data for interapplication communication. However, XML on its own doesn't carry with it a reliable way of getting your critical XML documents through the treacherous waters of today's B2B or geographically dispersed intracompany communications environment. Hardware issues can occur, such as network or server machine failures. Applications that participate in a distributed application environment can crash or have scheduled downtimes. Applications need a way of transmitting data between one another reliably. JMS provides the guaranteed delivery semantics to ensure that critical business data gets delivered across a sometimes hostile environment in a once-and-only-once guaranteed fashion.

JMS As the Basis for an E-Business Messaging Environment

The term *e-business messaging* is intended to describe the type of communication infrastructure that's necessary for doing B2B communications over the Internet. Although the term *B2B* is overloaded and overused, the generally accepted – and simplest – definition of B2B is the ability to conduct business electronically between applications that span corporations.

Whether dealing directly with suppliers or communicating through a trading exchange, a manufacturer or a wholesaler may have thousands of suppliers to deal with. With that size of a trading partner community, there may be thousands, even hundreds of thousands, of electronic data exchanges between multiple parties.

There are many challenges to consider when building an e-business messaging infrastructure:

- Guaranteed delivery of data
- Massive scalability
- High availability and reliability
- Security, authentication, and access control
- Internet firewalls
- Standards and connectivity

A JMS communications layer provides a key ingredient for such a communication infrastructure. It provides the guaranteed once-and-only-once delivery and guaranteed ordering of messages in an asynchronous fashion between applications, and a simple yet flexible API to connect applications into the messaging system. It strongly dictates message delivery behavior with regard to message persistence, acknowledgments of message receipt, transactional groups of messages, failure conditions, and recovery rules.

“There are three things you **need to learn:** connect, send, and receive”

However, JMS by itself doesn't provide all the additional elements to meet the described requirements of e-business messaging. It's a specification that's intentionally agnostic with regard to wire transport protocols and deployment architectures. It doesn't dictate a security model beyond a simple username and password at connection time. In recognition of this, many JMS vendors are providing these additional capabilities on their own.

I'm Sold on JMS! What Does This API Look Like?

The JMS API is conceptually simple to understand. There are three things you need to learn: connect, send, and receive. Beyond that are a handful of interfaces, such as ConnectionFactorys beget Connection objects. A connection may have one or more Session objects. A session is responsible for managing producers and consumers (senders and receivers).

A ConnectionFactory may either be a TopicConnectionFactory for pub/sub or a QueueConnectionFactory for point-to-point. Likewise, a connection may either be a TopicConnection or a QueueConnection. Figure 3 illustrates the interfaces.

The publish() method sends a message to a pub/sub topic. The send()

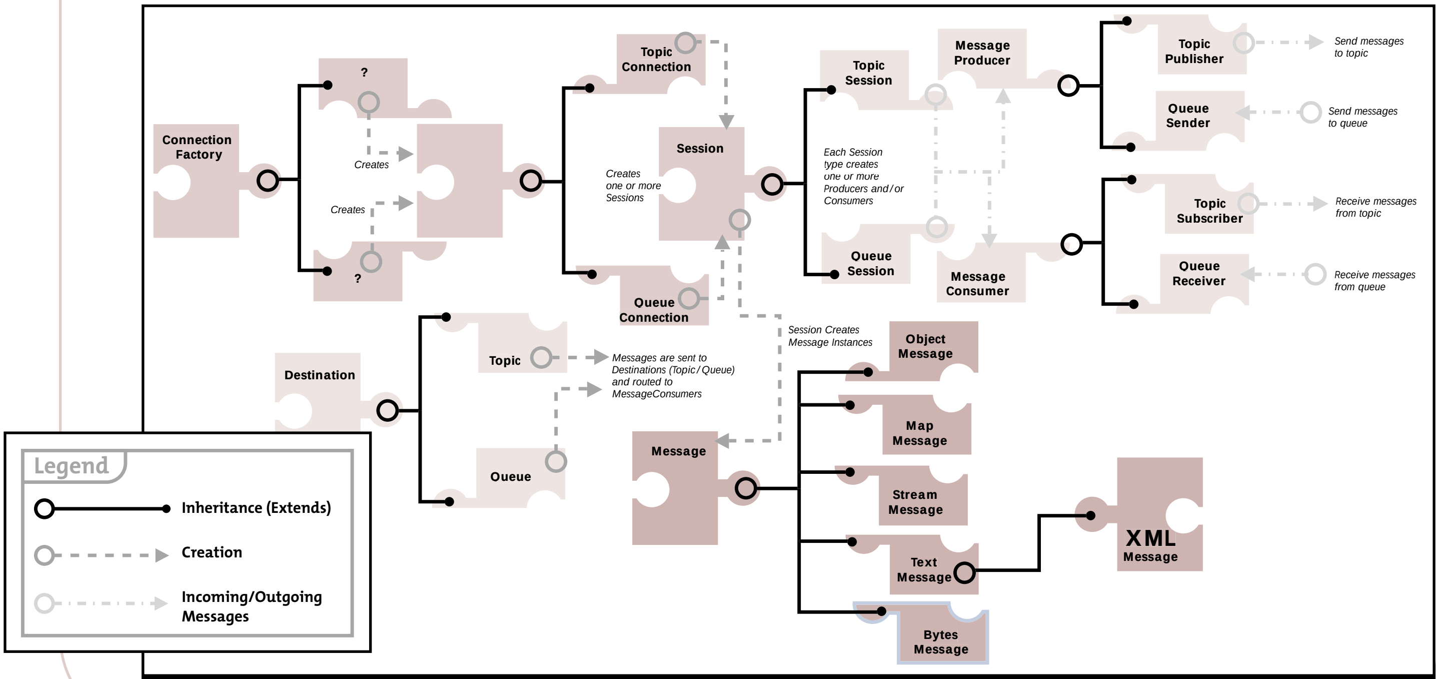


FIGURE 3 Simple interfaces in JMS

method sends a message to a point-to-point queue. From the consumer's perspective, the message may be passively consumed via the onMessage() callback, or a message may be proactively received using the receive(), receive (long timeout), and receiveNoWait() methods.

The code snippet in Listing 1 shows the steps necessary for setting up a connection, session, TopicPublisher, and sending a message. This simple example sends a message to itself.

That's all that's required to set up a sender and a receiver. It's quite simple. The onMessage() handler is responsible for receiving the message, which is invoked automatically by the JMS provider. It looks like this:

```
public void onMessage(
javax.jms.Message message){
TextMessage textMessage =
(TextMessage) message;
System.out.println(textMessage.get
Text());
}
```

The point-to-point queue version of this is very similar. The structure of the code remains the same, and the names of some of the objects are slightly different. That's one of the nice things about JMS. Even though there are two different messaging models, the naming conventions and the API usage model are extremely similar. It's easy to switch from one to the other if you change your mind.

Now that you see how easy it is to use the JMS specification, you're well on your way to becoming an expert in the J2EE platform's hottest new addition to its family. Get the message!

chappell@progress.com

Author Bio

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

Listing 1

```
Properties env = new Properties();
// ... specify the JNDI properties specific to the JNDI SPI being used
…
jndi = new InitialContext(env);
// obtain a connection factory
factory =
     (TopicConnectionFactory)jndi.lookup("TopicConnectionFactory");
// create a connection
connect = factory.createTopicConnection (username, password);

// create a session for publishing, and one for subscriptions
pubSession =


connect.createTopicSession(false,Session.AUTO_ACKNOWLEDGE);
subSession =
     connect.createTopicSession(false,Session.AUTO_ACKNOWLEDGE);

myTopic = (Topic)jndi.lookup("My First Topic");

// create the publisher and the subscriber
publisher = pubSession.createPublisher(myTopic);
subscriber = subSession.createSubscriber(myTopic);

// associate the onMessage() handler with this subscriber
subscriber.setMessageListener(this);

// start the flow of incoming messages
connect.start();

TextMessage textMsg = pubSession.createTextMessage();
textMsg.setText("My first JMS message!);

publisher.publish(
     textMsg,
     javax.jms.DeliveryMode.PERSISTENT,            // delivery mode
     javax.jms.Message.DEFAULT_PRIORITY,     // message priority
     1800000);
// Time-to-live (30 minutes)
```

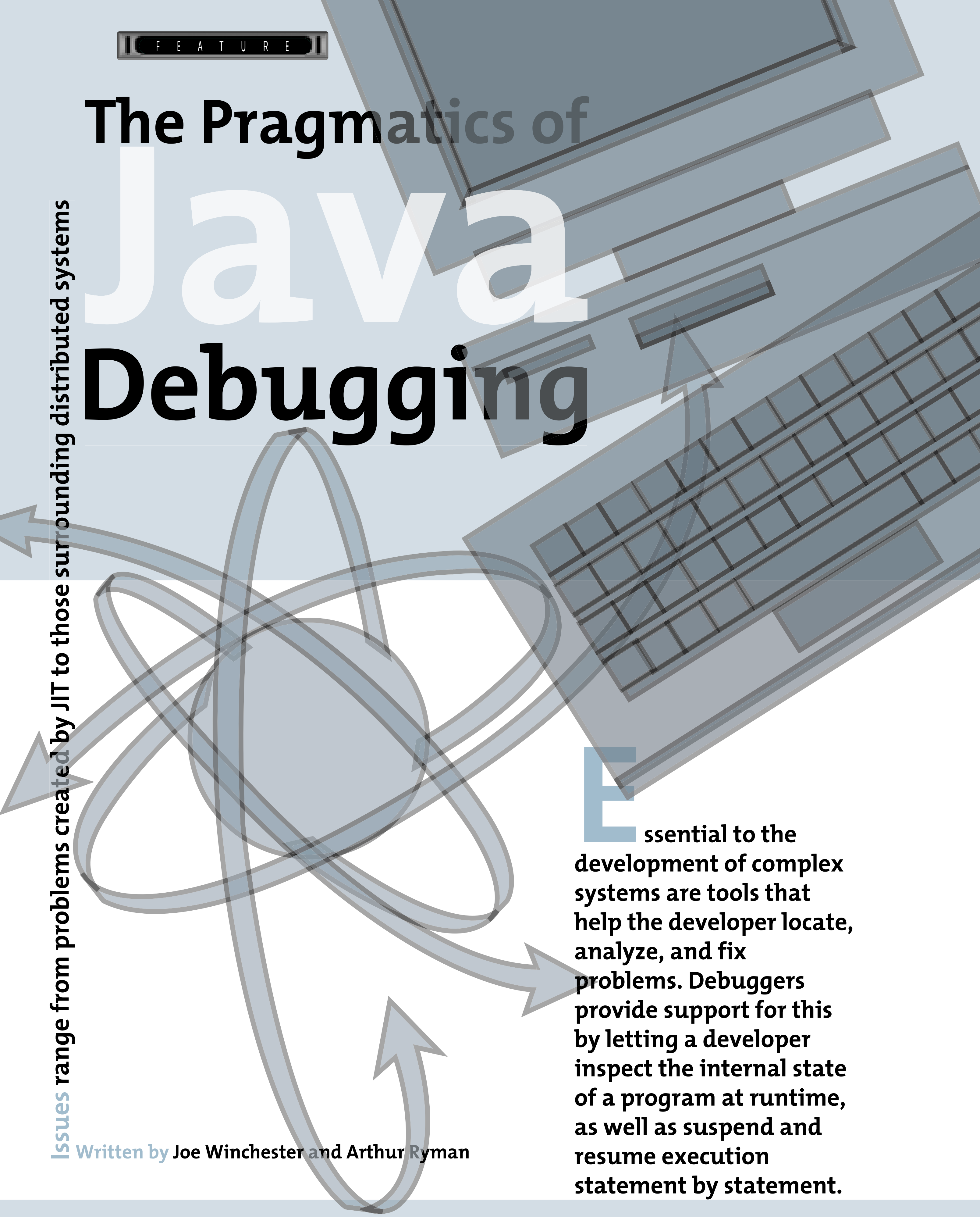
Download the Code!
www.JavaDevelopersJournal.com

FEATURE

The Pragmatics of Java Debugging

Issues range from problems created by JIT to those surrounding distributed systems

Written by Joe Winchester and Arthur Ryman



Essential to the development of complex systems are tools that help the developer locate, analyze, and fix problems. Debuggers provide support for this by letting a developer inspect the internal state of a program at runtime, as well as suspend and resume execution statement by statement.

The originators of the Java programming language defined a debugging architecture, but since its conception Java has advanced into new areas of deployment topologies and optimization technologies that present a further set of problems. This article covers some of the background behind these issues as well as the activity in the Java community to provide solutions. Examples of debugging solutions are drawn from the IBM VisualAge for Java integrated development environment (IDE), although the issues are applicable to other environments as well.

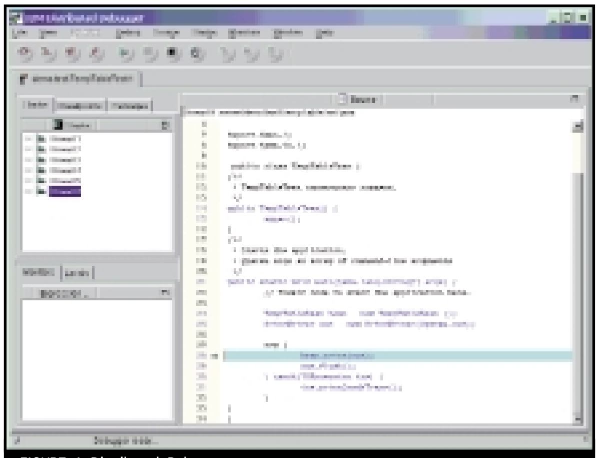


FIGURE 1 Distributed Debugger

Debugger Basics

Java source code is written in .java files. The source is compiled into a .class file – bytecodes interpreted by a Java Virtual Machine (JVM). The JVM dynamically loads class files by searching the program's classpath. The purpose of the JVM is to execute the bytecodes and let the user navigate and execute their application. It does this by translating the bytecodes into a set of lower-level machine instructions that are specific for the platform on which the JVM is running. This level of indirection is what allows different JVMs to be written for different hardware or operating system APIs, while letting the bytecode class files meet the promise of "write once – run anywhere."

Dynamic interpretation of the bytecodes into machine instructions gives Java the advantages of portability and operating system neutrality, but it comes at the price of reduced performance.

When a program gets compiled into bytecodes, the compiler focuses on making the bytecodes executable by the target JVM. However, if the program is going to be debugged, the developer needs to be able to trace the bytecodes back to their source. Java is a very dynamic language, so the physical layout of a class is not determined at compile-time but is actually generated by the JVM when it loads the class. Therefore all class, method, and field names are preserved in the class file, which also includes the name of the source file and a line number table that maps ranges of bytecodes to their corresponding line in the source file.

The JVM provides an API for setting breakpoints and manipulating the stack frame. This is the API that the Java Development Kit (JDK) debugger uses and it also surfaced in the package sun.tools.debug. The JDK debugger is a command-line debugger that, when executed (with the jdb command), lets the developer perform basic tasks such as inserting breakpoints on statement lines, catching thrown exceptions when they're raised, and printing and advancing the stack trace. The jdb program is limited in its functionality due to its command line interface, but there are a number of good debuggers that have graphical interfaces to let the developer view and monitor the program while debugging it. The IBM Distributed Debugger and Symantec Visual Cafe are two such debuggers, as is the integrated debugger that comes with the VisualAge for Java IDE.

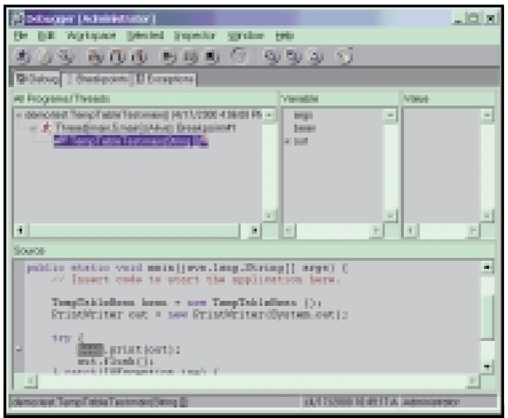


FIGURE 2 Integrated debugger

To reduce the size of the class file, the javac compiler normally removes symbolic information not required at runtime, such as method argument and local variable names. The debug API returns generated names such as arg_1, instead of the original argument name given by the developer. If you need to debug a program with the original names the javac option –g can be used.

Writing a Program to Aid Debugging

When developers write programs in a high-level language, they concentrate on making maintainable, error-free, and extensible source code. Another often overlooked goal is to write code that's debuggable. Certain programming practices can help to debug a program such as using temporary variables to store the result of intermediate method calls. Often a developer uses a temporary variable to store the result of a call that's used more than once in the method. If the method result is used directly on the next line, the statements are typically chained together. If a temporary variable is used instead, at debug time you can see the intermediate result. The following code shows an example of chaining method calls together:

```
anObject.setSomething(anotherObject.getSomething() +
yetAnotherObject.getSomethingElse() );
```

In the debugger, you can't see the result of getSomething()or getSomethingElse(). The code could be rewritten as:

```
Object something = anotherObject.getSomething();
Object somethingElse = yetAnotherObject.getSomethingElse();
anObject.setSomething(something + somethingElse);
```

When you single-step through this code, you can see the result of each method result in turn. The debugger API lets you insert breakpoints at the start of each statement, so writing code that doesn't combine many expressions into a single statement also provides more target points to insert a potential breakpoint.

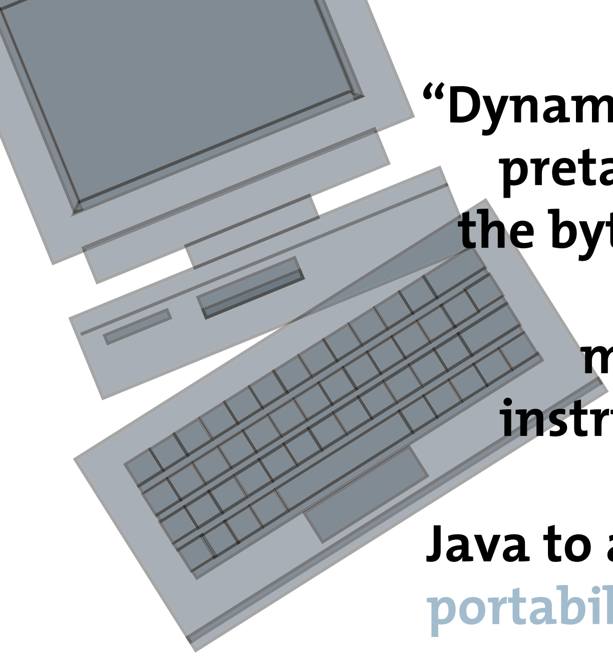
Debugging a Program Running Inside a JVM

The typical life cycle for a Java program is the developer writes some code, compiles it, and then tests it by executing the compiled class file in a JVM. To do this the developer must first compile the program to allow debugging, and then use a debugger that calls the Java debug API to control the JVM program execution.

When the JVM raises an exception or hits a breakpoint, the debugger visually shows a stack trace of the program so developers can inspect the contents of the program variables. The Distributed Debugger also shows the source code if it's available for the debugger to use (see Figure 1).

When the debugger has halted program execution and you've inspected the program state, you can use the step functions to continue execution statement by statement, or just resume the program until it reaches the next uncaught exception or breakpoint. This lets you understand the dynamics of the program execution and locate errors. Once you've located a problem you might see the errant code and want to change some of the source and test the change. To do so typically requires that you exit the program, recompile the .java source file, and rerun the program with the new source. In an environment with fast turnaround time where you're trying to quickly test and fix code, this stop-change-compile-start cycle can be time-consuming.

In addition, if the program is being debugged in a server environment, replacing the server class files might be difficult. If the server isn't designed for development it might place locks on JAR files, and there



“Dynamic interpretation of the bytecodes into machine instructions allows Java to achieve portability and operating system neutrality”

may be no way to reload a modified class, requiring you to restart the server, then re-create the error condition. Servers may take a long time to restart and, if the server is a production server, downtime may be difficult to obtain. Some servlet engines, such as WebSphere, can reload a servlet, but if the servlet uses other classes such as JavaBeans or EJBs, and you modify them, you may still need to restart the server.

Debugging a Program Running Inside an IDE

Developers writing code they wish to test can compile their source into class files and debug it in a JVM as described above. However, the turnaround time of recompilation, debug, analyze, fix source, and compilation can hinder overall development productivity. VisualAge for Java is a development environment that lets developers write and debug code without any explicit recompilation or program halting. It has a virtual machine that's tightly integrated with the development environment and that allows recompilation into an already running program, as well as incremental compiling so that source and bytecodes are always kept in step. This means that when a breakpoint is reached, the developer can just modify the source and click *save*. The incremental compiler reports any problems, and if there are none the program resumes execution at the top of the new method.

Developers can also execute an ad hoc piece of Java code or modify program variables from within a running program. When the program is suspended in the debugger, the developer can select any method in the call stack and tell the debugger to drop to this frame. This is rather like a rewind button and is useful if the debugger gets invoked by an exception and the developer needs to go back a few method calls and retrace some steps to see the root cause of a problem. NullPointerException is a good example of where the exception is usually thrown too late; the problem isn't the method call on a null variable; it's the earlier method call that was supposed to set the variable to a non-null value that needs debugging. Figure 2 shows the integrated debugger that comes with VisualAge for Java.

The VisualAge for Java debugger works on a Java source running within its own virtual machine. The virtual machine is part of the development environment, which means that it can't be easily replaced. To debug code executing in a different JVM, you can use the Distributed Debugger, but this sacrifices the ability to execute ad hoc Java code, rewind program execution, change variable contents, and modify source inside a running program.

VisualAge for Java also supports this fast code-debug-fix cycle for the WebSphere environment. A stripped-down version of the WebSphere environment, also known as the WebSphere Test Environment, is shipped as part of VisualAge for Java. The WebSphere Test Environment executes as part of VisualAge for Java's internal virtual machine, which means you can test and debug server components such as servlets or EJBs using the techniques described previously. When you've completed the first pass of testing your code, you can export and deploy it in a true WebSphere environment where you can further debug it with the Distributed Debugger. In addition to hosting the WebSphere Test Environment (which emulates the true WebSphere environment inside the IDE), you can also load other environments such as Jakarta Tomcat and New Atlanta ServletExec into VisualAge for Java, which lets you do integrated development and debugging inside the IDE before deploying the server code into a production environment.

Java Debug Topology

The Distributed Debugger lets you debug a Java program that's been compiled for debugging. The remote debugger runs on a client and can debug Java programs that are either executing on the same client (local) or on another machine (remote), such as a server. These are the two most common scenarios for debugging: the Java program is a traditional client program such as a two-tier fat client application, or the Java program is a component, such as a servlet or an EJB, executing as part of a remote application server.

To allow the debugger to work with both local and remote programs, it's split into two portions: the debugger user interface that the developer uses to view and control the program being debugged, and the debug engine that the interface talks to and in turn debugs the JVM. This separation of interface from engine, which is part of the VisualAge for Java Distributed Debugger, allows multiple engines to be controlled from the same interface. This means that developers can seamlessly debug Java on their client through remote calls on a server such as an IBM OS/390 or an AS/400, or a Microsoft Windows NT server.

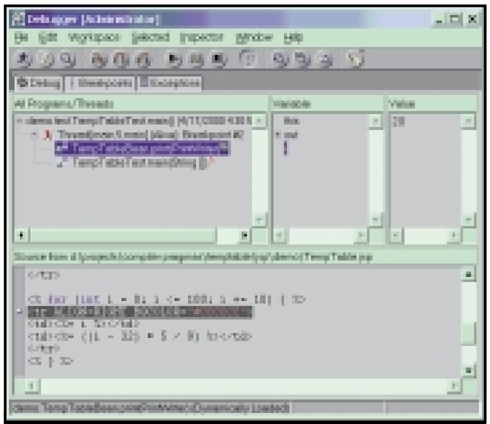


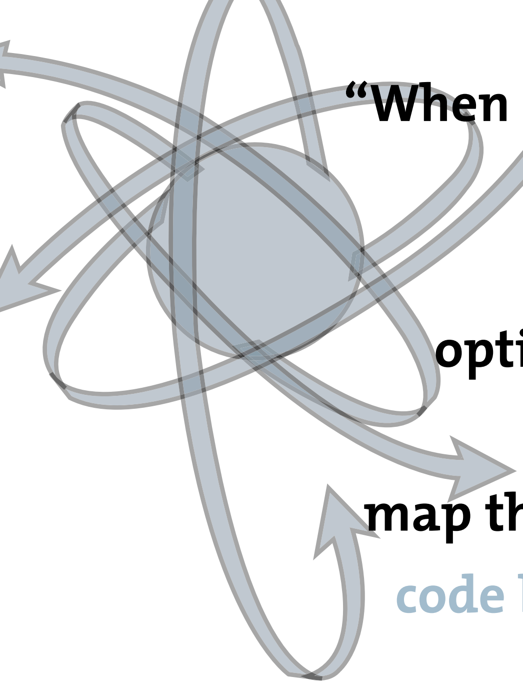
FIGURE 3 The breakpoint in the original source

Cross-Language Debugging

Although Java promises to be the panacea for all programming problems, a complex application will have code that's written in other languages. This code could be a dynamic link library (DLL) written in C, or some JavaScript in an HTML page. The Distributed Debugger uses a standard interface between the user interface and the debug engines that allows other languages to be debugged from the same user interface. The complete list of supported languages varies by platform, but the list includes Java, C, C++, Fortran, Cobol, PL/1, and RPG. Having a single user interface that can debug multiple languages on multiple servers lets the developer trace through an execution scenario and narrow in on the errant code or condition, irrespective of which part of the system or server it's on.

Translated Java Programs

When the Java language was first created, Java programs were running in only a few environments. Since then the use of Java has spread and Java programs can be generated from other languages such as JavaServer Pages and SQLJ. JSPs are used on the server and allow Web servers to generate dynamic HTML page content and send it to the client, while SQLJ allows SQL statements to be embedded in Java programs, and used wherever the database access is performed. Both JSP



"When debugging code that's been JIT optimized, the JVM must map the machine code back to the bytecode"

and SQLJ work by translating the developer's code into a Java program that's then compiled into executable bytecodes. For a JSP this is done by creating a servlet that contains that JSP's Java code. This extra step means that the program actually being debugged is not directly associated with the source the developer wrote.

A number of vendors, such as IBM and Oracle, recognized the need to let the developers debug the source they wrote rather than the translated source. In the C language, a similar problem is caused by the preprocessor, but the source line mapping is preserved through the use of the #line pragma. However, the Java Language Specification did not include a preprocessor or the equivalent of the #line pragma, so each vendor had to implement a proprietary way to preserve line numbers, typically by inserting comments into the translated Java source. Java Specification Request (JSR) 45 defines a standard line-mapping table to preserve the correspondence from the original source (e.g., JSP or SQLJ) to the translated Java source, which could then be read by the javac compiler or another program to postprocess the line number information stored in the class file.

After a class file has been modified to reflect the original source file name and line number information, standard debuggers can be used to debug the untranslated source. The difference can be seen in Figures 3 and 4. Figure 3 shows the VisualAge for Java debugger with the breakpoint in the JSP source statement, whereas Figure 4 shows the breakpoint in the Java servlet source that the JSP was translated into.

Server Dynamics

A complex application can often span several remote environments in which a servlet in one server can call an EJB in a container on another server, that in turn can execute an RMI call to yet another server. It's this kind of topology that the Distributed Debugger solves with its Object Level Trace (OLT), as it can start a debug engine on each server and let the interface seamlessly debug across the different boundaries. Server environments that the OLT supports include the IBM Component Broker and the WebSphere Application Server.

The Distributed Debugger can be opened from within the OLT, which lets you examine and profile program flow, as well as perform the traditional debugging tasks of setting breakpoints, inspecting program variables, and controlling program flow. The OLT is useful for viewing the dynamics of a distributed application, because it includes a trace server that receives trace messages from each process involved in the distributed application. The individual trace messages are assembled so you can follow the sequence of events from process to process.

Another problem that exists with debugging server programs is that when a problem occurs in the server, it's often difficult to re-create the same problem in another environment. To do so requires staging environments that must mimic the server environment in terms of hardware resources as well as runtime conditions such as server load. The reality is that some problems that occur in a production environment can never be re-created in a development environment. The Distributed Debugger can be configured so that a problem on a server can actually call back and invoke a client to let the developer debug the server environment from their console. The ability to have a fully functional debugger opened on a server program that's thrown an exception gives you a much greater view of the problem than just a textual stack trace.

Just-In-Time Optimization

Because bytecode interpretation isn't as fast as native machine code execution, a number of optimization technologies have sprung up. One of these is Just-In-Time or JIT compilation. JIT compiles each method to machine code the first time it's called, and caches the result so it doesn't have to recompile the next time. The default for the JDK JVM is to enable JIT, although you can use the -nojit option to disable it. JIT compilation further obscures the relationship between the Java source code and the machine instructions being executed. When debugging code that's been JIT optimized, the JVM must map the machine code back to the bytecode so the debugger can display the corresponding source.

Many commercial JVMs aren't able to cope with this and disable JIT compilation when they execute a program in debug mode. The IBM and Sun JDKs both disable JIT in debug mode, but the VisualAge for Java IDE does not. Since interpreted bytecodes run 10–20 times slower than machine code, debugging can become a tedious process, especially if you're debugging a complex program such as a Web application server. This is another factor that makes debugging in the WebSphere Test Environment more productive because the VisualAge for Java IDE's JVM always performs JIT optimization.

Disabling JIT can also lead to problems, because it means you're not actually debugging the same program that you'd otherwise be executing. Subtle bugs such as those caused by race conditions or nonsynchronized object lock conflicts might not appear in the slower running debugged program.

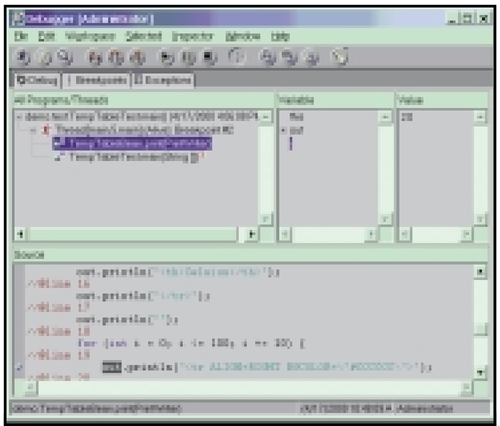


FIGURE 4 The breakpoint in the Java servlet source

This is rather like the quantum physics problem of Schroendinger's cat. In this thought experiment the physicist is presented with a closed box that contains a cat and a cannister of poison gas that's released when a radioactive atom decays. Since the atom exists in a superposition of decayed and nondecayed quantum states, the cat exists in a superposition of dead and alive states. Only by observing the system is it forced into a definite state. Is the cat dead or alive? For the physicist there's no real way of knowing because once the box is opened, the act of observing the cat may be the act that kills it. Just as with quantum mechanics where the observer of the cat affects the system being observed, so too can the debugger affect the system being debugged.

Static Optimization

Another way in which Java programs can be optimized is by static optimization. A static optimizer analyzes a whole Java program and attempts to create a more efficient version. There are two types of static optimization: bytecode and machine code.

Bytecode optimization is where the optimizer applies heuristics to rewrite the bytecode output of the Java compiler to make it more efficient for the JVM to process. Techniques used are:

- Inline method calls to avoid the overhead of locating, executing, and leaving a method
- Changing an object's type, retyping from an interface to an implementation if the interface has only one implementer (thereby allowing direct binding of the method call to the method location)
- Making fields final that the optimizer determines are not modified after they've been initialized

The output of bytecode optimizers is Java bytecodes, so they can still run within any JVM. Examples of bytecode optimizers are Dash O and Jove. Since static optimizers alter the bytecodes, the mapping from bytecodes to source code may also be affected, which could prevent debugging. In general, debugging should be performed on unoptimized code if possible.

Machine code optimization is when the Java bytecodes are translated into machine code at compile time rather than at JVM execution time. This is analogous to how languages such as C and Fortran are compiled, but with Java, the optimizer translates the bytecodes into machine code. Examples of such static optimizers are TowerJ from Tower and the IBM High Performance Java Compiler (HPJ) that comes with VisualAge for Java.

Once HPJ has optimized a program, the JVM is no longer used to execute the bytecodes since HPJ generates a self-contained executable program. This has the performance benefits of a natively compiled language, but it also means the program will run only on a platform for which the HPJ can generate an optimized program. These platforms include Windows, OS/2, AIX, AS/400, and OS/390. Because the HPJ bypasses the JVM, the standard Java debug APIs will no longer work. Instead, the Distributed Debugger, which handles both bytecodes and machine code, must be used to debug HPJ-optimized programs.

Java Platform Debug Architecture

JDK 1.3 includes significant enhancements to the debug API via its Java Platform Debug Architecture (JPDA). The JPDA separates the debug API into three distinct layers:

- ***Java VM Debug Interface (JVMDI):*** Provides an interface to allow a VM to be debugged
- ***Java Debug Wire Protocol (JDWP):*** Provides an API into a JVMDI
- ***Java Debug Interface (JDI):*** Lets a front end sit on top of the JDWP

This three-tier architecture tackles the problem of letting tool developers write debuggers that run portably across platforms, JVM implementations, and JDK versions.

At the highest level, someone writing a graphical debugger can hook into the JDI instead of the API previously described in the sun.tools.debug package. This means that the debugger will automatically work with all JVMs and platforms that Sun supports. If another company writes a different JVM, they'll also write their own JDWP implementation, which means the graphical debugger will be able to debug the other company's JVM. If the tool developer writes a debugger that's not written in Java, rather than program to the JDI layer they can program to the lower layers such as JDWP. The JDPA will work well if vendors who write their own JVMs and others who write their own debuggers all program to this interface. This consistency will mean that the debug experience for a developer working in a complex server environment with heterogeneous JVMs and JDK levels will become much more pleasant than it currently is.

Conclusion

This article has covered some of the issues that surround debugging Java programs, a topic of vital interest to professional developers. These issues range from problems created by JIT and static optimizers and new Java source variants such as JSP and SQLJ, to the issues surrounding distributed programs composed of servlets and EJBs that span multiple processes and servers. Java, however, has demonstrated a remarkable ability to adapt and find solutions to such problems. The Java Community Process through JSR 45 is addressing the need for a standard mechanism to debug translated source. The new JPDA in JDK 1.3 provides a common API that both debugging tool providers and JVM builders can program to.

We've provided some background into the problems of debugging Java programs, as well as some of the currently available solutions and some you should see in the near future. We welcome all feedback.

References

1. *JSR 45:* http://java.sun.com/aboutJava/communityprocess/jsr/jsr_045_debug.html
2. *Java Platform Debug Architecture:* http://java.sun.com/products/jdk/1.3/docs/guide/jpda/architecture.html
3. *IBM VisualAge for Java and IBM Distributed Debugger:* www7.software.ibm.com/vad.nsf
4. *IBM WebSphere:* www-4.ibm.com/software/webservers

Author Bios

Joe Winchester is a software developer at the IBM Research Triangle Park lab in North Carolina working on development tools for WebSphere including VisualAge for Java. He is currently working on composition editors that let programmers construct GUIs for different runtime environments as well as write program logic by connecting JavaBeans together.

Arthur Ryman is a senior technical staff member at the IBM Toronto Lab where he is currently working on the XML and Web Services Development Environment, a new tool suite for developing Java, SQL, and XML-based Web Services that support SOAP, WSDL, and UDDI. Previously he worked on VisualAge for Java, specializing in tools for developing servlets and JSPs.

joewin@us.ibm.com

ryman@ca.ibm.com

An Introduction to Genetic Algorithms in Java

Harnessing the power of evolution's optimization algorithm

Part 2 of 2

WRITTEN BY
MICHAEL LACY

Over the past decade the Internet has evolved from a research project living in the realms of academia and government to a global infrastructure for electronic commerce and digital communication that has sent the stock market on a roller-coaster ride to new highs (and lows).

It's a digital world in which a Darwinian survival of the fittest is taking place right before our eyes as Web sites and Web applications compete for the right to live another day. Whether it's another site offering a better service/product or the latest computer virus, a Web application faces many competitors that threaten its very existence. As in the biological world, only the fittest will survive. And the fittest are the ones capable of adapting to their environment faster than their competitors can.

As developers of these digital organisms, our job is to ensure that we equip our Web applications with the means for survival. Usually this entails an exhaustive list of if–then statements or extensive exception handling, all of which we must conceptualize in order to code. As Web applications become increasingly complex, the task of identifying all possible scenarios to encode becomes daunting and our job as developers becomes reactionary as we code new features, bug fixes, and virus patches to respond to the increased competition for survival in the digital world. What if we could provide software with the ability to adapt to its environment? Why not use the biological model of natural selection to do it?

In the January issue of ***JDJ*** (Vol. 6, issue 1) I introduced a technique born in the AI community that uses concepts from biological natural selection to solve complex and often highly nonlinear optimization problems encountered in computer science – the genetic algorithm. I examined the building blocks of genetic algorithms and why Java is well suited to their implementation. This month I'll discuss the details of a simple genetic algorithm implementation in the hopes that your curiosity will be sparked to pursue further investigation.

“Web applications are increasingly **complex and compete daily for survival with other sites,** **viruses, bugs, and server crashes**”

The nature of software development is evolving at a brisk pace. Web applications are increasingly complex and compete daily for survival with other sites, viruses, bugs, and server crashes. As I said before, only the fittest will survive.

Now that I've grabbed your attention, we need to set some expectations before embarking on our journey. First and foremost, genetic algorithms still exist primarily in academia due to their (sometimes) imposing computational requirements and learning curve. This means their commercial application is still relatively unproven.

The scenario described above in which Web applications have the ability to adapt to their environment is no more than that – a scenario. The realization of this scenario depends on us, the software developers of the Web application world, to discover suitable applications for genetic algorithms.

Second, a few pages covering the implementation details for a genetic algorithm is undeniably insufficient coverage for the development of a thorough understanding of the technique. We're merely skimming the surface of possibilities. With that said, let's jump right in.

Review: Genetic Algorithm

A genetic algorithm is a model for machine learning in which a population of randomly created individuals goes through a simulated process of evolution – a digital survival of the fittest in which each individual represents a point in a problem's solution search space. Reviewing the terminology discussed in Part 1, an individual is referred to as a *chromosome*. A chromosome consists of *genes*, which represent the parameters of the problem being optimized. A collection of chromosomes on

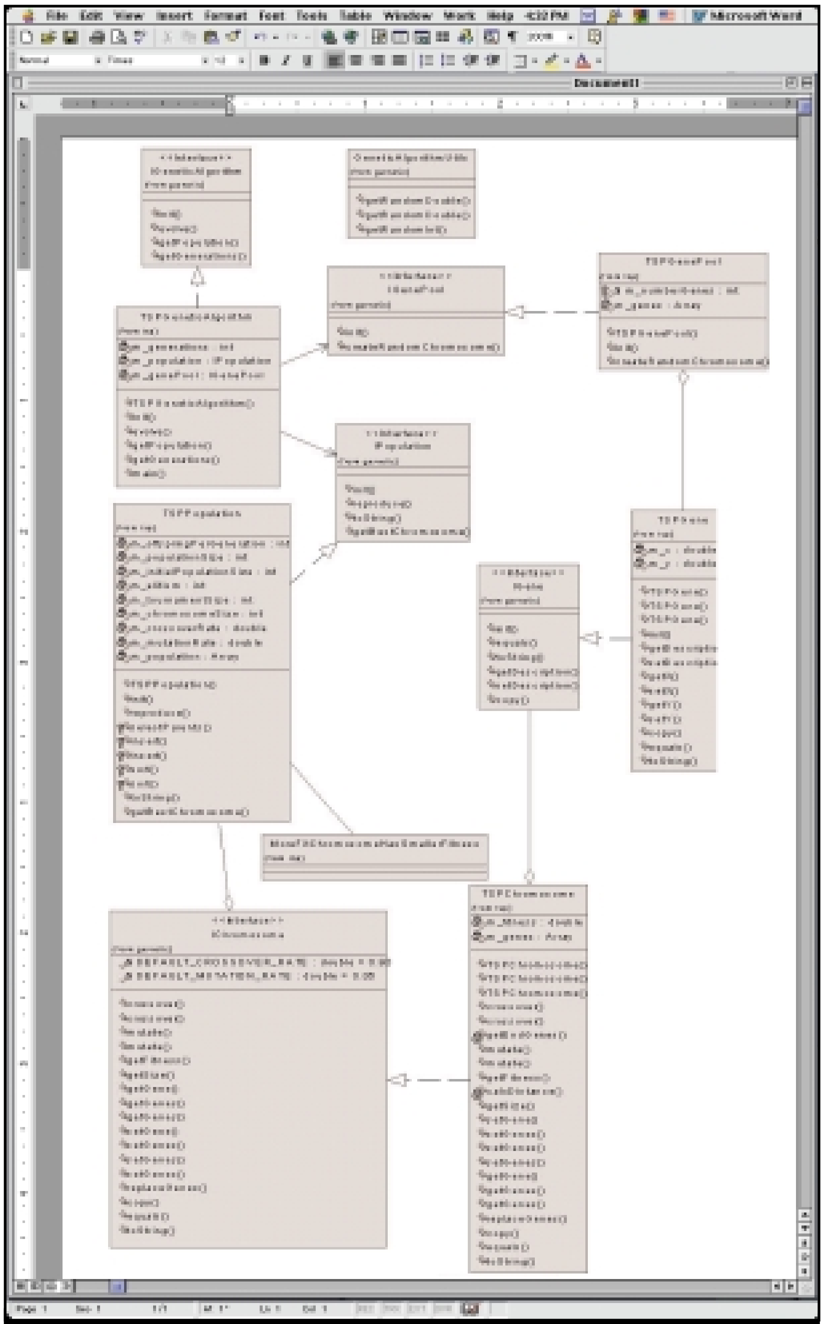

File Edit View Insert Format Font Tools Table Window Work Help
Microsoft Word
Document1

FIGURE 1 UML class diagram for genetic algorithm source code

which a genetic algorithm operates is labeled a *population*.

Through the employment of a fitness function, chromosomes are evaluated and ranked according to their relative strength/weakness within the population. The fitter are chosen to reproduce while the less fit generally don't survive into succeeding generations. After an arbitrary number of generations, the algorithm should converge on an optimal solution or, more specifically, on the chromosome representing an optimal solution. The remainder of this article will illustrate how this is done using, as an example, a classic computer science problem: the Traveling Salesman Problem (TSP).

Traveling Salesman Problem

The TSP is a conceptually simple problem: a salesman wishes to visit a number of cities, starting and ending at the same city, while visiting all cities only once and minimizing the distance traveled. However, the solution to this problem when the number of cities increases is not trivial. For the case of three cities, six permutations representing viable paths are possible. Easy enough. Now let's consider the case of 20 cities in which there are over 2.4 billion billion permutations. If we exhaustively searched through all the possible path traversals at a rate of 500 million per second, it would take over 150 years to examine them all! There is obviously a need for a method to comb through this massive search space more efficiently. Otherwise my great-great-great grandchildren will be the only ones around when the exhaustive search concludes. Meet the genetic algorithm.

Selection of Parameters

As with any problem we attempt to code a solution for, the first step entails clearly defining the problem to be solved, the required parameters for obtaining a solution, and the encoding to be used to represent a solution to the problem. Using a genetic algorithm to obtain an optimal solution for the TSP requires that we identify the relevant parameters and their encoding in terms of chromosomes, genes, and populations. A quick inspection of the TSP reveals the pertinent parameters for encoding a representative solution to the problem: a list of cities to be traversed in order and the distance between two arbitrary cities. Now let's put it into genetic algorithm terminology.

Parameter Representation

As discussed earlier, a chromosome represents a single solution in the search space of the problem being optimized. With reference to the TSP, a chromosome is nothing more than an ordered list of cities in which each city is represented only once, and their order determines the total distance traveled. With chromosomes being a collection of genes, a gene for the TSP is simply an object representing a city (its name and *x* and *y* coordinates). For simplicity I'm using the longitude and latitude measurements as Cartesian *x* and *y* coordinates, respectively.

Listing 1 is the source for TSPGene.java, Figure 1 is the UML class diagram of the source code provided at the end of this article, and Figure 2 is a graphical depiction of how the TSP parameters translate into genes, chromosomes, and a population. Upon examination of the source code, you'll notice that genes are initialized via the retrieval of data from a properties file.

The focus of this article isn't how to build an extendable genetic algorithm framework. However, as you glance through the code you'll notice that through the use of interfaces and externally defined parameters, extendability of the genetic algorithm can be easily achieved to solve, or rather optimize, problems with similar representations. For example, these representations can include permutation problems with unique genes (such as the TSP), permu-

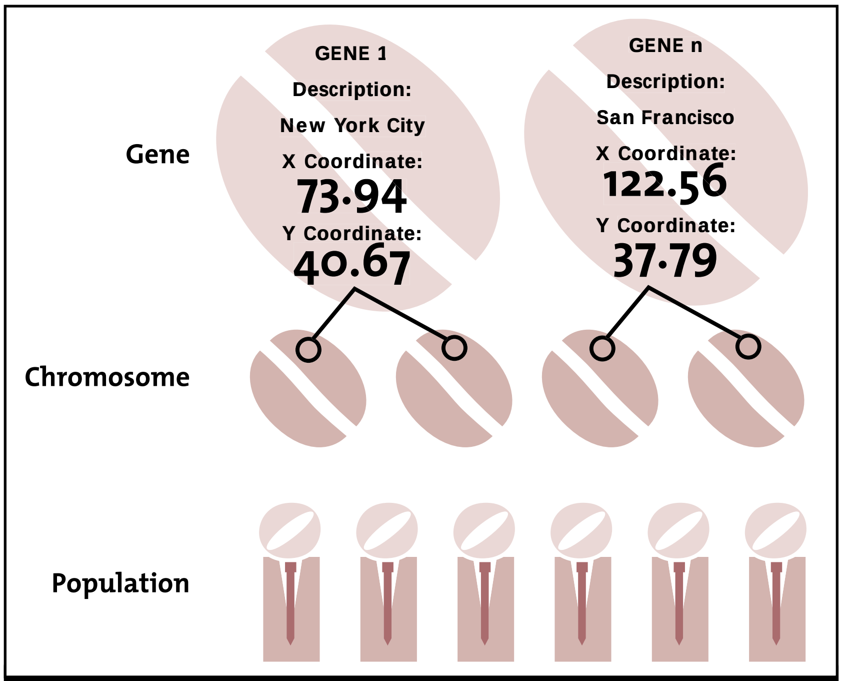


FIGURE 2 Gene, chromosome, and population depiction of the TSP

tation problems without unique genes (meaning a gene may be included zero to many times in a single chromosome), and real-parameter problems in which the genes represent real numbers and their bounds. I leave it to you to develop such a framework, a framework in which all you have to do is set the genetic algorithm parameters and gene data externally and run the genetic algorithm *without* any source code modification. Solving the other types of problems described above then becomes a matter of appropriate problem definition rather than sophisticated programming.

Fitness Evaluation Function

Now that we have the TSP parameters represented as genetic algorithm objects (population, chromosomes, genes, etc.), we can begin to describe the fundamental operations performed on the objects once the algorithm is set in motion. The first one we'll look at is the fitness, or cost, function. It provides the method by which chromosomes are compared to one another in order to determine relative survival fitness. For the TSP the fitness of a chromosome is calculated by traversing the genes in order, summing the distances between adjacent cities. The chromosome with the lowest fitness score is the fittest as it represents the shortest path between the cities. I can't emphasize enough the importance of the cost function as it determines which chromosomes are the fittest within a population. Without an appropriate definition of the fitness evaluation, you might as well rummage around the solution search space at random. Listing 2 provides the source code in TSPChromosome.java in which the fitness is calculated.

Initial Population

With the parameters and cost function defined, it's time to set the genetic algorithm in motion. Recalling the genetic algorithm workflow defined last month, individuals are selected from a population (combined in a process called *crossover*) to create a set of children, and the children are randomly mutated to create a new set of chromosomes to be reinserted into the population. Once enough children chromosomes have been created to replace a population, a generation is said to have *passed*. With each generation, all the chromosomes are evaluated according to the cost function. Only the fittest survive into the next generation where the selection, crossover, and mutate process begins anew. After a number of generations have elapsed, the best chromosome is selected from the population and represents the optimal solution to the problem being solved. Figure 3 depicts a flowchart of the algorithm.

Upon examination of the properties file, you'll notice two properties concerning the population size. One is the initial population size and the other is the "regular" population size. The way I've encoded the genetic algorithm for the TSP is that an initial population of 500 random chromosomes is created and sorted, and the 100 fittest chromosomes are the ones selected for the genetic algorithm starting point.

The advantage of this approach is that by creating a large initial population, you can cover a greater amount of the solution search space initially and then pare down the population to the fittest to focus the search using relatively strong chromosomes. Otherwise, the computational power required to perform a genetic algorithm on a population of 500 chromosomes is far greater than that required for 100 chromosomes.

Selection

Continuing with the genetic algorithm workflow, it's now time to select parents for reproduction. The goal of selection is to choose parents that are capable of creating fitter children while maintaining genetic diversity. What this means is that we want the genetic algorithm to pick not only the fittest to reproduce, but also occasionally the less fit chromosomes to maintain variation in the population. If a few relatively strong chromosomes are continually selected for reproduction, it's likely that the algorithm will converge on these "super" chromosomes since it's their genetic material that's most likely to pass on to the next generation. And it may be the case that the chromosomes represent a local minimum in the cost function rather than the global minimum we're searching for.

The technique I've used in the selectParents() method of TSPPopulation.java (see Listing 3) is referred to as *tournament selection*. A small group of chromosomes is randomly selected, in this case six, and the two fittest are selected to reproduce. Keeping the tournament size small results in a smaller selection pressure, thus increasing genetic diversity.

"...we're approaching a point in software **development where our jobs are becoming more and more reactionary to the competitive landscape that the Internet provides**"

AUTHOR BIO
Michael Lacy is an engineer for the platform development group at Shutterfly, an online photo service, where he develops Web-based solutions for digital image printing, enhancing, and sharing. He's also a certified Java programmer and developer.

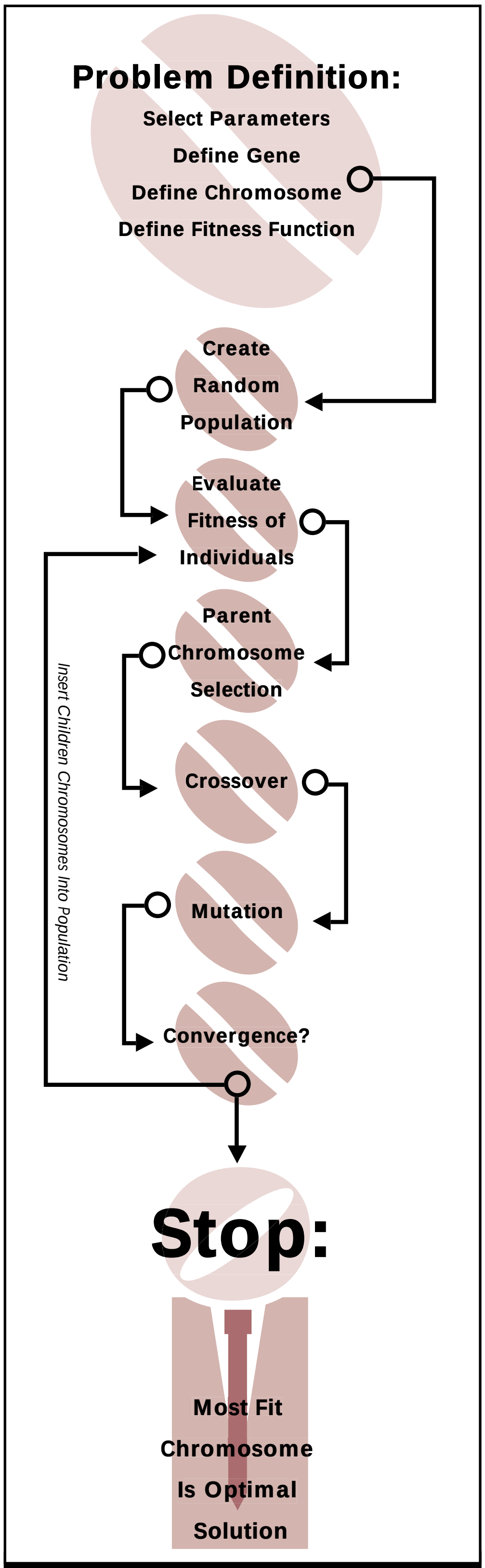


FIGURE 3 Genetic algorithm flowchart

Crossover

Once parents have been selected, the crossover operator combines the two parents randomly to create two children chromosomes. A crossover probability (in our example 90%) is used to determine whether crossover will occur for two given parents. If crossover doesn't occur, the children will be exact replicas of the parents. Crossover is an essential operator for a genetic algorithm as it is the mechanism by which the algorithm "moves" through the solution search space toward convergence.

Continuing with the TSP example, a typical crossover operation includes selecting a chunk of a chromosome and iterating through the other parent chromosome to extract the genes that weren't selected from the first parent in order to combine them in a new child. As discussed earlier, the TSP is basically a permutation problem in which gene uniqueness must be maintained. For other problems, such as permutation without the uniqueness constraint or real-valued problems, crossover takes on a completely different appearance. But for now, we'll use the TSP crossover described above. Listing 4 displays the crossover() method from TSPChromosome.java, which illustrates in code the concepts described above.

Mutation

Mutation is similar to crossover, except that only a single chromosome is involved. As with crossover, there's a probability associated with mutations, albeit a low one (5% in our example). The mutate() method (see Listing 5) defines a very simple mutation method: two genes are selected at random from the chromosome and swapped. A number of mutation techniques are available depending on the problem encoding (permutation, real-valued, etc.), and I recommend further investigation to understand them. Returning to the comparision with crossover, mutation also ensures that a genetic algorithm "moves" through the solution search space, although not always in the direction of increasing fitness. Remember that these operators are blind, that they're determined by probability and chance. The cost function ensures that the population as a whole moves toward an optimal solution.

Insertion

Parents have been selected and children chromosomes created via crossover and an occasional mutation. What next? It's time to insert the children into the population and begin the selection, crossover, and mutation process anew. To preserve the strongest chromosomes within the population, a concept called *elitism* that protects the *n* (*n* = 10 in our TSP example) most elite chromosomes is introduced and replaces the rest of the population with the newly created children.

Convergence

The goal of employing a genetic algorithm for the TSP is to converge on an optimal path (i.e., chromosome) through the cities in which the distance traveled is minimized. The convergence criterion is thus the minimum distance, but since we don't know it a priori, we can't use it to test that the algorithm has converged. Convergence is a key area of research in genetic algorithms, and the approach commonly used today is to provide large populations and a high number of generations to give the algorithm the best chance of convergence. One of the reasons genetic algorithms still exist primarily in academia is that, based on all the parameters that can be set for a genetic algorithm and the different encodings it can assume, there isn't consensus on convergence criteria, meaning that it's largely a process of trial and error for the domain represented.

Conclusion

I've briefly examined the key components of a genetic algorithm and provided examples using the Traveling Salesman Problem. Much of this affirms that use of a genetic algorithm is by no means an exact science and is largely predicated on trial and error in determining an appropriate encoding of the problem being optimized as well as the selection of the parameters controlling the genetic algorithm.

My hope is that I have presented an interesting programming technique and sparked your interest in wanting to know more about genetic algorithms and the power they hold for optimizing complex problems we encounter every day as software developers.

In my opinion we're approaching a point in software development where our jobs are becoming more and more reactionary to the competitive landscape that the Internet provides. To shift back into the proactive role, we need to develop software that relieves us of the burden of coding for every possible scenario. We won't get there overnight, but we have to start somewhere, and the genetic algorithm is a logical starting point as it provides a framework for the evolution of digital applications to solve problems as they arise. I encourage you to dabble with the code provided with this article and think seriously about the future of software development. As Bob Dylan so eloquently stated back in 1964, "the times, they are a-changin'."

michael_lacy@yahoo.com

Listing 1: TSPGene.java

```
package com.lacy.genetic.tsp;
import java.util.Properties;
import com.lacy.genetic.*;
public class TSPGene implements IGene
{
     private String m_description;
     private double m_x;
     private double m_y;

// Constructors
     public TSPGene() {}
     public TSPGene(Properties i_properties, int i_index) {
          init(i_properties, i_index); }
     public TSPGene(String i_description, double i_x, double i_y)
     {
      this.m_description = i_description;
      this.m_x = i_x;
      this.m_y = i_y;
     }

// Method to initialize a gene from a properties file
     public void init(Properties i_props, int i_index)
     {
     setDescription(i_props.getProperty("gene." + i_index +
          ".description"));
     setX((Double.valueOf(i_props.getProperty("gene." + i_index +
          ".x")).doubleValue()));
     setY((Double.valueOf(i_props.getProperty("gene." + i_index +
          ".y")).doubleValue()));
     }

// Accessor/Settor methods
     public String getDescription() { return m_description; }
     public void setDescription(String i_description)
          { m_description = i_description; }
     public double getX() { return m_x; }
     public void setX(double i_x) { m_x = i_x; }
     public double getY() { return m_y; }
     public void setY(double i_y) { m_y = i_y; }

// Utility methods
     public IGene copy() { return new TSPGene(getDescription(),
          getX(), getY()); }
     public boolean equals(IGene i_gene)
     {
     return (i_gene.getDescription().equals(m_description));
     }
     public String toString() { return m_description; }
     }
```

Listing 2: TSP Fitness Evaluation

```
// Returns the total distance traveled for the order of cities
// specified by this chromosome.

     public double getFitness()
     {
          if (m_fitness > 0.0)
               return m_fitness;

          double i_fitness = 0.0;

          for (int i = 0; i < m_genes.size(); i++)
          {
               TSPGene g1 = (TSPGene) m_genes.at(i);
               TSPGene g2 = null;
               if (i == (m_genes.size() - 1))
                       g2 = (TSPGene) m_genes.at(0);
               else
                       g2 = (TSPGene) m_genes.at(i+1);

               i_fitness += calcDistance(g1.getX(), g1.getY(),
                       g2.getX(), g2.getY());
          }

          m_fitness = i_fitness;

// Calculates the cartesian distance between two points specified
// by points (x1, y1) and (x2, y2) using the Pythagorean Theorem.

     private static double calcDistance(double x1, double y1,
     double x2, double y2)
     {
          double deltaXSquared = (x2 - x1) * (x2 - x1);
          double deltaYSquared = (y2 - y1) * (y2 - y1);

          return Math.pow(deltaXSquared + deltaYSquared , 0.5);
     }
```

Listing 3: Parent Tournament Selection

```
// Returns 2 IChromosome objects using tournament selection
     protected IChromosome[] selectParents(int i_tournamentSize)
     {
          IChromosome[] parents = new IChromosome[2];
          Array gladiators = new Array(i_tournamentSize);

// pick a random spot within the population
            int startIndex = GeneticAlgorithmUtils.getRandomInt(0,
             m_population.size() - i_tournamentSize);

// get as many chromosomes as specified by tournament size
            for (int i = startIndex; i < (startIndex +
             i_tournamentSize); i++)
            {
              gladiators.put(i - startIndex, m_population.at(i));
            }

// sort the sub-population by fitness
            sort(gladiators);

// return the best two
            parents[0] = (IChromosome) gladiators.at(0);
            parents[1] = (IChromosome) gladiators.at(1);

            return parents;
            }
```

Listing 4: Crossover

```
/** Performs a crossover with this chromosome and the one speci-
fied in the argument. Returns two children chromosomes while pre-
serving the uniqueness of genes in the chromosomes, meaning that
all genes will be represented once and only once in the chromo-
some. */

public IChromosome[] crossover(IChromosome i_chromosome, double
    i_crossoverRate

    {
    int size = this.getSize();
    IChromosome mom = this.copy();
    IChromosome dad = i_chromosome.copy();
    IChromosome[] children = new IChromosome[2];
    double random=GeneticAlgorithmUtils.getRandomDouble(1.0);

    if (random < i_crossoverRate)
    {
    int crossoverPoint=GeneticAlgorithmUtils.getRandomInt(1,size-1);

    IGene[] child1begin=mom.getGenes(0, crossoverPoint-1);
    IGene[] child1end=getEndGenes(child1begin,
                                  dad.getGenes(0,size-1));
    IGene[] child2begin = dad.getGenes(0, crossoverPoint - 1);
    IGene[] child2end = getEndGenes(child2begin, mom.getGenes(0,
                                       size -1));
    children[0] = new TSPChromosome(child1begin, child1end);
    children[1] = new TSPChromosome(child2begin, child2end);

    return children;
            }

// if no crossover, children are same as parents
            children[0] = mom;
            children[1] = dad;

            return children;
      }
```

Listing 5: Mutation

```
/** Mutates this chromosome by randomly selecting two genes and
swapping them Uses the mutation rate specified by i_mutationRate
*/
      public void mutate(double i_mutationRate)
      {
      double random = GeneticAlgorithmUtils.getRandomDouble(1.0);

      if (random < i_mutationRate)
       {
       int size = this.getSize();

       int r1 = GeneticAlgorithmUtils.getRandomInt(0, size - 1);
       int r2 = GeneticAlgorithmUtils.getRandomInt(0, size - 1);

       IGene g1 = (IGene) this.getGene(r1);
       IGene g2 = (IGene) this.getGene(r2);

       m_genes.put(r2, g1);
       m_genes.put(r1, g2);
            }
      }
```

Listing 6: Insertion

```
/** Method to insert children into the population, preserving the
m_elitism best from the previous generation */

protected void insert(IChromosome[] i_children, int i_elitism)
      {
            sort(m_population);

            for (int i = i_elitism; i < m_population.size(); i++)
                m_population.put(i, i_children[i - i_elitism]);

      }
```

Download the Code!
www.JavaDevelopersJournal.com

System Robustness Case Study: A Smart Location Service

Architecture analysis and planning are crucial

WRITTEN BY
KHANH CHAU

As a distributed object technology, CORBA provides tremendous flexibility for implementing robust enterprise information systems composed of distributed components. In large-scale deployment, these components run in multiple servers located in multiple hosts of different operating systems.

The CORBA architecture facilitates integration of these distributed components and defines a mechanism for obtaining the required object references.

In particular, the CORBA programming model requires that remote object implementations be published so client processes can locate them. To provide this directory capability, OMG defines two specifications: naming service and trader service, both of which specify the interfaces for object references registration, lookup, and management. Many directory service packages are commercially available and generically implemented. But a robust enterprise implementation must meet a number of requirements: scalability, reliability, availability, and more, all of which require intimate understanding of the problem domain. Since the generic implementations of the directory service aren't able to address those requirements, a smart location service is needed to achieve system robustness.

Building a location service, however, consumes time and resources, and can also get in the way of other development. In this article, I outline the steps that allow a location service to be incrementally built, tested, and released into a development environment in baby steps, unintrusively.

To begin, I'll discuss factors that influence the decision for building a location service, then define an implementation plan. To wrap up, I'll discuss a sample application in an enterprise architecture integration (EAI) effort.

System Robustness

When building an enterprise information system, the architecture team is tasked with requirements that describe architecture robustness. System robustness comes in many forms. In general, a good design seeks to balance five essential elements: scalability, reliability, availability, fault-tolerance, and manageability.

Scalability defines the system ability to handle more workload by adding more computing resources in a predictable manner. If the resource requirement increases exponentially as the demand increases, the system is not scalable. Be careful to differentiate scalability from increasing speed. Planning to add more resources to "speed up" transaction processing time is not a design for scalability. Speed enhancement has limits and is successful only in conjunction with improvements in several areas. Doubling the number of CPUs doesn't automatically cut processing time in half.

Reliability is a measurement of resilience to system failures. Mean time between failures (MTBF) is a standard measurement. In many cases, components that are well tested do not break, while the under-tested components do. System errors such as network and I/O also contribute to system failures. Thus a reliable system must handle those conditions gracefully, which closely relates to the topic of fault tolerance. Replication of system resources is a common prescribed solution for fault tolerance.

Availability defines a measurement of online time. System components that run but are unable to handle new requests do not contribute to system availability. For a system to be available, the components must have the ability, or an appearance, to handle new requests.

To support the above, the system frequently expands beyond the scope of the local network. More resources are strategically added at various locations to expedite fulfillment of client requests. A successful expansion helps scale the system, but it creates more points of failure. To support the expansion, enterprise system management tools are often used to monitor and manage software applications and hardware devices.

For an enterprise information system to be robust, it must address these issues. Architecting a smart location service is a crucial step in that direction, because to enable scalability and availability, more servers are started to host replications of remote objects. This raises the issue of load balancing so that requests are distributed evenly among servers. A round-robin scheme provides a basic functionality, but a smart location service helps client processes find the best available remote services. In this way requests

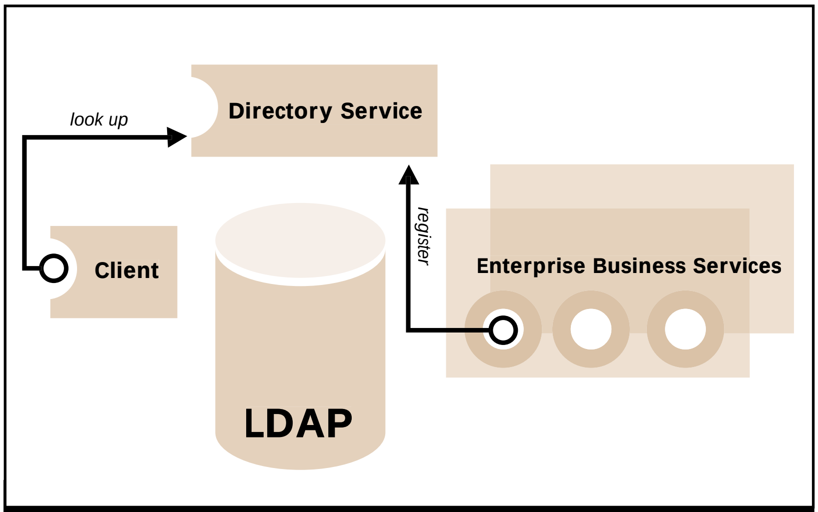


FIGURE 1 Typical use of directory service

are serviced in a timely manner. To achieve this, the location service collects statistical information and acts upon them using predefined algorithms.

Since the location service acts as a hub to monitor and collect vitals from CORBA servers and objects, it can be used to assist a system management tool by activating or deactivating server instances as needed. In addition, a scheme for server migration, deactivating a server in one host and starting it in another, helps load balance computing resources. Restarting servers that hang up increases overall system reliability and fault tolerance. This effort relieves computing resources for other processes to use. All these tasks contribute to meeting the design requirements of building a robust enterprise information system.

CORBA Object References

The OMG IDL is a descriptive language used to define an interface that the client can call and the server's object implementations can provide. These object implementations are activated inside one or more server containers. The servers publish these as object references, which essentially contain the information on how to locate the servant and the signature of the interface or IDL type. Both are needed for the ORB to facilitate the communication link from the client's proxy object to the remote object implementation.

Before making remote invocations, the client process must obtain the object reference. Since this is a necessary step in the process, all ORB products support proprietary mechanisms to manage object references. Most, if not all, provide full implementations of the OMG naming specifications and allow persistence of object references onto backing stores such as LDAP or RDBMS. Few vendors provide trader implementations. The difference between a naming service and trader service is that the latter allows publication by type versus name. Type publishing allows more flexibility because properties can be associated. Nevertheless, the two services serve the same basic goal – to provide a facility for clients to look up an object reference, and for servers to register/deregister object references (see Figure 1).

To handle more workload, a system is deployed with multiple instances of the same type of remote objects. This presents an issue for client processes that use the naming or trading facility. It requires cumbersome coding, in the naming service case, to navigate the naming context hierarchy, and lacks the means for finding the best object reference. Given a choice of multiple object references in one lookup, the client process has to answer these questions:

- On which servers are these object references located? Which server is the least busy? Does it have the capacity to handle my requests in a timely manner?
- Is it better to connect to an object reference hosted by a less busy server two hops away or a busier server in the local subnet?

Furthermore, stored object references do not indicate that the remote server (that hosts the remote object) is running. Another problem is the remote server, because of IDL-type version difference, might not be able to activate the needed servants. It's possible for a client process to receive a "stale" object reference, thus causing a remote exception when the client attempts to narrow or invoke the operations. In the first case, more code is added to the client side to handle exceptions and initiate another lookup. In the versioning problem, the client is unable to perform work and is likely forced to exit. The IDL-type versioning feature is necessary to support incremental system development.

Location Service Architecture

As more code is added to the client side, more administration is needed to make sure clients are using the same lookup policy. It's better to devise a location service to centralize the policy and serve purposes such as verifying object references, collecting server vitals, and collaborating with system management tools. This location service does not have to be built from the ground up. Since the goal is to enhance the directory service capability, it should be designed using existing directory implementations. Numerous free Java and C++ implementations of naming and trader services are already available from organizations such as www.trcinc.com, http://jacorb.inf.fu-berlin.de/ , and www.mico.org.

By inserting a location service, the client process looks up object references

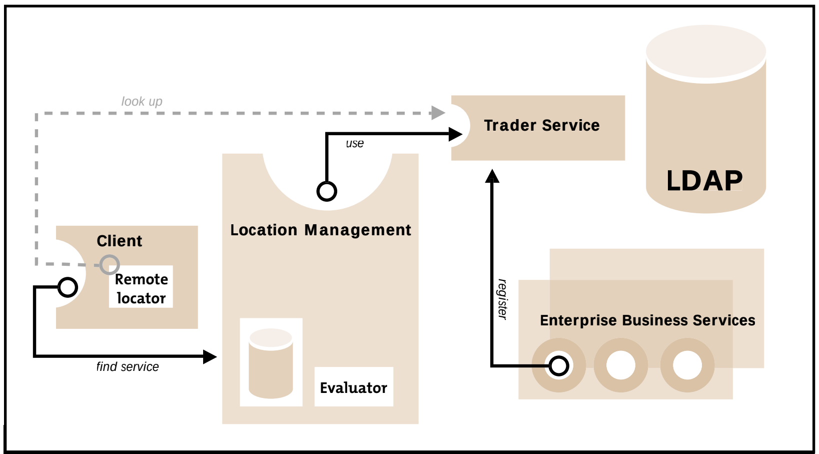


FIGURE 2 Use a location service to smartly manage object references

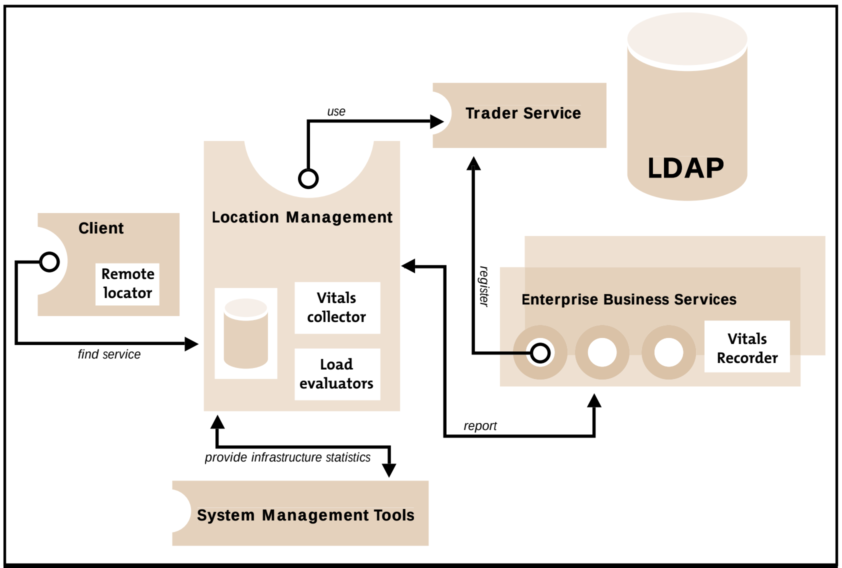


FIGURE 3 Location service integration architecture

via a different route (see Figure 2). Ideally the location service sports the same interface as the directory service (that is, either naming or trader) but doesn't have to. The location service uses an evaluator – a predefined algorithm – to select an object reference from the presorted list. When the client process receives an object reference from the location service, it's assured that the object reference is the best available in the system.

Figure 2 also shows the relative position of the location service compared to Figure 1. Enterprise business services continue to use the directory service of choice for object registration and deregistration. In this particular example, a trader service implementation is selected and runs in its own process space. It can collocate in the same memory space as the location service; thus it appears to run in one single process. Either way, the trader service is accessible from both the client's remote locator agent and the location service. This pattern is useful for incremental development of any enhanced service.

The remote locator running in the client process defines an interface that the client uses to request binding to remote object references. In this configuration, the initial implementation of the remote locator interface enables lookup of the object reference via the trader service. As the location service implementation is completed, it's inserted seamlessly by updating the implementation of the remote locator.

Note that the trader service uses a backing store such as LDAP for storing object references. The location service has a working area, which is done entirely in memory. The working area contains lists of object references that are indexed and updated by the evaluator. If the location service is restarted, the evaluator prioritizes the lists using newly collected system information. This configuration speeds up the update and retrieval of object references in the location service during normal operations.

Location Service Implementation

Large-scale system development using distributed objects is difficult. Risks must be monitored, contained, and mitigated. Especially during the initial cycle of development, an attempt to do too much can negate the progress. A game plan must be developed to make a smooth transition from the initial phase deliveries to deployment.

Implementing a robust location service, therefore, can be a daunting task and should be done in several phases. Each problem domain has unique user and system requirements that define the breakdown. For many implementations, the following guidelines can be used:

- ***Phase 1:*** Provides the infrastructure to support future development. During this time, the infrastructure team defines and packages a set of classes, and configures an environment to facilitate the registration and lookup of object references. For Java development use the Java Naming Directory Interface (JNDI) if possible, since it allows interoperability of different types of directory service implementations. Instructions should be given so that developers understand javax.naming package usage and the hierarchy the team comes up with. For a non-Java environment, more work is needed to supply the implementation that mimics the javax.naming subset of the JNDI service. During this phase, select a directory service type and find an available implementation to bootstrap the development.
- ***Phase 2:*** Defines an implementation with minimum features that must be supported by the location service. Statistical information or system vitals such as the number of running threads, memory usage, and processing time need to be classified. The algorithm to manage an object reference is defined according to these vitals. For instance, before returning an object reference, make sure it's ready to handle requests or provide an object reference with the least workload. Keep in mind that the location service complements the existing CORBA directory service.
- ***Phase 3:*** The location service is integrated with external services for better system management. System management tools can provide feed-

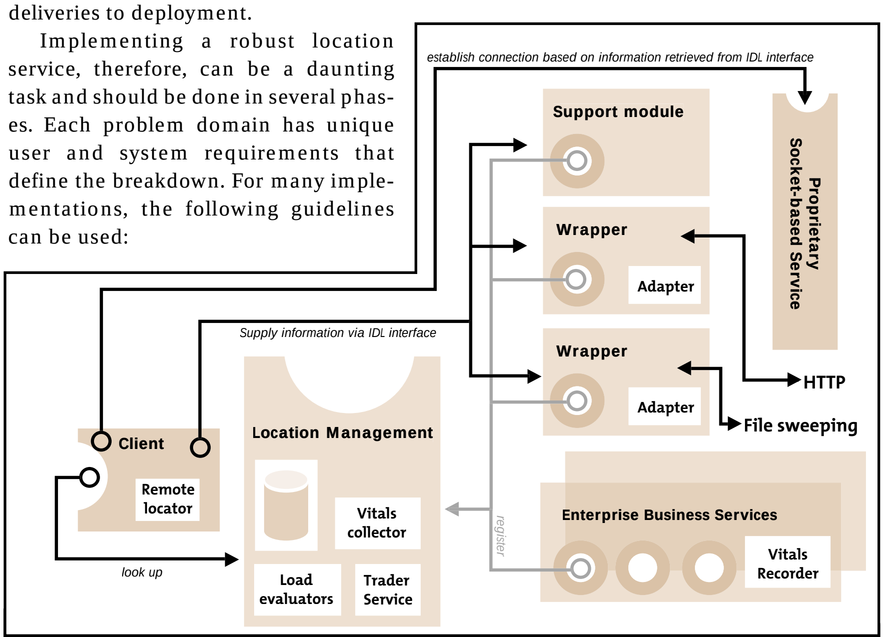


FIGURE 4 EAI application

back of the general system's health such as network utilization, I/O, and memory usage, which are important vitals for the location service. In addition, the location service can utilize system management tools to launch or shut down server processes in selective hosts (see Figure 3).

Once the groundwork is in place, implementing the location service occurs concurrently with other development tasks. This effort consists of designing an IDL type that supports the CosNaming interface, and the location service engine, which consists of collectors and evaluators. The collectors are responsible for gathering statistical information from running remote processes and possibly from the system management tool. The evaluators act on the vitals by retrieving the registered object references and apply sorting rules. These rules for managing object reference order are unique within the problem domain.

AUTHOR BIO

Khanh Chau, an infrastructure architect with The Technical Resource Connection, Inc., is also an instructor and project lead for The TRC Java Developer boot camp program. He helped design, implement, and deploy a spine infrastructure framework (SIF) architecture for a major national real estate services company, and is currently developing an enterprise e-commerce portal for one of Germany's largest insurance companies.

An Example of EAI Use

An enterprise architecture integration (EAI) effort that uses CORBA can benefit from implementation of a smart location service. Consider a scenario in which enterprise data is served by several legacy subsystems (see Figure 4).

- ***A subsystem that serves data via an ASCII-delimited format using Winsock:*** In this scenario, a wrapper is created and defined via IDL. The interface returns crucial information (e.g., IP address, port) for the client to connect to the legacy service. In addition, the interface could return metadata information used to validate request and response messages.
- ***A subsystem that accesses XML information using HTTP***: An adapter interface is created to facilitate data transferring using HTTP protocol. The retrieved data is parsed, categorized, and persisted as needed. The interface provides information for the subsystem to do its work, and is driven by the client
- ***A subsystem that communicates business data via a file-sweeping approach:*** Files that need to be processed by the client are located in the inbound directory and the results are stored in the outbound directory. The client does not know the location of the inbound/outbound folders. The wrapper shields the knowledge and allows implementation to access data via predetermined business logic.

In each scenario, the subsystems are wrapped so they become accessible via IIOP. The publishing and lookup of the wrappers are stored in the location service, hence creating a loosely coupled yet highly cohesive mechanism.

Summary: Where Do You Want to Go?

A prerequisite for building large-scale enterprise information systems is an understanding of the requirements and technology used. CORBA defines a programming model to connect disparate systems and components while providing boundless opportunities to customize for system robustness. This opportunity comes with a price because its implementation consumes time and resources. In today's fast-moving business, dealing with system robustness is frequently an afterthought.

This is acceptable when an organization tries to get its footing by delivering a working system. However, effort should occur in parallel to identify an architecture that enables growth. A strategy is needed to make sure a migration path is available. Good software practices and strong understanding of distributed computing issues are essential. But there's no substitute for architecture analysis and planning, as it's crucial for building robust enterprise information systems.

khanh.chau@trcinc.com

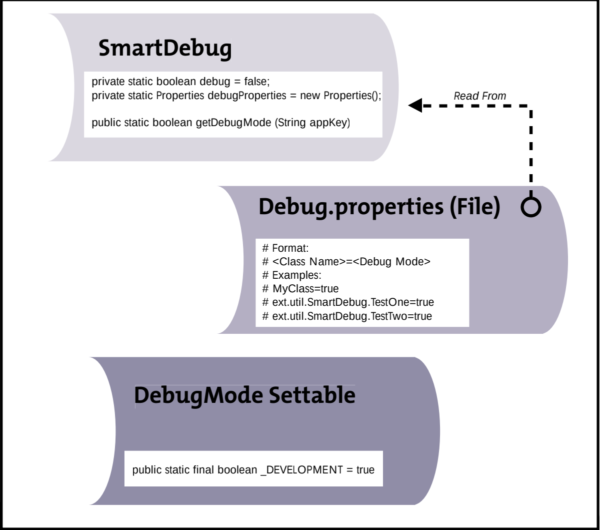
Conditional Compilation in Java

Clean debug code is its best use

WRITTEN BY
SREEDHAR CHINTALAPATY

Conditional compilation is not available in Java – and Java's platform independence is the cited (and largely justified) reason. Nevertheless, one valuable use of conditional compilation, which is to cleanly insert debug code into applications, is thereby lost.

Several methods to work around this disadvantage are available. Joe Chou details these approaches in previous issues of ***Java Developer's Journal*** (see www.syscon.com/java/archives/0312/chou/index_b.html).



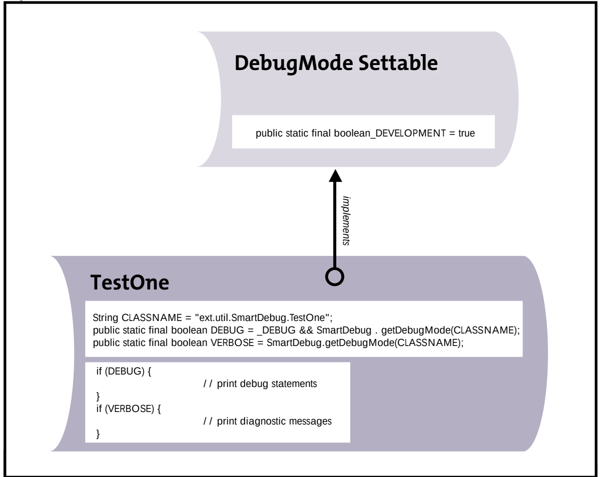
```
private static boolean debug = false;
private static Properties debugProperties = new Properties();

public static boolean getDebugMode (String appKey)
```

```
# Format:
# <Class Name>=<Debug Mode>
# Examples:
# MyClass=true
# ext.util.SmartDebug.TestOne=true
# ext.util.SmartDebug.TestTwo=true
```

```
public static final boolean _DEVELOPMENT = true
```

FIGURE 1 The three components of SmartDebug



```
public static final boolean_DEVELOPMENT = true
```

```
String CLASSNAME = "ext.util.SmartDebug.TestOne";
public static final boolean DEBUG = _DEBUG && SmartDebug . getDebugMode(CLASSNAME);
public static final boolean VERBOSE = SmartDebug.getDebugMode(CLASSNAME);
```

```
if (DEBUG) {
                  / / print debug statements
}
if (VERBOSE) {
                  / / print diagnostic messages
}
```

FIGURE 2 How to use SmartDebug

In this article I propose an approach that's arguably simpler, more scalable, and more flexible than any existing method. Although it can potentially be used for full-fledged conditional compilation, we'll limit ourselves to the problem of clean debug code.

Problem Statement

From a functional point of view, informational output can be classified as either:

- ***Diagnostic messages:*** Designed to remain in the code even after release. These are what an as-released application writes out into its logs when executed in verbose mode.
- ***Debug messages:*** Used by programmers in the development stage to debug their code. These correspond to the program's debug mode execution and would ideally be deleted from the release version of the code.

Therefore, we need a simple lightweight approach that can:

1. Handle selection of classes to run in debug (or verbose) mode without having to recompile them.
2. Leverage the compiler's built-in optimization capabilities to safely ignore debug code when compiling the release version of the application.

The Proposed Solution

The proposed solution consists of three components as shown in Figure 1:

- ***Debug.properties:*** A text file containing class names and the desired debug mode, respectively.
- ***DebugModeSettable:*** An interface used primarily to hold a final Boolean value _DEVELOPMENT. This is set to true during _DEVELOPMENT. When the code is ready for shipping, this is set to false and the entire code is recompiled.
- ***SmartDebug:*** A class that provides access into debug.properties via a properties object. This provides a mechanism for class-level control at runtime.

To see how this design leverages built-in optimization features of Java compilers to conditionally compile code, consider how Java compiles the class TestOne: with DebugModeSet-table._DEVELOPMENT set to true, the compiler expects to determine the value of TestOne.DEBUG at runtime by calling SmartDebug.getDebugMode(). Therefore, all code enclosed within if (DEBUG){…} is compiled.

However, with DebugModeSettable._DEVELOPMENT set to false, Java knows that TestOne.DEBUG is always going to be evaluated to false. So it does not compile any code enclosed within if (DEBUG_){…}.

Verifying the above is trivial:

1. Compile the class Test-One.java with DebugModeSettable._DEVELOPMENT set to true; decompile TestOne.class.
2. Now compile TestOne.java with DebugModeSettable._DEVELOPMENT set to false and decompile TestOne.class.
3. Compare the two decompiled versions.

VERBOSE, on the other hand, will always have to be evaluated at runtime. So any code enclosed within if (VERBOSE){…} will be compiled, regardless of the value of DebugModeSettable._DEVELOPMENT.

Usage

Figure 2 illustrates the use of this approach.

As an example, assume that you

would like to use this approach on a class called MyClass. The recipe, then, is as follows:

1. Let MyClass implement DebugModeSettable.
2. Set the following classwide variables in MyClass:

```
 private final String CLASSNAME = "
MyClass ";
 private final boolean DEBUG =
_DEVELOPMENT && SmartDebug
.getDebugMode(CLASSNAME);
 private final boolean VERBOSE =
SmartDebug.getDebugMode(CLASSNAME);
```

3. Wrap debug code within if (DEBUG){…}, and diagnostic code within if (VERBOSE){…} respectively.
4. Edit Debug.properties to add the following line:

```
 MyClass=true
```

MyClass.java may look like the example shown in Listing 1.

Limitations

We have used this approach profitably in a couple of projects at PTC. The limitations of this approach are as follows:

1. Every top-level class needs to implement the interface DebugModeSettable.
2. It takes a little discipline on the part of programmers to enclose their debug/diagnostic code appropriately, but the gains far outweigh this burden.

Conclusion

This approach can be used to conditionally compile any code, not just debug/diagnostic code. On those rare occasions when programmers *do* need to port their Java code, it could prove just as valuable to Java as conditional compilation is to C/C++. Clean debug code, however, is its best and most uncontroversial use.

sreesarma@yahoo.com

AUTHOR BIO

Sreedhar Chintalapaty is an implementation consultant with PTC. He is a java neophyte, currently involved in developing product data management and collaborative product commerce solutions using PTC's Windchill and NetMarkets products.

Listing 1

```
import ext.util.*;

public class MyClass implements DebugModeSettable {
   private static String CLASSNAME              = "MyClass";
   private static final boolean DEBUG = _DEBUG &&
SmartDebug.getDebugMode(CLASSNAME);
   private static final boolean VERBOSE = SmartDebug.getDebugMode(CLASSNAME);

   // Some stuff here
   // resetStatus()
   // terminateProgram()

   public static void doSomething(String action) {
   if (VERBOSE) {
  System.out.println("Entered MyClass.doSomething - Action = " + action);
   }

   int currentStatus = getCurrentStatus();

   if (action.equalsIgnoreCase("TerminateProgram")) {
  if (DEBUG) {
   System.out.println("In MyClass.doSomething - Terminating Program; Exit
Status is " + currentStatus);
  }
  terminateProgram(currentStatus);
   }

   if (action.equalsIgnoreCase("Reset")) {
  if (DEBUG) {
   System.out.println("In MyClass.doSomething - Current Status " +
currentStatus + " being reset");
  }
  resetStatus();
   }
 } // End doSomething

}
```

Download the Code!
www.JavaDevelopersJournal.com

Web Application Framework

Support your application development with Struts

WRITTEN BY
Craig McClanahan,
Larry McCay, &
Erik Bergenholtz

The J2EE architecture is a great advance for developers. Its standardized framework defines and supports a multitiered programming model, freeing application developers to concentrate on solutions.

This article is based on version 0.5 of Struts; some of the APIs will change slightly in subsequent versions.

There are a number of ways to architect applications using J2EE technologies, including JSPs, servlets, and EJBs. However, there aren't any application development frameworks available that address how developers should create applications based on the J2EE architecture.

The development community has recognized the need for an open-source framework that's useful in building Web applications using J2EE technology. This has led to the creation of the Struts project, a burgeoning grassroots movement that will soon become a J2EE "household name."

Struts Introduction

Struts is an open-source initiative from the Jakarta Project, which is sponsored by the Apache Software Foundation. The framework was created by Craig McClanahan in May of 2000, after which the code was donated to the Apache Software Foundation and then to the development community.

The Struts charter is to provide an open-source framework that facilitates building applications based on the J2EE servlet and JSP technologies. The primary functionality provided by Struts includes request dispatching, tag libraries to accelerate GUI development, internationalization, and automatic form population.

Struts encourages application architectures based on the Model-View-Controller (MVC) design paradigm, also known as Model 2.

Model 1 (page-centric) vs Model 2 (servlet-centric)

In a Model 1 design the application logic and program flow are combined into JSP pages. Model 1 designs are considered page-centric because the application flow is controlled by the JSP pages. Therefore, there's a tight coupling between the pages and the logic in the application.

However, while Model 1 is well suited to relatively simple applications, it's limited for complex ones. For example, the fact that the JSP pages in a Model 1 design contain both the presentation and control logic means the application is more difficult to maintain. A large amount of Java code within the HTML eliminates the role separation between Web page designer and Java coder, creating dependencies across multiple developers and increasing the likelihood of errors.

In addition, Web applications inherently have flow control issues, which can lead to many problems unless proper precautions are taken to avoid them. In many cases, "proper precautions" mean the creation of a great deal of duplicate code. For instance, to facilitate proper security, a developer must not make assumptions about the users' authentication before they arrive at the current page. This requires some sort of user authentication to happen in every page.

In a Model 2 design the application flow is controlled by a central servlet that utilizes an implementation of the Mediator design pattern to delegate requests to the appropriate handler. The request handler will most often be a presentation JSP; however, it could also delegate a processing page JSP or servlet to handle the application logic. Then the JSP or servlet forwards the request to the appropriate presentation page back through the servlet. To acquire the appropriate page to forward the request to, resolve the logical name that's passed from a configuration file or database to the servlet as part of the pathinfo or as a request parameter. This provides a loose coupling of the pages and components within the application.

Model-View-Controller

Another term used to describe the Model 2 design approach is *Model-View-Controller*. In describing this design, the JavaBeans or EJBs used within the application are considered the Model, JSP pages are used as the View, and the mediating servlet is considered the Controller.

The Struts framework provides developers with a well-thought-out design that allows you to focus your program flow, validation, and request dispatching. Struts accomplishes this by providing specific components for each piece of the MVC architecture – Model, View, and Controller.

Model

The Model represents the data that the View presents. In Struts, the Model is comprised of a few specific Struts components, as well as application and business logic, encapsulated in a reusable component such as a JavaBean or an EJB. The Struts-specific Model components include ActionForm and Action classes.

The ActionForm class serves several purposes in Struts. Typically, when building a Struts application you'll implement a class that implements the ValidatingActionForm interface for each input form in your application. The ActionForm should contain the attributes that you've defined in your input form using the tag libraries provided by Struts. When the form is submitted, ActionServlet creates or reuses an instance of the ActionForm associated with the input form and adds this instance to the user's session. The attributes of the input form will be populated in the ActionForm instance by calling the corresponding mutator methods. You're required to implement accessor and mutator methods for each property defined in your ActionForm.

Once the ActionForm is populated, the validate method is invoked. The validate method is responsible for ensuring that the user populates the required fields with data in the appropriate format. As you can see from the ValidatingActionForm interface, the validate method returns an array of strings. If the validate method determines there's invalid data in the ActionForm (nonexistent or otherwise incorrect), the strings returned from the validate method are communicated to the user (through the <struts:errors/> tag). If errors are detected, the original form that was submitted is populated using the values in the ActionForm, and the error messages returned by your ActionForm are displayed back to the user. This way the user only has to change/correct the fields that were determined to be incorrect.

Once the validate method returns null, indicating there are no errors, the ActionServlet calls the corresponding perform method of the Action class. The perform method is responsible for processing the request and subsequently forwarding the request to the appropriate view component (a JSP) as indicated by the returned ActionForward instance. When implementing an Action, you can either implement the Action interface or extend the abstract base class ActionBase. ActionBase provides default implementations for the getLocale, setLocale, getResources, isCancelled, and saveErrors methods. It's not recommended that the Action contains business logic, but it should make calls into the business component stack that implements the logic. After the Action has processed the request, it should return an ActionForward object that identifies the JSP page to which control should be transferred. This JSP is responsible for generating the appropriate response to the requesting device.

The benefit Action provides is that business components don't need to be aware that they're accessed from a servlet/Web environment. Business objects shouldn't take ServletRequest and ServletResponse as parameters. This tightly couples business components to a Web environment and limits the scope in which they can be used.

View

Struts relies on JSPs to create the appropriate View for the requesting client. For convenience reasons Struts provides a comprehensive tag library (based on the JSP custom tag library facility in JSP 1.1). These tag libraries can be used to construct the user interface. The advantage of using these tag libraries is that they provide some additional functionality. For example, the following snippet may be found in an HTML/JSP form:

```
<input type="text" name="userName"
value="<%= user.getUserName() %>">
```

Using Struts custom tag libraries, the above would become:

```
<struts:text name="userName">
```

The advantage here is that HTML developers may not be familiar with scripting beans, such as the one shown in the first snippet. The Struts tag library gets around this by providing a simple tag library that can be used by nonprogrammers. The second snippet assumes the <struts> tag is nested within a Struts form tag, so the field will call upon the correct bean to get its values.

When implementing a Struts application, use the Struts tag libraries to construct the user interface. Included in this tag library are tags for typical form elements such as checkboxes, hidden fields, and radio controls. In addition, the library contains various tags for displaying data, creating hyperlinks, and internationalization.

Controller

The Controller in the Struts framework is implemented by the ActionServlet class. As the name describes, the Controller is implemented as a servlet. The servlet is responsible for delegating requests to the appropriate components as specified according to the action.xml file. Each Struts-based Web application requires an action.xml file, the same way you have one Web.xml per Web application. The action.xml file provides logical mappings that the ActionServlet will use to determine how to process the request. Specifically, the mapping tells ActionServlet which Struts components to use and what the program flow is. This is done by providing a mapping for each requested URI (i.e., /logon) to an Action class and an ActionForm class. The benefit of this is that Action classes can be leveraged for multiple requests; changes can be made to the application without requiring code modification. The action.xml mapping file can also provide a mechanism to separate program flow from business logic.

You'll need to implement an ActionMapping class if additional mapping information is specified in the action.xml file. However, if no such additional attributes are needed, Struts provides a default implementation (ActionMapping) that knows how to deal with the default XML format of action.xml.

In the sample application that's part of the Struts distribution, there are two additional attributes in action.xml – *success* and *failure*. These attributes define which JSP page should be forwarded to which request based on a success and failure condition of the validate method of the ActionForm class.

A Note on Internationalization (I18N)

We've already stated that Struts provides developers with an extensive tag library. One of the most striking features of this tag library is a message tag that deals with internationalization (I18N).

“Struts provides **developers with an** **extensive tag library**”

AUTHOR BIOS

Craig McClanahan was involved in the Apache JServ project, focused on implementing a next generation architecture and feature set for the core servlet engine. He has recently joined Sun as technical lead for the servlet and JSP reference implementation, Tomcat. Craig has a BA in accounting from the University of Puget Sound.

Larry McCay is a senior architect at Bluestone Software and has been a major contributor to Bluestone's Total-E-Business platform and its Application Manager. He's been designing and developing software for over 10 years. Larry has a BS in business administration with a specialization in MIS.

Erik Bergenholtz is a development manager at Bluestone Software and has been a major contributor to Bluestone's Total-E-Business platform. He's been designing and developing commercial software for the past 10 years. Erik has a BS in computer science and an MS in software engineering.

The functionality provided through this tag is based on what's already provided by the Java platform: PropertyResourceBundle and MessageFormat. The approach is simple yet effective.

Beyond what's already provided by Java, Struts provides a MessageResources class that lets you treat a set of ResourceBundles as a database. In Java, when you request a message from a ResourceBundle, it's based on the default locale of the JVM. This is not useful when the JVM in which your application is running is serving up content for individuals with different preferences. Struts allows you to request a message string for a particular user for a specific language preference. This is implemented by having the application store a locale object within the user's session under the name "org.apache.struts.action.LOCALE". If no locale object is found in the user's session under this key, the default resource bundle will be used.

To support multiple languages in your application using Struts, provide a resource bundle for each locale that you want to support, along with the appropriate extension. The base name of the resource bundle to be used by the application is specified in the application's deployment descriptor.

The message tag is used as follows within an HTML/JSP form:

```
<struts:message key="mykey"> where
"mykey" is the key into the proper-
ties file.
```

The Future of Struts

Interest in Struts has increased dramatically over the past few months. At the time of this writing, Struts has a following of 900 mailing list members as well as four committers.

Since Struts has such a large number of involved participants (and more are expected to join), it's likely that the Struts framework will mature significantly over the next few months. In addition, commercial software companies are starting to look at the Struts framework as a possibility on which to base their development efforts. Plus, as a J2EE developer, you'll be hearing more about Struts in the near future since Sun is planning to integrate the Struts framework into their Blueprints example application implementation.

There are a number of ways to participate in the Struts project. You can join one of the Struts mailing lists by sending e-mail to the user (struts-user-subscribe@jakarta.apache.org) or developer (struts-dev-subscribe@jakarta.apache.org) mailing lists. The user mailing list is made up of users of the Struts framework, while the dev mailing list is made up of developers interested in furthering the Struts framework.

You can contribute to Struts by providing patches and modules to the dev mailing list. There's also a "Todo" list on the Struts Web site, http://jakarta.apache.org/struts/, that outlines the immediate tasks the group is working on. Help is always welcomed.

Last, you can achieve committer status, which enables you to have direct access to the source control. To become a committer you must be voted in; the rules can be found on the Jakarta Web site, http://jakarta.apache.org/guidelines/index.html.

References and Resources

1. Gamma, E., Helm, R., Johnson, R., and Vlissides, J. (1995). *Design Patterns - Elements of Reusable Object-Oriented Software*. Addison-Wesley.
2. http://jakarta.apache.org/struts/
3. www.bluestone.com

craig.mcclanahan@eng.sun.com

lmccay@bluestone.com

bergen@bluestone.com

PREMIER ISSUE

WebSphere DEVELOPER'S JOURNAL

U.S $15.00 (CANADA $18.00)

Volume: 1 Issue: 1

websphere developersjournal.com

SYS-CON Events presents WebSphere EDGE Conference & Expo

From the Editor by Jack Martin pg. 5

Guest Editorial by Ajit Sagar pg. 7

Case Study by Robert Diamond pg. 70

Success Stories by Brady Flowers pg. 82

Tips & Techniques by Sean Rhody pg. 94

WSDJ News pg. 98

Interview by Robert Diamond pg. 104

IMHO by Jason Westra pg. 110

NEWSSTAND PLEASE DISPLAY UNTIL APRIL 30, 2001

Consolidating Legacy Data with WebSphere

written by Brady Flowers

page 8

Feature: Programming EJBs Using IBM WebSphere Application Server A how-to primer Jack Martin 22

EJB: IBM Advanced EJB and WebSphere Programming Support for any kind of e-business Brady Flowers 30

Feature: IBM WebSphere Performance Pack – Network Design Implications Finding common ground Sean Rhody 38

Tutorial: Introduction to WebSphere Application Server EE for OS/390 A means for generating traffic Ajit Sagar 46

XML: WebSphere XML Implementations Open standards-based technology makes the difference Israel Hilario 54

Feature: How to Install and Administer WebSphere Performance Pack Run on any platform Robert Diamond 62

WebSphere & EJB Corner: An Introduction High performance translations Jason Westra 74

Feature: How to Configure WebSphere in Multiplatform Environments Exploring technology Jeremy Geelan 86

DeployDirector 1.3

By Sitraka Software (Formerly KL Group)

REVIEWED BY JOE MITCHKO

AUTHOR BIO

Joe Mitchko is a principal engineer with eTime Capital, Inc., Sunnyvale, California, where he specializes in Internet architecture.

jmitchko@rcn.com

Environment:
Supported servers: Solaris, WinNT, Win2000, HpuX, AIX, Linux servers
Supported clients: Win95, Win98, WinNT, Win2000, Solaris
JRE: Version 1.1 or higher

Test Environment:
Dell Inspiron 3800, 256M RAM, 18G H/D, Windows 2000 Professional (Service Pack 1)

Sitraka Software
260 King Street East
Toronto, Ontario
Canada M5A 4L5
Phone: 416 643-3539
Web: www.sitraka.com
E-mail: direct@sitraka.com
Evaluation copy available

DeployDirector provides a comprehensive environment for maintaining Java applications across the enterprise. It consists of several modules, including a server-based repository, an administrative interface, and a client-side agent. The administrative tool allows you to configure and deploy new or updated versions of an application to thousands of client desktops across the enterprise.

A Java applet performs installs and updates, obtaining all the necessary components from a server-based repository. For installs, the applet is invoked from a Web page while the application automatically calls for updates.

DeployDirector stores the installation and update information for one or more applications in a centralized storage facility called a *vault*. The vault is maintained through a GUI-based administration utility that allows you to add or change applications that are to be deployed. Multiple versions of an application can be stored in the repository along with the various rules governing how they are to be installed or updated.

Applications installed with DeployDirector are front-ended with an additional piece of code that senses when an update is available. The Java application start-up mechanism, called the Client-Side Application Manager (CAM), automatically checks for updates every time an application is invoked. If an update is available, the CAM will prompt the user to update the application when a new version is available. Similar behavior can be found on software products such as RealNetworks audio player, which notifies you when a new version is available and downloads it off the Internet.

In addition to Java applications, DeployDirector handles Java applets as well. In this case, DeployDirector maintains a client-side image of the applet, thus eliminating the need for the Web server to download the entire applet every time a page is accessed. An additional optimization feature reduces network traffic by eliminating the need to constantly redeploy large JAR files every time a change is made. Instead, DeployDirector will download only class files that have been modified and make the appropriate changes to the JAR file on the client side.

DeployDirector is designed to update client applications from anywhere there's network access. A client application can be updated from within the enterprise (behind the firewall) or outside the enterprise through the Internet.

The DeployDirector Servlet can run in a single stand-alone server or within a cluster. For large multinational corporations, DeployDirector servers can be deployed at several regional data centers serving thousands of users. On the client side, DeployDirector maintains a minimal footprint making it somewhat transparent to the user. It also maintains the relationship of an application to the correct version of its runtime JRE while keeping multiple JRE versions transparently. This capability takes a lot of the guesswork out of deploying multiple JRE versions on a client machine.

Installation and Configuration

The install CD comes with two Zip files, one platform-specific and one that's common for all environments. The platform-specific Zip files are optional, depending on whether the target server contains a preloaded JRE (and correct release level). In my case I was running on Windows 2000 and opted to extract both Zip files. Both files are extracted to the same DeployDirector home directory. The Windows platform Zip file (Win32.zip) contained the Java runtime environment and several DLL and OCX files.

After invoking the start-up script, the DeployDirector server started right up and promptly did nothing. At this point you need to go to the administration page by pointing your browser to the server, using a dedicated port setting. To continue, enter the licensing keys provided via e-mail (one of those postregistration deals). This part of the installation was relatively painless. Now you're on an administrative Web page containing vari-

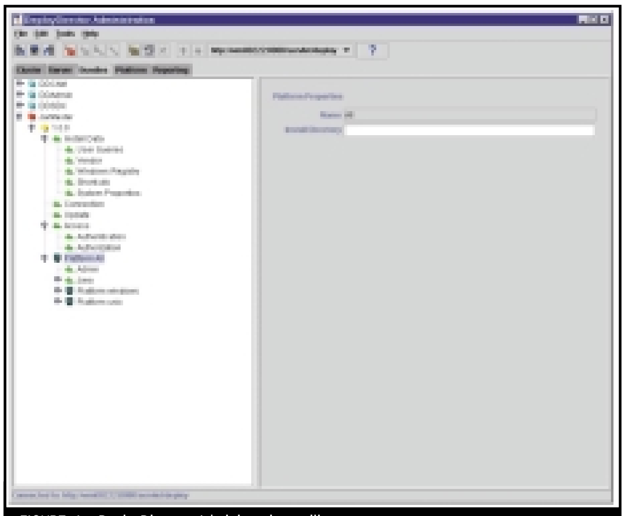


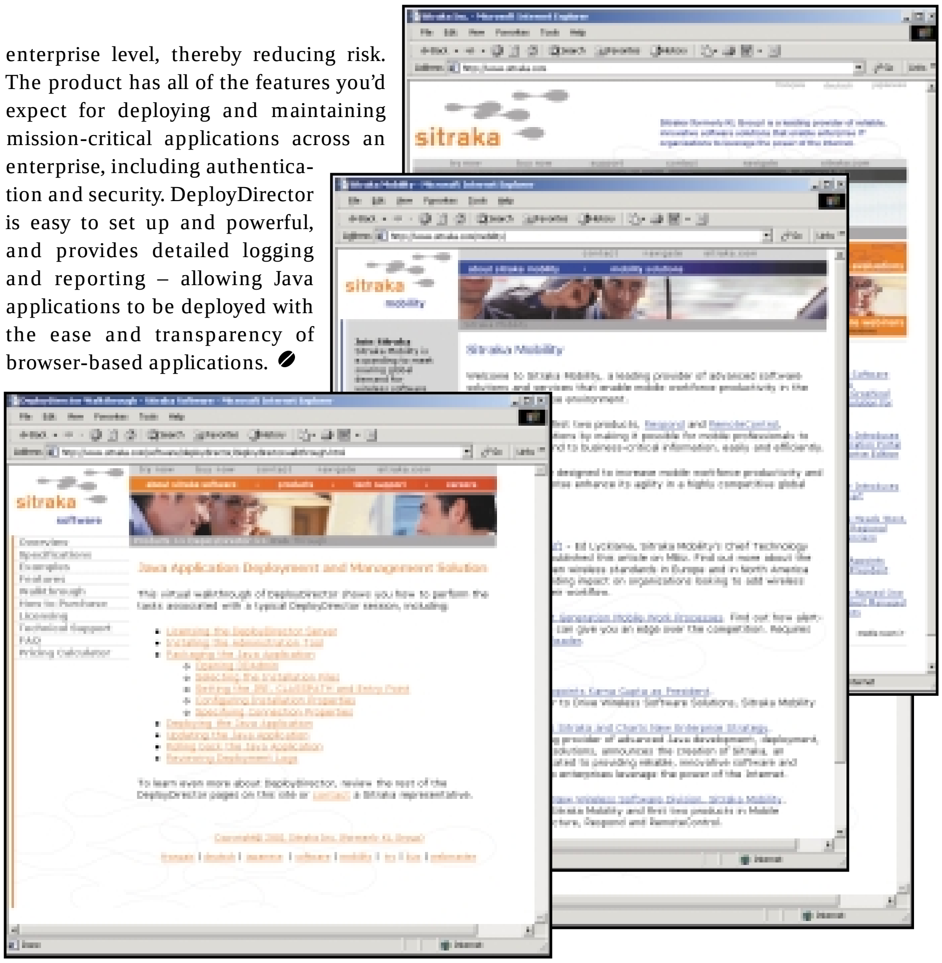
FIGURE 1 DeployDirector Administration utility

ous options and installation details. Now comes the interesting part. So far, no administrative tools have been installed. They have yet to be deployed to your machine. This is where DeployDirector shows off its capabilities. DeployDirector's first task is to deploy the administration application to your machine. It comes preloaded with bundles containing the administration applications (neat stuff). If you scroll down in the admin page, you'll see links allowing you to install the admin application. Once selected, a Java applet takes over and guides you through the standard setup process. For instance, the setup program will ask you for the install directory, and about placing an icon on your desktop and other setup-related questions. Once installed, it was easy to get to the administration application through the Windows start menu (see Figure 1).

Now you're ready to set up application bundles. The installation comes with a demo application and a tutorial that guides you through the process of setting up your deployment applications. I'd recommend going through it.

Summary

DeployDirector seems quite ready to take on the task of maintaining applications across any enterprise. Using clustered services, corporations can gradually scale their implementation of DeployDirector from the departmental to the enterprise level, thereby reducing risk. The product has all of the features you'd expect for deploying and maintaining mission-critical applications across an enterprise, including authentication and security. DeployDirector is easy to set up and powerful, and provides detailed logging and reporting – allowing Java applications to be deployed with the ease and transparency of browser-based applications.



Java Application Deployment and Management Solution

This virtual walkthrough of DeployDirector shows you how to perform the tasks associated with a typical DeployDirector session, including:

- Licensing the DeployDirector Server
- Installing the Administration Tool
- Packaging the Java Application
  - Opening DDAdmin
  - Selecting the Installation Files
  - Setting the JRE, CLASSPATH and Entry Point
  - Configuring Installation Properties
  - Specifying Connection Properties
- Deploying the Java Application
- Updating the Java Application
- Rolling back the Java Application
- Reviewing Deployment Logs

To learn even more about DeployDirector, review the rest of the DeployDirector pages on this site or contact a Sitraka representative.

Advanced Source Control in VisualAge for Java

Enjoy the benefits of your SCM system

Part 1 of 2

WRITTEN BY
TIM DEBOER,
STEVE FRANCISCO &
ELLEN MCKAY

For software developers source code is more than just files on a disk – it's the result of hours of thought and work and must be protected. If you look at the number of software tools available to help developers track and store their source code, you'll see that code protection is a key part of any serious software development effort.

A software configuration management (SCM) tool is as important as a compiler or debugger, but as any experienced developer can tell you, getting your favorite tools to work together isn't always easy – especially when they're from different companies.

VisualAge for Java version 3.5 includes an External Version Control (EVC) tool that interacts with external SCM systems. Each project in VisualAge for Java can be associated with an SCM system (called "adding a project to version control") and a specific group of files in the SCM system's source code. Once you've added a project to version control, you can add classes to your SCM systems, check classes in and out of the system, and import the most recently checked-in version of a class from the SCM system into the Integrated Development Environment (IDE).

In Part 1 we discuss the following:

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

In Part 2 we'll continue with a discussion on path prefixes and refreshing your SCM files in the IDE.

Using the Team Server Instead of the EVC Tool

VisualAge for Java, Enterprise Edition, includes a team server (EMSRV) that enables you to share your code with other developers through a shared repository. One of the biggest decisions a development team needs to make is how to manage its source code: Should it use the team server, the EVC tool, or both?

The VisualAge for Java repository is inseparable from the IDE and is at the heart of most functions offered by the product. However, developers don't need to learn its SCM role if they're using another SCM tool. If you prefer a third-party SCM tool, you can use a local repository instead of the shared repository.

There are certain advantages to using only a local repository and the EVC tool:

- Less administration for the team (no team server required)
- No confusion about the location of the source code's most current version
- No confusion about source management processes
- More mobility since users can work disconnected from the network
- Less training for new developers since there's only one paradigm to learn

However, there are several reasons why developers may also wish to use a shared repository:

- Developers working on multiple teams may need to access both versioning systems.
- Code libraries may be more efficiently stored on a common server.
- Access to code previously stored in a repository might be necessary.
- Personal preference.
- Easy code-sharing capabilities.

There's no single right answer. The decision of which method to use depends on the characteristics and requirements of your development team.

The remainder of this article assumes that an external SCM tool will be used by the development team, and discusses how VisualAge for Java's EVC tooling bridges the gap.

Automatic Versioning of Your SCM Files

Every time you use the EVC tool in the VisualAge for Java IDE, the SCM files you're working with are automatically versioned, and new copies of those files are created in the VisualAge for Java repository. Also, before actions such as refreshing and checking out are performed, all the SCM files that are being modified are automatically versioned and named.

Versioned copies of your code can be very useful. If, for example, you refresh your project and accidentally replace a file you've changed with another unchanged version from your SCM system, you can simply replace the new version in the workspace with the older version from the repository using the Replace With > Another Edition menu item.

Although developers using the tool don't need to worry about the internal versioning that occurs, the tool itself relies on this information to track changes and determine courses of action during processing.

Merging Changes

Checking out source code before making changes is good programming practice when working with most SCM tools that are accessible via EVC. It lets teammates know you're modifying a file so they don't try to make simultaneous changes. Unfortunately, this rule is often broken. For example, a developer who needs to fix a bug while flying to a customer site doesn't typically have a connection to the LAN to check out the file before making changes. Whatever the reason, the bottom line is that conflicts will happen and merging will be necessary.

You can modify the source code for a project associated with an external SCM tool even if the code has not been checked out yet. The edits are allowed and you're reminded that you didn't check out the source yet. In addition, the status icon beside the filename shows a warning symbol. If it's necessary to check in the changes later, you can check out the file, resolve any conflicts, and then check in the code.

You can merge the code of an older version of a file with the code contained in a newer version. For example, you can import a file into the IDE without checking it out, then edit it and save your changes. If you then decide you want to check the file out from your SCM system but don't want to lose your changes, simply check out the file, compare the new and old versions, and merge any changes.

If you check out a file that someone else has changed, the difference is detected and you're asked how you want to proceed. You can either ignore the changes, leaving your source in the workspace, or you can ask to load the most recent version. This is where the automatic versioning comes in handy. Loading the new code will replace your changes, but they're safely versioned in the repository. Using the IDE's "Compare With" functionality, you can bring your modifications into the current source code and check in a properly merged file without accidentally deleting your co-worker's efforts.

Working with Class Groups and Resource Files

In the VisualAge for Java IDE you can elect to work with files on an individual or group basis. For example, if you wish to check in only a few files for a project, select those files and check them individually into your SCM system. If, however, you wish to check in all the files contained in a particular project or package, select the project or package and check in all the contained interfaces and classes at the same time.

However, any resource files associated with the selected project or package won't be checked in. Since resource files are not actually contained in projects or packages (they're only associated with them), any SCM actions you perform at a project or package level won't affect them. Instead, you must work with the files separately in the Resources page of the Workbench or Project browser.

Conclusion

Managing source code is an essential part of the development cycle, and there are many configuration management systems available to help you store and manage your code. However, because there are so many systems available, developers must not only decide which tool is best for them but also how that tool can work with their other development tools.

The EVC tool in VisualAge for Java version 3.5 enables you to manage your source code from within the IDE and work with many external configuration management systems. This allows you to enjoy the benefits of your SCM system while still taking advantage of VisualAge for Java's powerful development environment.

In Part 2 we'll continue this topic with an article on path prefixes and refreshing your SCM files in the IDE.

Author Bios

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

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

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

deboer@ca.ibm.com

cisco@ca.ibm.com

ecmckay@ca.ibm.com

INTERVIEW WITH Anne Manes

J D J I N T E R V I E W

Director of Business Strategy, Sun Microsystems

An Interview by Alan Williamson

PREAMBLE:

On February 5 Scott McNealy announced to the world's press Sun's new strategy for building open Web services using open and industry standard technologies under the umbrella name of Sun ONE. ***JDJ***'s editor-in-chief got a chance to speak to one of Sun ONE's key architects, Anne Manes.

<alan>: **Sun ONE: What's it all about? Give us the 60-second elevator pitch please, Anne.**

<anne>: Okay. Well, in a single line, Sun Open Network Environment (ONE) is basically an architecture to support Web services and it is also Sun's implementation of that architecture. So we have two different things: we have the basic architecture that is based on open technologies and we have a set of iPlanet products and a few Sun products thrown in to round it all out which provide a complete implementation of that architecture.

<alan>: **Reading between the lines, it's very hard to see anything new in the announcement. Sun, with *JDJ*, has been heralding these open standards for years. So why now?**

<anne>: The big difference in what we said before and what we're saying now is that we've crystallized and brought everything together in terms of Web services, because Web services seem to be the next new thing. We've got all these folks scribbling around, saying, well, we know how to build Web applications but how do I go about building Web services. And this is more than just J2EE, which is the platform for building Web applications; this is J2EE plus XML plus a number of other activities that are going on in the broad spectrum of development among consortiums, and things like that, that are putting together everything that is necessary to build the Web services infrastructure. So, in addition to things such as J2EE servlets, JSP, and things that are necessary to build the Web application, we are also endorsing things like UDDI, which is a directory for finding businesses and the services they provide; WSDL, which is a way of describing those services as open standard XML protocol; and eXML, which is a full-fledged B2B e-commerce framework for actually doing international trade.

<alan>: **What is your definition of Web services? Every company seems to have a different variation on this. What is Sun's definition?**

<anne>: Let me describe first what I mean by services. Sun has been promoting service-driven computing for a very long time, starting back with things like RPC, GENIE, and JNI. We've been talking about this for about four years now.

"Borland recognizes that the Sun ONE architecture formalizes Sun's complete vision for enterprise solutions, bringing together a comprehensive set of complementary technologies. Perhaps the most important aspect of this architecture is Sun's commitment to interoperability through open standards. Customers can take advantage of Sun ONE's platform and still integrate industry-leading technologies from other vendors, such as Borland Application Server, Borland JBuilder, and AppCenter, to give them a unique, competitive advantage."

—Tony De la Lama
VP and General Manager, Java Business Group
Borland

Borland

Basically, it's some sort of content that you need to access via some sort of network. A Web service is a specific type of service that is specifically suited for working across the Internet, and across the Web in particular, over HTTP, and it is built using XML, and the service is registered in a Web service directory as opposed to a service-specific registry. Basically, the difference is the type of protocol you use to communicate with it and the type of registry it gets to register in it.

<alan>: **With respect to the overall strategy, and I refer to Scott McNealy's Sun ONE address, isn't Sun ONE much bigger than just the Web?**

<anne>: Yes. One of the purposes of Sun ONE is that it extends beyond that of the Web services. The primary focus of Sun ONE today is to enable Web services because that is what everybody is attempting to do today. But Sun ONE is extensible to support other types of services, including wireless services and services in your home and services in all different types of devices. Our goal is to have all these different types of services completely interoperable, and seamless integration.

<alan>: **We read continually that this isn't a knee-jerk reaction to Microsoft's .NET strategy. However, every single speaker in the 2.5 hour presentation went to great pains to keep drawing parallels to Microsoft. Surely, if Sun ONE is so different from .NET, what sense is there in continually mentioning Microsoft?**

<anne>: <giggles> Primarily, they're the only competitor we see in the space because everyone else who is developing Web services is based on Java and XML and it is only Microsoft that is not based on Java and XML. Therefore it's appropriate for us to compare our approach to Microsoft's approach. Microsoft's approach is very much Windows bound; everything they've developed through .NET is based on Windows and Microsoft technologies that are closed and force you to stay within their platform. Anything we've described with what we are doing with Sun ONE can be implemented by any vendor in the world. We have an implementation, but it's not the only implementation out there. In fact, we believe IBM could easily

"Sitraka Software views the announcement of Sun ONE as a true milestone for Webtop technologies. Our enthusiasm for this important initiative is driven by our own commitment to developing enterprise solutions based on standards like XML and the Java Platform. In Sun ONE we see a tremendous opportunity to advance enterprise Java development and empower smart Web services by providing scalable, high-performance tools for development and deployment that integrate with the Forte development environment and iPlanet application servers."

—Larry Humphries
President, Sitraka Software

Sitraka

announce tomorrow an implementation based on their WebSphere offering. .NET is the only alternative to Sun ONE right now.

<alan>: **Sun ONE…JavaONE. Seems too much of a coincidence there. Are we going to start seeing another Sun revolution once a year in the heart of San Francisco?**
<anne>: <smiles> We haven't made any decisions along those lines.

<alan>: **This drive for openness is admirable and something the industry has been needing to hear for years. However, surely the Sun ONE architecture should only define the standards. Are you concerned that your "no-vendor-lock-in" message will become diluted with the continual references to your own products, such as iPlanet and Forte?**
<anne>: Sun ONE, first of all, is open architecture, and that is something anyone can implement and that is based on Java and XML. The Sun ONE implementation implements that architecture.

<alan>: **Drawing a parallel here with the J2EE standard for a moment, is Sun ONE going to be a standard that is going to be versionable?**
<anne>: We don't currently have any plans in place to do any conformance testing. Sun ONE is really only a set of guidelines. It doesn't say you must use every single one of these standards. We are planning on coming out with a number of developer aids, such as design patterns and best practices, sample programs, positioning papers, and other things that will basically help you build open, interoperable Web services.

<alan>: **So this is an evolutionary process, and so far we are only seeing the first step?**
<anne>: Exactly.

<alan>: **With Java taking a pivotal role within the Sun ONE strategy, what can we expect to hear in June at JavaONE?**
<anne>: You should expect to hear more about Sun ONE and where we are up to in terms of a status report.

<alan>: **Scott McNealy is now infamous for his Microsoft bashing. He seems to have influenced all the key Sun executives, as all their speeches have some derogatory remark for Microsoft. Do you think this is detracting from Sun's core message for a greater and more open industry?**
<anne>: Well, our core message is that you should be building solutions based on open technologies, not closed technologies. We don't want people to be swayed by Microsoft's continual claim for openness, 'cause we don't believe it's really open. But with respect to Scott, I can't speak for him…that's Scott!

To listen to the complete presentation by Scott McNealy, visit www.sun.com/webcast/software2001/.

"The first generation of the Internet was about human end users accessing information and applications through a browser. The next wave of Internet technology will make Web-based application and information services available to developers to use in the programs they build. Web services frameworks based on Java and XML such as Oracle9*i* Dynamic Services make it possible to deliver on this vision today."

—John Magee
Senior. Director of Marketing
Oracle9i Application Server
Oracle World Wide Marketing

Oracle

Oracle9*i* Dynamic Services

The Internet has changed the way businesses and consumers interact. By providing personalized, self-service Web applications that are available at any time, anywhere in the world, businesses have reduced costs while simultaneously offering customers faster, more accurate, and more responsive service. The next step in the evolution of the Internet will be to bring businesses closer together by enabling companies to exchange information and business processes automatically, in real time, over the Web. Web services will allow business partners to integrate their business processes, increasing efficiencies and reducing costs. For example, using Web services, one company's purchasing application could interact directly with another company's ordering application, with minimal manual intervention – speeding the transaction and reducing the risk of human error.

Oracle delivers on this vision with Oracle9*i* Dynamic Services, an extensible, programmatic framework that enables businesses to work more flexibly and efficiently with partners, suppliers, distributors, and other third parties by easily incorporating information, application functionality, or business processes in real time over the Internet. By providing a complete infrastructure for accessing, managing, executing, and delivering Web services, Oracle allows companies to leverage the expertise and resources of partners while remaining focused on their own core business strengths. With these services, applications can be dynamically configured and updated from services discovered in real time.

Because Oracle9*i* Dynamic Services isolates developers from service implementation and protocol details, applications can be developed more quickly and don't need rewriting when business partnerships or underlying technologies change.

Standards-Based, Open, Flexible

Oracle9*i* Dynamic Services leverage existing industry standards – Java, LDAP, and the latest XML specifications published by the World Wide Web Consortium. They will also support emerging protocols like Universal Description, Discovery and Integration (UDDI), SOAP, and Web Services Definition Language (WSDL) as soon as they are finalized and ratified by the W3C. The modular design of the Oracle9*i* Dynamic Services engine makes it easy to add protocol adapters to support new and evolving protocols. The same open architecture that allows these services to adapt quickly to evolving standards also makes it easy for businesses to customize the framework to meet their specific needs while still maintaining compatibility with external Web services, regardless of protocol or platform. In addition, Oracle9*i* Dynamic Services allows developers to create and deploy Web services on any of the more than 40 operating systems that Oracle runs on, including all major releases of Linux, UNIX, and Microsoft Windows.

Robust Web Service Execution and Management Environment

An effective Web service strategy involves more than just protocols. Developers need a way to easily customize the service execution flow by aggregating services and including custom business logic. Administrators must be able to manage services, track performance, and be notified of failures. Services must be able to be cataloged in a centralized, searchable repository, and be deployed in a scalable and reliable manner.

Oracle9*i* Dynamic Services leverages the power of the Oracle9*i* database to catalog, manage, and audit services, and the Oracle9*i* Application Server to deploy scalable, high-performance, and highly available Web services. It provides features to discover and bind services at runtime, combine existing services into a customized execution flow, and deliver the same service to multiple channels and devices. It also offers features to manage and monitor services, automatically failover to alternate services, and perform auditing (e.g., to track usage patterns, develop customer profiles, and bill clients).

The current production release is available for download at http://otn.oracle.com.

—Rob Cheng
Senior Product Marketing Analyst
Oracle9i
Oracle World Wide Marketing

SiteSticky 4.0 by NetDIVE Inc.

REVIEWED BY JOE MITCHKO

AUTHOR BIO

Joe Mitchko is a principal engineer with eTime Capital, Inc., Sunnyvale, California, where he specializes in Internet architecture.

jmitchko@rcn.com

JAVA DEVELOPER'S JOURNAL JDJ WORLD CLASS AWARD

SiteSticky 4.0
NetDIVE Inc.
353 Kearney Street, Suite 4
San Francisco, CA 94108

Phone: 415 981-4545
Fax: 415 981-4546
Web: www.netdive.com
E-mail: sales@netdive.com

Test Environment: Dell Inspiron 3800, Windows 2000, Pentium III whitebox system running on Windows NT4.0, Pentium II 233 whitebox system with 128M RAM, NT Server 4.0, Microsoft's IIS 4.0

Pricing: Enterprise software starts at $99.95 per concurrent user with a minimum of 100 users, and an annual subscription fee of $1,999. For software as small business services, it's $199 (and up) per month .

In the increasingly fast-paced world of the Internet, myriad new Web sites seem to arrive daily, intent on gaining market share. Whether your site is involved in business-to-consumer sales, online services, or portal services, the name of the game is the same: try at all costs to retain users and give them the best customer service experience.

Keeping content up to date and constantly adding new features help, but require technical resources and time. Other methods include developing a community of users with a common interest and getting them to hang out at your site. The term *stickiness* describes how well a particular site attracts new users and keeps existing ones coming back.

NetDIVE provides a number of products that add community to your site and give your Web site customers the highest-quality service. Their products for the community building purpose are SiteSticky and WeMessage; for customer service it's CallSite, and for communication and messaging they are eAuditorium and WeMeeting. They're all designed to plug easily into your site.

One thing particularly nice about NetDIVE products is the customization capability. You can completely change the look and feel of their user interface. All products provide complete localization capabilities as well, so you can choose several user interface variations, allowing you to switch among several languages, including Spanish, French, and Arabic. The internationalization features of the products appear to work well as evidenced by the significant global interest.

SiteSticky is NetDIVE's application for Web-based chat and community building. It has two components: a Java application service that runs on your Web server and a Java applet that can be embedded in one or more pages on your site. There can be additional components involved in the Enterprise class version to enable such things as full cobrowsing and the ability to tunnel through firewalls.

Providing an applet-based product for third-party use on other Web sites isn't an easy task. To stay compatible with all Java-enabled Web browsers, SiteSticky was developed based on the Java 1.0.2 API. Our tests show that SiteSticky ran in various versions of Internet Explorer and Netscape browsers without any compatibility issues.

WeMessage Portal is an instant messaging software system similar in nature to AOL's Instant Messenger, for large portals that want to offer instant messaging services to their users. It has lots of features and, best of all, it's written in Java.

CallSite is a Web-based customer service and call center application that allows visitors to a Web site to click a button for instant multimedia communication with the agents of the company. It can be used for a variety of purposes, including interactive online technical support and assisting users making online purchases. As a developer, you'll find CallSite particularly powerful for integrating your Internet-based customers with your sales, shipping, suppliers, and partners due to its completely open standard–based design. In general, NetDIVE has done a good job of integrating their products.

Another unique feature of NetDIVE applications is their patent-pending voice communications module. This allows you to instantly engage in voice communication in two modes – hands free and CB mode (where you're required to keep a button pressed down to talk).

Installation

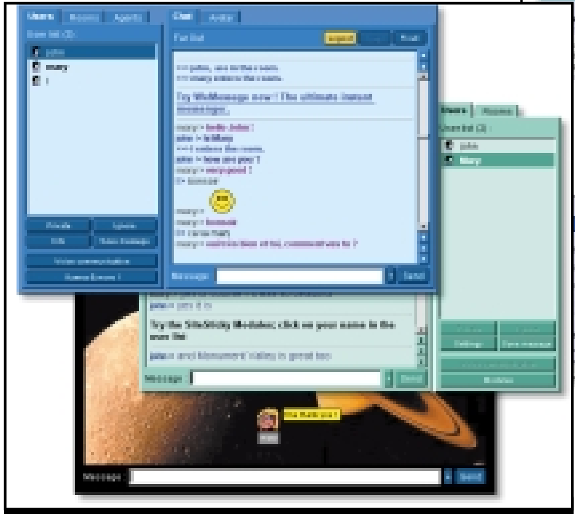
Installing NetDIVE enterprise class products was straightforward, but required some technical skill, including familiarity with the target Web server. The installation instructions did a good job of guiding me through the process. As always, it helps to read the instructions first (I know it's hard). To run the SiteSticky, WeMessage, or CallSite service, you need to have a JVM installed on your Web server with the proper path and classpath settings in place. NetDIVE recommends JVM version 1.2, although they claim the product is compatible with version 1.1. I used version 1.2 from Sun. You can also use Microsoft's Win32 JVM. There are several ways to launch the service, depending on what version of JVM you're using.

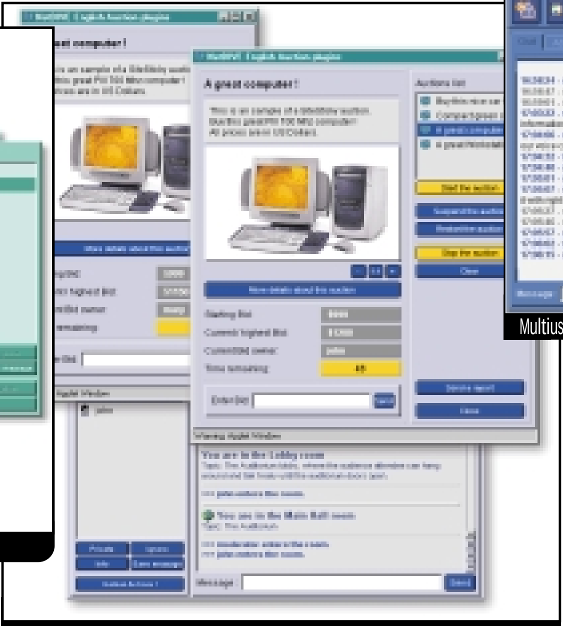
Scripts are provided for quick start-ups in your environment. Although starting the services on my NT 4.0 server was easy, it required bringing up the DOS command prompt in the foreground. For production situations it would be better to have the servers run as NT services instead. Also, with firewalls everywhere and more restrictions every day, NetDIVE offers the ability to run all their software systems as servlets using port 80 instead. This requires you to install a servlet engine on your Web server, which may complicate the overall installation process depending on the Web server you're using. You'll also need to adjust the applet parameters at your site to point to the servlet. I'd suggest making the right choice (firewall-proof versus nonfirewall-proof) ahead of time so you don't have to go back and change every applet parameter in your HTML.

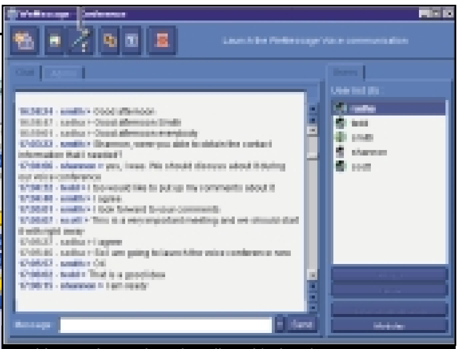
Test Drive

You can try NetDIVE software systems by going to the NetDIVE Web site (www.netdive.com) and clicking on the Try Now button.

I couldn't pass up trying out the cobrowsing capabilities of NetDIVE products. This feature allows two or more users to browse the Web together as if they were looking at the same


SiteSticky client-side interface


eAuditorium live auction client-side interface


Multiuser voice conferencing client-side interface

browser. For instance, you can shop at an online store with someone who may be on another machine halfway across the country. The possibilities of cobrowsing while engaging in voice communication are endless. Besides extending the online shopping experience (SiteSticky), this could be a boon for customer service reps (CallSite).

Programming

Now for the good stuff! The first things I often ask when looking at a product like this is how easy is it to program and what technical expertise do I need. First, NetDIVE apps are based on complete open standards, so on both the client and server side they're extremely programmable and easy to integrate into other components (such as SQL databases).

You can also control what API plug-ins can be used, as well as other control settings. The challenge from a Java programming standpoint is creating your own API plug-ins. NetDIVE apps provide you with an API base-class library, allowing you to create your own plug-ins with relative ease. You just need to stick to the interface provided. A reusable class library makes it even easier to program new features.

Documentation

The documentation provided with NetDIVE apps is complete but not extensive, and could use some grammatical improvements as well. For instance, in several cases I had to read a sentence a number of times to figure it out. Generally, however, I found it technically correct and to the point.

However, NetDIVE provides complete installation of all its Enterprise systems (remotely via TelNet), so you can always have the NetDIVE engineers install the whole system for you and not bother with the online documentation.

BOOK REVIEW

JAVA MESSAGE SERVICE

BY RICHARD MONSON-HAEFEL & DAVID CHAPPELL

PUBLISHED BY O'REILLY

REVIEWED BY JAMES MCGOVERN

View of JMS from the Inside

This is my first book review for ***Java Developer's Journal***. As coauthor of a competing book, I figured I'd be very critical of the author's writing and the book's content. I must admit I'm pleasantly surprised; this is a really good book and, in many ways, better than the book I coauthored.

One of my biggest pet peeves is to walk into a bookstore and see overpriced 1,000-plus page books that are essentially a regurgitation of the manual. Even worse is to see these large books essentially reprint free documentation and expect consumers to pay for it. This book does none of that. Its page count is a healthy 220 pages and it's written in a very easy-to-read style. I went to Amazon to see how competing books are priced, and this is the least expensive. I recommend that everyone interested in JMS pick up a copy.

This book is not written by two guys who got paid by a publisher to rewrite their interpretation of the existing specifications. The two authors, Richard Monson-Haefel and David Chappell, are known in the industry and were actually part of the organizations that contributed to the JMS specification. This gives them great insight about the real intent of JMS. They do a fine job of explaining concepts and subtleties of many of the APIs. One good example is their explanation of when and when not to use the TopicRequestor object.

If you have no prior knowledge of JMS and its purpose, the first chapter gives a complete description of the Java messaging paradigm. Chapter 2 gets into a tutorial about the JMS APIs.

O'REILLY

Hot off the Press!

News

O'REILLY Online Catalog

Java Message Service

Chapters 3 through 6 cover publish and subscribe messaging, point-to-point messaging, guaranteeing messaging, transactions, acknowledgment, and failures.

Chapter 7 is my favorite, as many books on the market target only developers. This chapter is also useful for architects who need to figure out the right way to design, develop, and deploy Java-based messaging solutions. It covers just the right amount of information on performance, security, and when multicasting versus hub and spoke is appropriate.

Chapter 8 goes into detail on J2EE, EJB and JMS, and describes how JMS can be used with MessageDrivenBeans. I haven't seen this covered in any other book to-date. I suspect that using JMS with EJBs will be a popular method in the future for application development.

No book review would be complete without pointing out a couple of flaws. First, JMS in its next revision will support XML messaging. There are actually several competing standards in progress that could be used with JMS. Maybe a mention of the Java API for parsing (JAXP) and the Java API for XML messaging (JAXM) would have been appropriate. The authors only mention that the future may bring an XMLMessage type, but today stick with the TextMessage type. I found this curious because one of the authors, David Chappell, works for Progress and their product, SonicMQ, actually has XML support built in. I'm not really sure if the author was keeping himself honest in this regard, since he does plug his product elsewhere in the book in subtle ways.

My only other thought is that in Chapter 9, the authors list the different JMS providers but make no mention of a very popular Message Queue product, Microsoft's MSMQ. All of us Java types have a bias toward Microsoft, but they really do have a good product that's worth considering as part of your solution.

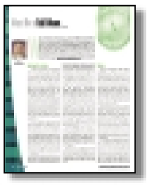
Overall, this book is a good source for those who want to know the ins and outs of utilizing message-oriented middleware using JMS. *Java Message Service* does an admirable job of providing examples that are adaptable to many situations. The authors made the topic understandable. On a scale of 1 to 10, I give this book a 9.5. *Java Message Service* is an excellent book and an essential item for your library.

jmcgovern@enherent.com

AUTHOR BIO

James McGovern is a senior technical architect for Enherent Corporation in their software development center in Windsor, Connecticut, where his focus is on strategy and architecture for high-profile e-business Web sites. He is also a coauthor of several Java-related books.

Letters to the Editor



I just want to wish all the *JDJ* staff a Happy New Year! More power to your magazine and we do enjoy reading Alan Williamson's monthly column, **Industry Watch**.

–Liza
l.aquino@pacificanalyst.com.ph]

"An Introduction to Genetic Algorithms in Java" by Michael Lacy (***JDJ***, Vol. 6, issue 1) was a very interesting article. However, I do take exception with one premise. He states, "...nature accomplished this extraordinary programming feat without a single developer coding." There was indeed a Developer responsible for this "feat." Again, using Michael's own words, "One look at the genetic code of even the simplest living organism reveals a structure that's enormously complex and efficiently tuned, ensuring the survival of the

organism in its environment." This statement suggests the necessity of a Creator. I think it's foolish to presume that this level of complexity happened by random chance. This thought is based on the "theory" of evolution and is wholly unproven. If it's to be used in an argument, it should be referenced as theory, not fact.

–Tony Moe
TMoe@SystemAutomation.Com

At Christmas Kids Day where I work, this picture of my 18-month-old daughter was taken by a co-worker. It was really not a setup, she just sat down and decided to read your magazine!!

As her father and I are both Java developers, we found it really funny that at that age she already wants to learn Java!!

–Sophie Lamothe
sophie.lamothe@bnpparibas.com

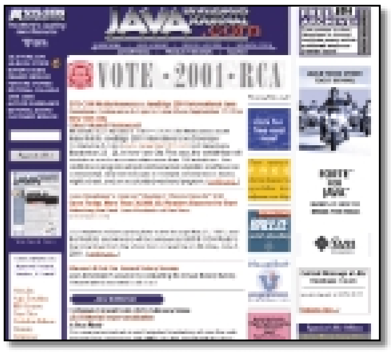
March **2001**

What's Online This Month...

JavaDevelopersJournal.com

www.javadevelopersjournal.com is your source for industry events and happenings. Check in every day for up-to-the-minute news events and developments, and be the first to know what's going on in the industry.

Participate in our daily Live Poll and let your opinion be heard.



JavaEdge 2001 International Java Developer Conference & Expo September 23–26 , New York City

Go to www.JavaEdge2001.com for daily updates and news about JavaEdge 2001, the largest Java developer conference and expo ever on the East Coast. Sponsored by **SYS-CON Events**, the conference program will present exclusive keynotes and focus on cutting-edge Java technologies in multiple simultaneous tracks, night school, and an accelerated weekend program. Alan Williamson is the conference tech chair.

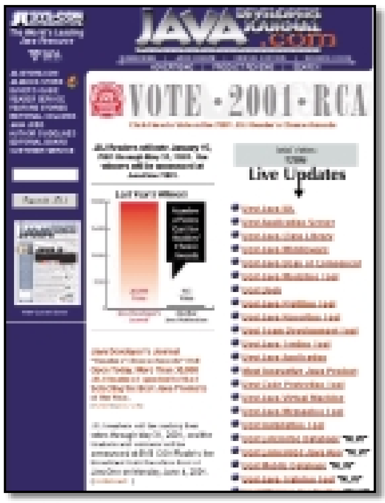
Salary Survey

Participate in our annual job/salary survey! We'd like to know the number of Java employees in your organization, the city and state you work in, what your job function is, how many years you've been in your field, and more.

The information you provide doesn't require any name or personal information and thus remains 100% anonymous. Results will be posted on our Web site and published in a future issue of *JDJ*.

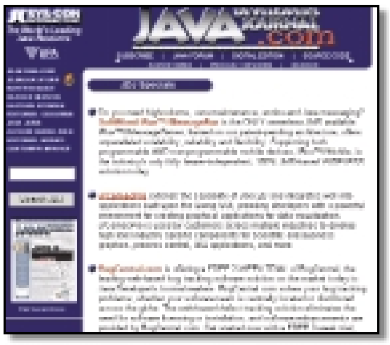
JDJ Readers' Choice Awards

Vote for your favorite Java software, books, and services in our annual *JDJ* Readers' Choice Awards, January 10 through May 30, 2001. Winners will be announced at JavaOne 2001 and presented at the **JavaEdge 2001 International Java Developer Conference & Expo**.



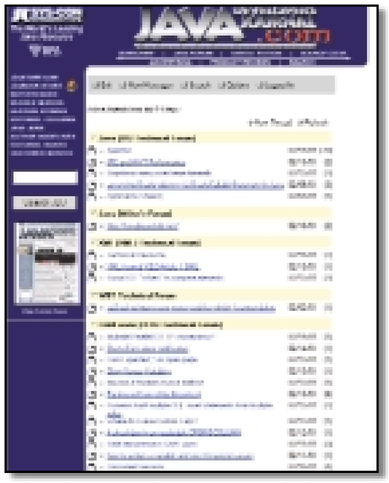
Special Money Saving Offers

Click here for free downloads from the industry's leading Java vendors.



JavaDevelopersJournal.com Developer Forums

Join our new Java mailing list community. You and other IT professionals, industry gurus, and ***Java Developer's Journal*** writers can engage in Java discussions, ask technical questions, talk to vendors, find Java jobs, and more. Voice your opinions and assessments on topical issues, or hear what others have to say. Monitor the pulse of the Java industry!

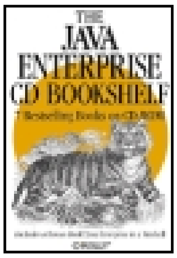


JDJ ADVERTISER INDEX

ADVERTISER	URL	PHONE	PAGE
Adrenaline	www.adrenaline.com	877-890-9280	70
Allaire	www.allaire.com/download	888-939-2545	33
Amazon.com	www.amazon.com		31
Artistic Systems	www.artisticsystems.com	336-668-4106	13
Career Opportunities Section		201-802-3028	101-104
Compoze Software	www.compoze.com	866-COMPOZE	49
Compuware	www.compuware.com/numega	800-4-NUMEGA	39
Corda Technologies	www.corda.com	888-763-0517	75
CTIA Wireless	www.ctiashow.com		87
Dice.com	www.dice.com	515-280-1144	4
Elixir Technology	www.elixirtech.com/download	65 532-4300	63
Esmertec	www.esmertec.com	877-751-3420	27
Fiorano	www.fiorano.com	800-663-3621	79
Flashline, Inc.	www.flashline.com	216-861-4000	43
Hit Software	www.hitsw.com	408-345-4001	67
IAM Consulting	www.iamx.com	212-580-2700	69
ILOG	www.ilog.com/jdj	800 FOR ILOG	21
Inetsoft Technology Corp	www.inetsoftcorp.com	908-755-0200	45
Informix	www.informix.com		11
Infragistics, Inc.	www.infragistics.com	800-231-8588	14-15
Instantiations, Inc.	www.instantiations.com	800-808-3737	85
INT	www.int.com	713-975-7434	34
Intuitive Systems, Inc.	www.optimizeit.com	888-655-0055	57
JavaEdge Call For Papers	www.sys-con.com/javaedge	201-802-3069	93
JavaEdge Conference & Expo	www.sys-con.com/javaedge	201-802-3069	96-97
JDJ The Complete Works	www.jdjstore.com	888-303-JAVA	83, 102
Jinfonet Software	www.jinfonet.com	301-990-6330	37
JustComputerJobs	www.justjavajobs.com	877-905-NERD	18
LOOX Software, Inc.	www.loox.com	800-684-LOOX	25
Macromedia	www.macromedia.com/ultradev	415-252-2000	35
Netaphor	http://cyberons.com	877 NETA4SW	70
No Magic	www.magicdraw.com	303-914-8074	7
Northwoods Software Corporation	www.nwoods.com	800-226-4662	80
Parasoft	www.parasoft.com/jdj3	888-305-0041	51
PreEmptive Solutions	www.preemptive.com	800-996-4556	59
Programix	www.jthreadkit.com		24
Progress Software	http://www.sonicmq.com/jdj301.htm	800-989-3773	2
ProSyst	www.prosyst.com	866-PROSYST	73
Quadbase	www.quadbase.com	408-982-0835	74
QuickStream Software	www.quickstream.com	888-769-9898	42
Rational Software	www.rational.com/jdj2	800-728-1212	10
SD 2001	www.sdexpo.com	800-441-8826	77
SecureByDesign	www.pkcs7.com	906-487-6520	55
Silverstream Software	www.silverstream.com/userconference	800-465-5680	29
Sitraka Software	www.sitraka.com/jdj	800-361-3264	17
Sitraka Software	www.sitraka.com/j2ee-jdj	888-361-3264	40
Sitraka Software	www.sitraka.com/components	888-361-3264	71
Sitraka Software	www.sitraka.com/jdj	888-361-3264	108
Softwired	www.softwired-inc.com	41-14452370	65
Sybase	www.sybase.com	800-8-SYBASE	23
SYS-CON Custom Media	www.sys-con.com	925-244-9109	91
SYS-CON Events	www.sys-con.com	201-802-3069	102
SYS-CON Media Reprints	www.sys-con.com	201-802-3024	86
The Breeze Factor	www.breezefactor.com	435-658-3433	19
Thoughtworks	www.thoughtworks.com	888-THOUGHT	53
Tidestone Technologies	www.tidestone.com	800-884-8665	47
Togethersoft Corporation	www.togethersoft.com/jdj	919-833-5550	6
Unify Corporation	www.unifywave.com	800-GO UNIFY	61
WebSphere Developer's Journal	www.webspheredevelopersjournal.com	201-802-3020	81
Wireless Business & Technology	www.wbt2.com	201-802-3020	79
XMLEdge Conference & Expo	www.sys-con.com/xmledge.com	201-802-3069	105
Zero G	www.ZeroG.com/installs	415-512-7771	3

O'Reilly Combines 3 *JDJ* Award-Winning Java Books + 4

(*Sebastopol, CA*) – Last June three O'Reilly Java books claimed the top spots in the Best Book category of ***Java Developer's Journal***'s Readers' Choice Awards. With an overwhelming vote, *Java in a Nutshell* by David Flanagan, *Java Servlet Programming* by Jason Hunter, and *Enterprise JavaBeans* by Richard Monson-Haefel won the industry's most prestigious award. Now O'Reilly has released *The Java Enterprise CD Bookshelf*, combining the three books in an easy-to-carry, easy-to-access format.


THE JAVA ENTERPRISE CD BOOKSHELF
7 Bestselling Books on CD-ROM
O'REILLY

Four other Java classics from O'Reilly are included in the "bookshelf": *Java Security* by Scott Oaks, *Java Distributed Computing* by Jim Farley, and *Java Enterprise in a Nutshell* (electronic version and hard copy) and *Java Foundation Classes in a Nutshell* by best-selling author David Flanagan.

For more information go to www.oreilly.com/catalog/javacdbs/.

Sims Computing Releases Flux 2.0

(*Billings, MT*) – Sims Computing has released Flux, the Enterprise Job Scheduler, version 2.0. New features include queuing, scalability, holiday calendars, and audit trails. In addition, Flux 2.0 integrates with J2EE applications and supports Sybase EAServer and the ASE database.
www.simscomputing.com

New Version of Java-Based Business Rules Technology

(*San Jose, CA*) – Brokat has released Blaze Advisor Solutions Suite Version 3.2, developed by Blaze Software, a Brokat company. One of the most important new features is the application generation wizard, through which users can create reusable rule management templates.

The product provides complete support for importing classes from XML schemas and interoperation with XML documents.and is J2EE compliant.
www.brokat.com

NeuVis Introduces NeuArchitect 3.0

(*Shelton, CT*) – NeuVis, Inc., has announced the availability of NeuArchitect 3.0, cornerstone of the NeuVis I-RAD Platform. New features include UML class diagram import, database reverse engineering, JSP support for EJB environments, code construction override, message encryption. and LDAP access control.
www.neuvis.com

Vertical Sky Partners with WebGain

(*Chicago, IL*) – Vertical Sky Inc. has partnered with WebGain Inc. to provide integration between the Vertical Sky Evolution Management solution and WebGain Studio for companies that are rapidly moving their business processes and technologies to the Web.
www.VerticalSky.com

Flashline Unveils Component Manager Enterprise Edition

(*Cleveland, OH*) – Flashline.com, Inc., has come up with Flashline Component Manager 2, Enterprise Edition (CMEE), an application that enables, promotes, and measures software component reuse throughout the enterprise. CMEE includes a searchable repository, quality assurance standards templates, and a tracking and reporting system.
www.flashline.com

Instantiations Delivers Java Code Refactoring Product

(*Portland, OR*) – Instantiations has begun shipping jFactor for IBM's VisualAge Java. The new product provides developers with a set of refactoring tools that allow them to incrementally and dramatically improve the design and quality of their Java programs.

www.instantiations.com

Servertec Internet Server 1.1 Released

(*Kearny, NJ*) – Servertec has unveiled a new release of

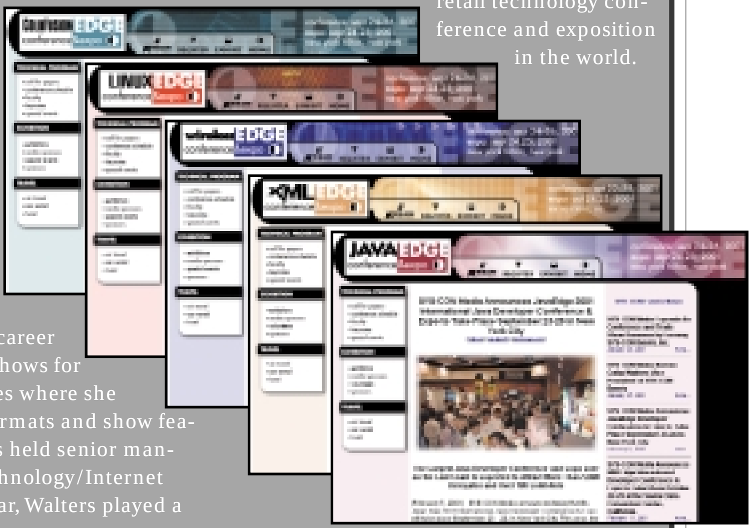
SYS-CON Media Names Cathy Walters VP of Events

(*Montvale, NJ*) – Veteran technology conference and expo industry executive Cathy Walters has joined **SYS-CON Media** as vice president of the **SYS-CON Events** division.

"We are delighted to have Cathy Walters on board," said Fuat Kircaali, founder and CEO of **SYS-CON Media**. "She will play a critical role in the expansion of **SYS-CON**'s trade show and conference business."

"I'm very excited about joining **SYS-CON,** and look forward to being part of the team that has created the most successful developer events of recent years targeting cutting-edge i-technology industries," said Walters. "I want to help the team grow the Java, XML, Linux, ColdFusion, and wireless conferences. We'll also create new developer conferences and trade shows with the same high standards."

Walters, a conference and trade-show veteran with over 20 years' experience, was formerly president of Expo Marketing & Management, Inc. She has held senior management positions at the National Retail Federation and National Expositions, Inc. The first half of her career focused on developing shows for a wide range of industries where she innovated conference formats and show features. Since 1991 she has held senior management roles in the technology/Internet marketplace. In particular, Walters played a significant role in developing Retail Systems Conference & Exposition into the leading retail technology conference and exposition in the world.



Servertec Internet Server, a small, fast, scalable, and easy-to-administer platform-independent application/Web server written entirely in Java. The update features support for file and database-based distributed session persistence and caching; custom, console, file, database, and e-mail monitor event handlers; and custom, plain, and secure socket handlers.
www.servertec.com

DevelopOnline and PointBase Form Strategic Alliance

(*New York, NY*) – PointBase is partnering with DevelopOnline to offer the PointBase 100% Pure Java object-relational database management software through DevelopOnline's Web-based open-platform development site.

DevelopOnline's site gives developers the ability to test drive the PointBase 100% Pure Java object-relational database management software on one or more open platforms. Developers can then choose the best fit for their new product design. In addition, they can sign up for development support and begin to build their new electronic device on the same site.
www.developonline.com
www.pointbase.com

EJB Testing Tool for IBM WebSphere Available

(*Waltham, MA*) – Empirix, Inc.'s Bean-test offering has been optimized for IBM WebSphere Application Server Version 3.5. Bean-test works with WebSphere to speed the development and improve the performance of Web applications that use EJB technologies.
www.empirix.com

Versant Announces Intelligent Transaction Caching Platform

(*Fremont, CA*) – Versant Corporation has launched Versant enJin, the industry's first intelligent transaction caching platform, designed to accelerate the performance of Internet-based transactions. The product integrates seamlessly with all EJB-compliant application servers, including IBM WebSphere and BEA WebLogic.

Key features include seamless persistence for EJBs and Java objects, distributed caching, automatic failover for high availability, and support for XML data.
www.versant.com

Borland Launches AppServer 4.5

(*Scotts Valley, CA*) – Borland Software Corporation has released the latest version of its application server product, Borland AppServer 4.5, with support for the latest J2EE and EJB standards. AppServer 4.5 offers advanced clustering, load balancing, and robust failover capabilities, and provides full integration between front-end development – including wireless and Web applications – and back-end systems.
www.borland.com

Sun Offers Sun ONE

(*San Francisco, CA*) – Sun Microsystems, Inc., has launched the Sun Open Net Environment (Sun ONE), a new generation of software for open, smart Web services. Sun ONE provides a complete open architecture, roadmap, and product portfolio, fulfilling Sun's seven-year Service Driven Network vision and an operational plan for simplifying how open Web services are created, assembled, and deployed across and beyond the Internet.

Sun ONE features include comprehensive product offerings to create, assemble, and deploy Web services; breakthrough and extended products, including the Sun ONE Webtop technology developer release and iPlanet Directory; Web, application, portal, commerce, and communication servers; specifics of the Sun ONE architecture, based on open standards; and an operational plan to extend Web services to the next level of intelligence.
www.sun.com

Sitraka Launches JClass ServerChart

(*Toronto, ON*) – Sitraka Software Group has introduced JClass ServerChart, a robust and scalable Java component for server-side charting. The product integrates with all major J2EE-compliant application servers, and leverages J2EE standards to ensure maximum enterprise scalability.

SYS-CON Media Announces XMLEdge Expo

(*Montvale, NJ*) – **SYS-CON Media**'s XMLEdge 2001 International XML Conference & Expo will take place October 22–25 at the Santa Clara Convention Center in Santa Clara, California. The exhibit floor will be expanded to accommodate the needs of more than 100 exhibitors. The conference program will present exclusive keynotes and focus on cutting-edge XML technologies in multiple simultaneous tracks, night school, and an accelerated weekend program.

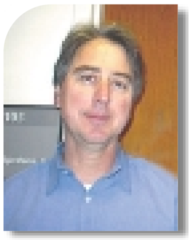
"The Conference will include two new tracks this year," according to Ajit Sagar, technical chair of the conference and editor-in-chief of ***XML-Journal***. "As producers of the leading XML developer conferences in the world for the second year in a row, we'll offer a unique learning experience not available anywhere else."

Daily updates and news about XMLEdge 2001 can be found at www.xmledge2001.com, announced Cathy Walters, vice president of **SYS-CON Events**.

Each month **SYS-CON Media** reaches over half a million i-technology professionals through its specialty journals, magazines, books, conferences, and the **SYS-CON** Interactive portal with its 27 Web sites at www.sys-con.com.

Which Level of Engineer Are You?

It's time for a reality check



WRITTEN BY
BILL BALOGLU &
BILLY PALMIERI

The high-tech employment marketplace has been rocked by the failure of hundreds of start-ups and dot-coms. The stock market seemed to turn and attack the "hot" new companies that it celebrated just months ago.

Analysts estimate that this past year in the Silicon Valley/San Francisco Bay Area 30,000 people lost their jobs; 15,000 of them are engineers and 1,500 are senior Java engineers. Similar metrics can be applied to other regions where dot-com fever spread like wildfire.

How will all this market change affect engineering rates, salaries, and opportunities for the future? A lot depends on your level.

Which level of engineer are you? Take a stab at these questions that are among those we use to qualify Java engineers and determine their level of expertise:

1. Should a class file name appear in the classpath (e.g., should MyExample .class be in the classpath?)?
2. Does Java allow multiple inheritance among Java classes?
3. Which directory of the JDK 1.2.2 (or JDK1.3) contains the file rt.jar?
4. Which file, in which directory, contains a list of Java security providers for Java 2?
5. In JDK1.2.2 (or 1.3), what's the default garbage collection algorithm?
6. Which Java extension package supports HTTPS protocol?

Check your answers in the box below.

- If you got only questions 1 and 2 correct, you'd most likely qualify as an entry-level engineer.
- If you got questions 1–4 correct, you may qualify as an intermediate engineer.
- If you got questions 1–6 correct (without consulting a book), you'd most likely qualify as a senior engineer.

When contemplating the future of a high-tech career, keep two things in mind:

- Like it or not, the world is changing to an Inter-/intranet-based economy.
- The Internet economy should continue to grow and be healthy in the long term.

Here's our compensation forecast for senior, intermediate, and junior engineers:

Senior engineers are likely to see contract rates and full-time salaries leveling off and possibly going down from where they've recently been. We're seeing this as a reality check time for both senior engineers and companies that employ them.

The most savvy of senior engineers recognize that there's a change going on in the marketplace and they need to be flexible in considering opportunities. Many are looking at the long-term value they can get from a particular contract or full-time position as opposed to being "sold to the highest bidder."

Companies are becoming more cost-conscious as "money is no object" spending is replaced by real budgets and head-count issues. For full-time positions, managers might hire engineers who are a bit less experienced and less expensive, but trainable.

Because so many Java engineers are back in the job market, hiring managers are reviewing more résumés and interviewing more people, which gives the false impression that there's suddenly a glut of senior engineers to choose from.

In reality, senior engineers are still the smallest percentage of all engineers and the best ones are not much easier to find than they ever were. Many of those new résumés are actually from junior or intermediate engineers who were snatched up by start-ups or dot-coms in the recent hiring frenzy.

The bottom line for senior engineers is that the time between jobs is likely to get longer, compensation will become more competitive, and managers will have more hiring options to choose from. It's important to stay flexible, keep an open mind, and sharpen your negotiating skills.

Intermediate engineers are likely to see rates and salaries go down, but they're also likely to find more opportunities as companies look to hire more mid-level, affordable engineers for full-time positions.

If you've been burned by a pre-IPO-gone-bust experience, look for a full-time position at a mid-sized or well-established company, especially one with exciting technology that will enhance your skill set.

Contracting opportunities are likely to become scarcer for intermediate engineers as companies balance their budgets by using fewer, but more senior contractors for specific projects. But full-time positions should be available.

Junior engineers will see shrinking contract rates and fewer contract opportunities, but they're likely to find more full-time opportunities open to them. This is actually a healthy scenario, because a permanent position at a strong company is the ideal way for a junior engineer to develop his or her skills.

Junior engineers benefited from the contract hiring frenzy of the last few years, when companies were forced to pay inflated contract rates and salaries for anyone who knew anything about Java. But without enough time to gain much meaningful experience at that short-lived start-up, many junior engineers still lack the skills to succeed as contractors.

Today the smartest junior engineers are less concerned with retiring before age 30 than they are with expanding their skills, their knowledge, their résumé, and their long-term marketability.

Pre-IPOs will need to be more creative and aggressive than ever to attract good Java talent. You'll need to prove that your business plan, funding, and technology are sound and that your start-up is more than another flash in the pan.

billb@objectfocus.com

billp@objectfocus.com

Answers

1. No 2. No 3. /jre/lib (not /lib), 4. /jre/lib/security/java.security, 5. Incremental collection, or Train collection, or Hotspot algorithm, or short (time-bound) and frequent collection. 6. Java Secure Socket Extension (JSSE) package or javax.net.ssl package.

AUTHOR BIOS

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

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

WRITTEN BY RICK ROSS

Don'tGive It Up Too Easily, Make Them Earn Your Affection

As an advocate for Java and Java developers, I've never been a more persuaded or passionate believer in the deep value of the Java platform and philosophy. For me, however, advocating this Java vision neither implies nor requires a pro-Sun outlook in any way. I admire Sun and respect many of the fine people I've met who work there, but in the end Sun is just a company and is obligated to place shareholder wealth ahead of all other interests. I truly wish more Java developers would recognize themselves as equal peers to the corporate players and stop ushering themselves so quickly into different companies' camps. We have our own interests to look out for, and we have no compelling need or motive to pledge allegiance to any company's flag.

The Java developer community, as a group, is a separate, equal, and powerful participant in the global dialog about this critical and firmly established technology. We should stand alone in our own Java developers' camp and insist that every leading company invest much more deeply in us if they hope to maintain our continuing support. They need us at least as much as we need them. Java developers today are far too quickly pigeonholed as pawns for Sun, and we should not be so easily won. Java's tenure as the leading platform for network software development is hardly so fragile that we dare not risk questioning Sun's actions or motives as Java evolves. Indeed, we must. When has it ever been wise to grant blind trust to any corporation for the care of something precious? What dupes we would be if we didn't keep a prudently watchful eye.

The highly polarized "Sun versus Microsoft" equation with "good versus evil" has blinded too many Java developers from seeing the subtle advantages of more vigorous and open competition in the Java economy. It doesn't have to be either Sun or Microsoft in control – this is much too simplistic to constitute an acceptable worldview or a model of a thriving developer economy. None of us should be a mere point on a line somewhere between Sun and Microsoft. Many other companies, organizations, and individuals have excellent vision too, and we should always have our eyes wide open to recognize the most worthy of them whenever they appear. Technology moves quickly, and ours is a shifting landscape in which the leaders can't rest on their laurels for long.

I still want Java to be truly portable, reliable, open, standard, bugfree, and high-performance. It baffles me that so many independent, intelligent developers regard Sun exclusively as the great champion in the realization of this vision. Literally hundreds of companies and thousands of individual developers have generously contributed their hard work and energy to make Java the awesome platform it is today; in the end, however, Sun almost always gets the credit. People who know Sun's history better than I do tell me that its long-term track record as a software leader is not very impressive. They say that Sun's emphasis and strength has always been in the hardware business, and that its software initiatives have often missed the mark.

In fact, if Sun had allowed Java to be more genuinely open from the beginning, we would now have much broader support for competitive virtual machines, development tools, and market-driven innovation. Even during the unprecedented Internet heyday of these past few years, the third-party marketplace for Java developer tools has been an anemic shadow of what it could and should be. In our capitalist system it's expectation of profits that drives valuable innovation, and apparently the incentives in the Java developer tools market have not been sufficient to entice and sustain healthy business competition. This is a tough problem that we, the Java developers, should focus on and seek to cure.

So I hope more Java advocates will separate themselves from the "Java equals Sun" mentality and start challenging Sun and every other company that wants Java developer support to prove its ongoing value. A vast number of brilliant minds have contributed to the shaping of the Java platform and the vision we support, and the majority neither work for Sun nor have any reason to see Sun's shareholder interests placed ahead of anyone else's.

Sun, Microsoft, Oracle, IBM, HP, and the rest should be given a wake-up call to treat the members of the developer community as genuine partners or inevitably face losing our support to those who will. Sun has no more permanent claim to our affection and loyalty than any of the rest. I'm here to stand only for Java and Java developers, so I feel that any company that wants our continuing support should have a very good answer to the question, "What have you done for us lately?"

Comments on Microsoft Settlement

It's obvious that this settlement in no way represents "peace" between Sun and Microsoft. On the contrary, I expect to see renewed hostility on a much larger scale as these companies and others battle for strategic position in the huge emerging "Web services" market built with technologies such as UDDI, SOAP, and XML. It's also clear that Microsoft still doesn't get it. Their ridiculous attempt to position the "JUMP to .NET" vaporware as help for Java developers exemplifies their arrogant disinterest in our real priorities and concerns. It would be so much more productive for Microsoft to invest in building technology tools that will be genuinely attractive to Java developers rather than cajoling us with such pathetic PR manipulations. Who do they think they're fooling?

rick@usermagnet.com

AUTHOR BIO

Rick Ross is founder of the JavaLobby (www.javalobby.org) and president of UserMagnet, Inc. (www.usermagnet.com). UserMagnet leverages its expertise in Java, XML, and Web services to deliver advanced e-loyalty solutions.