HOW TO SURVIVE YOUR FIRST CORBA PROJECT!

JAVA™ DEVELOPER'S JOURNAL

The World's Leading Java Resource

Volume: 4 Issue: 11, November 1999

COMING IN DECEMBER
SYS-CON Radio and Java Developer's Journal media cosponsors of Java Business Conference New York City Dec. 7-9, 1999
DON'T MISS OUR COVERAGE!

Leasing and Flying

An Online Airline Ticket Store
PART 4

JAVADEVELOPERSJOURNAL.COM

SYS-CON Radio
Interview with Peter Coad of Object International
pg. 46

RETAILERS PLEASE DISPLAY UNTIL JANUARY 31, 2000

SYS-CON PUBLICATIONS

Protoview

www.protoview.com

SEAN RHODY, EDITOR-IN-CHIEF

My Forté

Here's an old joke. A guy in a strange town needs to get a haircut. There're only two barbers in the town, but the guy doesn't know either of them. Which one does he pick? The answer is the guy with the worst haircut. Why? Because neither barber can cut his own hair, so the guy with the worst haircut is the better barber.

What's this got to do with Java? Well, it reminds me of the strange position that Sun has been in for the past several years. I think Java is one of the most profound software concepts to ever come along, and there's no question that Sun is the proud owner. I'm very happy with the way the language has been improved over the past few years. And anyone who reads ***JDJ*** knows that I think Enterprise Java Beans, or Java 2 Enterprise Edition, is a masterpiece.

But there's always been a fly in the ointment with Sun. While the language is great, and their vision impressive, let's face it – their tools stink. I hate to be that blunt, but in my opinion that's the way it is. I spend a good deal of time on each assignment reevaluating which development tool to use. There's a gang of three playing functionality leapfrog out there, and no matter where I go, someone has a favorite. But that gang of three is IBM, Inprise and Symantec. Sun isn't in there.

This shouldn't be too surprising in and of itself. Sun is first and foremost a hardware company. And if you've been reading faithfully, you know that I think Java is ultimately designed to sell hardware. But the point remains that while Sun has done a fantastic job crafting a new language, and an even better job of making it run anywhere and marketing that vision, they're not a software tools company.

I guess it's more surprising that IBM is in the gang, coming from the same hardware orientation. But the IBM guys have embraced Java so hard they like to refer to themselves as the "true Java leaders." I'll bend your ears (or eyes) some other time on the IBM story.

Probably the most surprising thing of all is that it's taken Sun several years to realize that they don't have the expertise in tool development to be more than a bit player. Let's face it – most developers use Windows to write code. And it's the guys that have been making windows development tools for years, Inprise and Symantec, that have captured the early lead in providing development tools.

So it came as no surprise to me that Sun recently acquired Forté. For those of you not familiar with Forté, let me help. Forté has been in distributed computing for several years, making a high-end tool that generated C++ code and could dynamically partition an application to run in various places. My friends who worked with Forté say that it can do some fairly sophisticated things, but there's a bit of a learning curve involved.

Forté saw the writing on the wall some time back and started working on a Java version of their product, which is now known as SynerJ. I haven't seen this product yet, but I'm going to make it a point to get a copy so we can review it.

But that doesn't really matter, at least not right now. The important thing is that Sun has recognized their deficiency and has taken steps to correct it. Perhaps that also explains the large investment Microsoft made in Inprise. Far from the bailout that they did for Apple, for which they are the largest single software vendor, this investment might have been a blocking play to keep Sun from acquiring Inprise. Thus Sun had to get another tool.

Only time will tell whether Forté will turn out to be a tool that Sun can use to contend with the rest of the gang. While Sun has the cachet of being the thought leader in Java, the other vendors have captured most of the market. It's going to be interesting to watch Sun try to overcome this lead. In the meantime, I need to get a haircut. Anybody know a good barber?

LETTER TO THE EDITOR...

Sean:

Thanks for a quality publication; it's "meaty." In reference to your covering M$ DCOM: you are clearly being open-minded – but sometimes that is not appropriate.

We all deal with "Poor Bill's" extortions; we all need to bend to the majority (however wrong) or else be destroyed. BUT...

When we are studying in our chosen scientific discipline (e.g., JAVA), we do not necessarily need to include those who try to pervert and obscure our science.

You are not a strictly "scientific" journal so you refer to the (very) popular "trash" we all must live with. BUT...

When the "trash" is so deliberately antithetical to the community you serve…when it is deliberately trying to destroy the community [you] serve, you must present them as the threat they are. If you have to catch flak, catch it for telling the (unpopular) "truth"(e.g., Microsoft is a malicious but powerful player) instead of for "fairness" (e.g., Microsoft is just another vendor).

Good luck,

***Brian Hayes*, a011882@ibm.net**

sean@sys-con.com

AUTHOR BIO

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

WRITTEN BY DAVID SKOK

J2EE Standard Dramatically Changes Application Server Market

Earlier this year I wrote an article describing how Enterprise JavaBeans had impacted the application server market, causing a convergence between Web application servers and distributed object and transaction servers. With the advent of the J2EE (Java 2 Enterprise Edition) standard, we're about to witness another seismic shift in this market.

It's worth looking at what J2EE is – and isn't. J2EE builds on the Java 2 standard, and adds specifications for most of the important programming interfaces in an application server. Key interfaces are EJB for middle-tier logic, and servlets and Java Server Pages (JSP) for dynamically generated HTML pages. Other APIs include JNDI (naming and directories), JMS (messaging), JavaMail, JTA/JTS (transactions) and RMI-IIOP (remote communications protocol).

While the J2EE specification comes with a reference implementation and compliance tests, it's important to understand that J2EE doesn't cover a number of APIs and functions that are central to today's application servers. For example, there are no specifications for how to connect to legacy or nonrelational data sources; how to create applications that interact with other systems via XML; how to cluster, load-balance and fail-over a server; or how to manage a server.

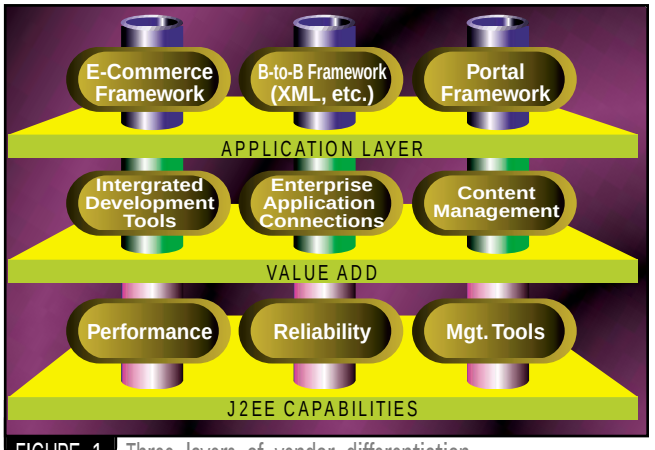


FIGURE 1 Three layers of vendor differentiation

Benefits of Standardization for Customers

From early indications it appears that the J2EE standard will do for application servers what SQL did for relational databases. From a customer's standpoint, industry standardization leads to the following important benefits:

- Cross-vendor portability, which lowers customer risk by eliminating vendor dependency; skills are also portable
- Large aftermarket of third-party software and components
- Wide range of tools and supporting products
- Large pool of skilled developers
- Readily available training, books and information

Reusable J2EE components will bring choice in a competitive market. Developers will assemble applications using a variety of commercially available J2EE components mixed with their own custom components.

J2EE Splits the Market in Two

J2EE has been adopted by key players in the application server market, and has been enthusiastically embraced by the Web development community. The only exception is Microsoft, which is pursuing its own strategy (DNA and COM+). Microsoft may not gain a great deal of the market due to the following:

—continued on page 56

dskok@silverstream.com

AUTHOR BIO

David Skok, chairman and founder of SilverStream Software, Inc., holds a BS honors degree from the University of Sussex, England.

The Business Advantage of EJB PART 2

More on developing portable EJB applications

WRITTEN BY
JASON WESTRA

Last month, in EJB Home, I covered the business advantage of Enterprise JavaBeans' portability from a high level. First I discussed the various types of portability. Then I covered (1) the portability goals the creators of EJB had in mind when developing the specification and (2) how your business can achieve a competitive edge through EJB. This month I'll finish up the discussion of EJB portability from a developer's perspective.

Avoiding Portability Pitfalls in EJB

While the EJB specification 1.0/1 has laid the foundation for building portable enterprise beans, there's much left to be specified before true interoperability will be a reality. The EJB specification is vague in a number of areas important to EJB portability, including a container provider's responsibilities, multiple-vendor EJB server integration, security, distributed/asynchronous event notification and mappings for COM integration.

I'm not going to bash EJB for its lack of portability this month. Besides, if you want to read articles that downplay EJB, you're reading the wrong magazine (and the wrong author)! I will, however, offer a number of tricks, traps and tips that I and my colleagues have encountered when building portable EJBs. Add these techniques to your armament and you'll shield yourself from many pitfalls in EJB portability to date.

Dos and Don'ts of EJB Portability

When you were a child, your mother or father probably separated your world into dos and don'ts, such as "Do your homework!" and "Don't put gum in your sister's hair!!" This is a nice model for describing good and bad software development techniques as well; thus, for your enjoyment, I've made a list of dos and don'ts on the topic of EJB portability. The list is definitely not complete and I'd love feedback on "gotchas" that you've encountered in your experience no matter the EJB server. Maybe a second article listing reader encounters could be compiled in the future.

If you don't recall last month's coverage of the wrapper design pattern, you'll want to dust off your October issue of ***JDJ*** and reread Part 1 on the topic of EJB portability. For the rest of you…hold onto your "wrapper." Not only is a wrapper a better place for your gum than your sister's hair but, as you'll see, the wrapper design pattern will play a key role in the portability of your enterprise beans!

Don'ts

Proprietary Value-Add Features

Don't use hard-coded references to a particular application server's proprietary features. J2EE is aimed at providing the APIs necessary for you to build portable, distributed applications to solve diverse business problems. However, it was created after many application server vendors had already implemented their own value-add features into their products. The problem inherent in these features is vendor lock-in, a problem EJB (and J2EE) looks to solve! Following is a list of value-add features to be wary of.

- ***Proprietary event models***: Examples of proprietary event models include WebLogic Events and the Novera Integration Server's Event Components. The EJB specification says nothing about asynchronous event notification (EJB is a transactional component model, while events are usually considered nontransactional). Because certain business problems require asynchronous communications along with publish and subscribe mechanisms, event models are an often implemented proprietary feature.
- ***Nested transactions***: To provide backward compatibility with existing transaction monitors and middleware, the EJB specification supports only flat transactions. Some EJB servers (I won't mention names, but the color blue comes to mind) may tout their value-add "nested transaction" support. Buyer, beware!
- ***System management facilities that aren't Java Management API (JMAPI) compliant***: Coding special management interfaces into an enterprise bean to monitor performance, or otherwise, will be disastrous when moving the bean to another EJB server.
- ***Shared services***: EJB has no concept of shared services that hold state and are accessible by all clients. If this feature is necessary in your application, you'll have to look beyond EJB, possibly to value-add features in your application server, such as Novera's Registered Objects and Daemons.
- ***Proprietary application server "login" techniques***: Often an application server requires a client to make a connection to it via a proprietary login object or JavaBean. The login not only establishes a connection to the application server, but performs necessary security checks on the user as well. Coding logins to a particular application server will tie you to this vendor and hinder an easy move to another application server if the future demands it. Thus the code in your enterprise beans needs to be portable, and you also have to think about the portability of your client code when designing for minimal impact from one application server to another.

Java Native Interface

This is an easy one. When building a portable EJB, don't implement JNI code in your beans. Although it's a necessary feature for Java to make inroads into Wintel applications, JNI hinders platform portability for your EJB.

Bean-Managed Persistence

If you're concerned about persistence portability, don't use bean-managed persistence in your entity beans. The August column of EJB Home (***JDJ*** Vol. 4, issue 8) warned about the disadvantages of bean-managed entity beans. When coding an entity bean with bean-managed persistence, you have the potential of tying yourself not only to a particular database, but also to a particular EJB server by utilizing the EJB server's connection management features.

Bean-Managed Transactions

While we're on the subject of bean management....Don't develop enterprise beans that use bean-managed transactions (e.g., TX_BEANMANAGED) unless absolutely necessary. Once a bean is coded to use such transactions, you're stuck with this decision. You won't be able to switch to another transactional property during deployment! An enterprise bean with a transactional property of TX_BEANMANAGED may also unknowingly tie itself to an EJB server's implementation of a transaction service. As a reminder, if you're using bean-managed transactions, don't nest transactions in your Enterprise JavaBeans!

From the last two "don'ts" you can see there's a trend here – bean-managed is generally a portability liability and requires complex implementations. On the other hand, the EJB component model for development and deployment of server-side components promotes portability while sheltering fledgling developers from the nuances of transaction management or object storage.

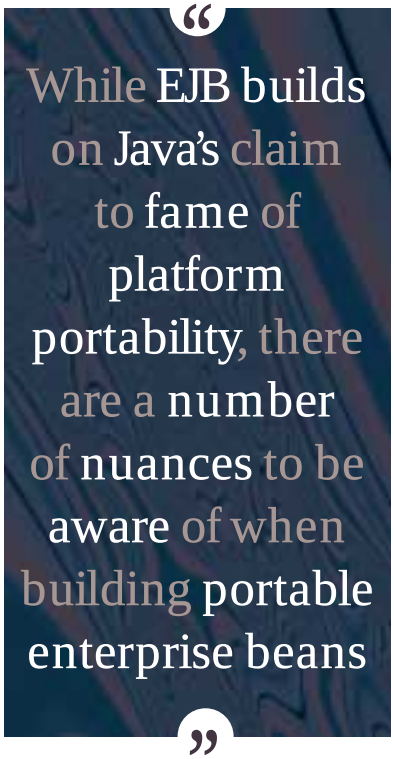
Dos

Proprietary Value-Add Features

Instead of using an application server's proprietary features, use the Java Platform for the Enterprise APIs (J2EE) offered by your application server. While too new to be standard across all application servers, these APIs will be offered in most servers in the near future. If you must use an application server's proprietary features, use a wrapper that maps to the API as best you can to encapsulate calls to these features. Wrappers help to localize proprietary code, so when your EJB server becomes J2EE compliant, you'll have minimized your changes to a few classes and methods.

Note: Even some Java Enterprise APIs may need to be wrapped to be truly portable across application server vendors! For instance, to access an enterprise bean, you go through the bean's home interface, which acts as a factory to give you a handle to your EJB. However, you first have to locate an enterprise bean's home interface by calling through the Java Naming Directory Interface (JNDI). The JNDI requires you to specify the InitialContextFactory to be used to perform a lookup on your enterprise bean's home. The parameter you pass for the InitialContextFactory corresponds to the EJB server that the enterprise bean is currently deployed in; thus it is inevitably proprietary to your current EJB server.

To avoid references to a particular vendor's InitialContextFactory and ease your pain when moving to a different EJB server, wrap calls to the EJB server's InitialContextFactory within your own class, such as the class in Listing 1.



This wrapper is fairly basic, but does the job of a wrapper by localizing calls to an EJB server vendor's InitialContextFactory. When getContext() is called, the wrapper uses its current values for user, password, url and contextfactory to return a valid InitialContext object. The InitialContext, of course, is used to perform a lookup of your enterprise bean's EJBHome interface. If your product contains thousands of JNDI lookups, you can see how the EJBContextWrapper would minimize your struggle to migrate to another EJB server.

To use the context wrapper, import it into your Java IDE of choice and simply modify the JNDIConf.properties – specifically, the name of the EJB vendor's InitialContextFactory in question. In the following code you can see that I have set the url and contextfactory properties to point to the BEA WebLogic application server where my enterprise beans are deployed.

```
url=t3://localhost:7001
contextfactory=weblogic.jndi.TengahInitialContextFactory
user=
password=
```

I've left the user and password empty as they're not required values to get an InitialContext returned. Following is the classic example of the AccountBean (getting as old as hello, world, wouldn't you agree?). This code gets a handle to the AccountHome through the Context object returned via the EJBContextWrapper.

```
Context ctx = EJBContextWrapper.getContext();

// Contact the AccountHome through JNDI.
AccountHome home = (AccountHome)
ctx.lookup("AccountJNDIName");
```

While serving our purpose for this article, the EJBContextWrapper could be enhanced in a number of ways. For instance, you could include more overrides for added user flexibility. Also, to support disparate EJB server vendors, you could add a dynamically loaded hashtable that contains each deployed enterprise bean's EJB server context information keyed by the bean's JNDIHome name. This would enable you to find the correct EJB server in the enterprise environment hosting the enterprise bean in question – for instance, by passing a java.lang.String argument into getContext() (i.e., getContext(String JNDIName)). This is used to look up the correct settings from the hashtable. This technique is needed only if you have multiple EJB servers running under different naming services in your enterprise.

Wrap Native Code

You want your business logic to remain portable! To do this, separate your business logic from native code calls by making a separate portable EJB that wraps a legacy/JNI EJB. This gives anyone needing access to the legacy code the benefits of EJB – namely, location independence, transaction support and lifecycle management – while keeping the business logic in a portable enterprise bean.

Container-Managed Persistence

Instead of using bean-managed persistence, offload persistence responsibilities to the EJB container by utilizing container-managed persistence for everything possible. In some cases a query may get too complex to be handled through the EJB server's container and custom database code is thus unavoidable. However, for a large percentage of your persistence needs you should be able to take full advantage of EJB's component model and map your entity beans at deployment time to their respective storage mechanism(s).

Container-Managed Transactions

Instead of writing transaction code in your bean, concentrate on its business logic and leave transaction management up to the EJB container. Determine the transactional behavior of your EJBs at deployment time and keep your enterprise beans flexible.

Lowest Common Denominator

There's no silver bullet solution to EJB portability. The answer lies in the Lowest Common Denominator approach, which says, "Do not use a feature that is not widely available among vendors or an industry recognized standard." To use the LCD approach when building portable enterprise beans, use only those features designated mandatory in the current EJB specification. This will ensure that all compliant EJB servers will offer your enterprise bean the services it needs to execute properly. The LCD approach applies to other APIs besides the EJB specification. It assumes you wouldn't incorporate proprietary features from your application server vendor into your application either.

Summary

While EJB builds on Java's claim to fame of platform portability, there are a number of nuances to be aware of when building portable enterprise beans. To ensure that the next enterprise neans you develop are portable and add business value to your users, make sound architectural decisions, utilize design patterns like the "wrapper" as portability enablers, and follow the tips listed above.

As mentioned, this list of EJB portability tricks isn't exhaustive by any means. I look forward to hearing from everyone who'd like to share their experiences.

Author Bio

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

jwestra@uswestmail.net

Listing 1: EJBContextWrapper

```
import javax.naming.InitialContext;
import javax.naming.Context;
import java.util.Properties;
import java.io.FileInputStream;

// EJBContextWrapper is a utility class to service requests
// for a JNDI context.  It loads JNDI lookup properties from
// a properties file, or uses defaults when none are preset.
// It also allows the properties to be bypassed by calling
// overrides of getContext()

public class EJBContextWrapper extends java.lang.Object
{

 private static String url = "";
 private static String factory = "";
 private static String password = "";
 private static String user = "";

 static {
  try {
   // load in the JNDI settings from configuration file
   FileInputStream in = new FileInputStream("JNDIConf.proper-
   ties");
   Properties props = new Properties();
   props.load(in);
   in.close();

   url = props.getProperty("url");
   factory = props.getProperty("contextfactory");
   user = props.getProperty("user");
   password = props.getProperty("password");

  } catch (Exception ex) {
   // defaults
   url = "t3://localhost:7001";
   factory = "weblogic.jndi.TengahInitialContextFactory";
   user = "";
   password = "";

   System.out.println("Failed to initialize EJBContextWrapper "+
                      "properties. Using defaults.");
   System.out.println("url =  "+url
                      +" factory = "+factory
                      +" user = "+user
                      +" password = "+password);
  }

 }

// constructor, do not let this get instantiated

 public EJBContextWrapper ()
 {}

 // getContext() override that uses loaded values to get a
 // context in JNDI

 public static InitialContext getContext() throws Exception
 {
  return getContext(url, factory, user, password);
 }

 // getContext() override that uses specified user/pwd and
 // loaded url/factory to get a context in JNDI

 public static InitialContext getContext(String aUser, String
 aPwd) throws Exception
 {
  return getContext(url, factory, aUser, aPwd);
 }

 // getContext() override that uses specified parameters to
 // get a context in JNDI

 public static InitialContext getContext(String aUrl, String
 aFactory, String aUser, String aPwd) throws Exception
 {
  Properties p = new Properties();
  p.put(Context.INITIAL_CONTEXT_FACTORY, aFactory);
  p.put(Context.PROVIDER_URL, aUrl);

  if (user != null || user != "") {
   p.put(Context.SECURITY_PRINCIPAL, aUser);

   if (password == null)
    password = "";

   p.put(Context.SECURITY_CREDENTIALS, aPwd);
  }

  return new InitialContext(p);
 }

}
```

Blue Sky

www.blue-sky.com

Strange...But True!

A stew with some peculiar ingredients

WRITTEN BY
ALAN WILLIAMSON

This has been a busy and bizarre month. A number of weird and wonderful things have happened, and I'll take you through them one by one.

If you remember, last month I promised to tell you about a strange rendezvous one of our chaps had with a Microsoft person. As regular readers know, we're based out in the back of beyond in the middle of the Scottish lowlands. We're the only software company for over a hundred miles in any direction, and it's a fact of life that you don't bump into many developers out on their lunch break. So you can imagine our surprise when Keith came in one day and announced he'd met a lawyer who worked for Microsoft at the local pub the night before. Hot damn.

Now we all know that Microsoft isn't the most popular company in the world of Java. With the present court case and what have you, the majority of the Java community isn't too happy with Microsoft's vision for the future. You also know that I have never been quiet about my thoughts on the whole issue, and have often been quite outspoken in this very column. So when one of my chaps "accidentally" bumps into a Microsoft lawyer in our neck of the woods, you have to be concerned. So far we've heard nothing more, but I'll keep you posted on any developments in this area.

For many a budding actress the draw of Hollywood is sometimes so great that they'll take on nearly any job to try and succeed. A number of mainstream actresses have questionable appearances on their résumés. Well, in our distant past we have a similar David "Red Shoe Diaries" Duchovny episode that has come back to haunt us.

Back when we started out, we had a lot of expenses to cover, most notably the rent and the cost of the bandwidth. Since Java was very slow at the start, we had to do a number of projects that would bolster the turnover until Java kicked in. One of these projects was that of Web hosting – providing an upstream service for companies. On the face of it, this wasn't a bad diversion, but it did lead us into hosting several adult-oriented Web sites.

The pornography laws here in Britain, however, aren't quite as liberal as those in the U.S. or some European countries. One of our clients pushed the boundaries of decency and, after a number of complaints, we decided to move the company on and have nothing more to do with it. Three years later, a knock on the door – the vice squad. Said company has been charged with a number of obscene publication violations, and they were tracing its history. (Fortunately, we have nothing to do with the actual prosecution and our company name will be left out of the whole trial. Right enough, me writing about it here in ***JDJ*** sort of null-'n'-voids the whole scandal thing! Oops.)

The officers stayed with us awhile and we took them around the place, showing them various bits of our company and our local town. It was the first time I had the opportunity to talk to guys that were in this sort of policing. It's a damn shame I'm not allowed to retell any of what they said to me, but whoa! Did they have bizarre stories to tell, including one involving a milk-crate, a bottle of ketchup and a horse! Nuff said, if you get my drift. But it gave me an idea for a book I could write that would collate all these sort of stories. Just need to find a publisher . . . oh, Fuat!

This month we had a very special guest stay with us, James Duncan Davidson, head of the Servlet API from Sun. He was over in Germany doing a conference and stopped by us for a few days before heading back to California. It was good to see him, but in true Alan Williamson style the visit didn't run as smoothly as one would have hoped.

I have a Toyota MR-2. Those of you who know the Japanese sports car will know it's not blessed with the largest of boots – trunks, to translate for our American readers. James was kind enough to e-mail me the size of his largest suitcase so I could determine whether I needed to borrow somebody else's car to pick him up at the airport. A quick runaround with the measuring tape seemed to suggest that the MR-2 was more than ample for the job at hand. So I made my way up to Glasgow airport.

Met James off the flight, but only when we tried to get his suitcase in the back did we realize that the boot was way too small. Bugger. Well, that's okay, we'll tie the boot down. But with what? I had no string, no rope, not even a belt. Oh, dear. We had a two-hour journey ahead of us, and there was no way we could drive with the boot bouncing up and down. It was at this point James came up with a blinder of an idea. Modem cable! We sacrificed a modem cable and tied the boot down with said cable, and it never moved a single inch. Fantastic. Only a pair of geeks could have come up with that solution.

It still wasn't plain sailing after that. By the time I got back to the office a raging thunderstorm had knocked out the power and the place was in complete darkness. Yes, James, we really are in the middle of nowhere. It's quite eerie how quiet a building becomes without the constant reassuring hum of servers in the background. I crawled into the house, found some candles and proceeded to give James the tour of the office, in the dark. There was something quite romantic about it all, but – no offense to James – I would have preferred to have been locked down with my other half, Ceri, in this pitch blackness.

When the power came on a couple of hours later, we both pounced on our e-mails like two addicts trying to get our next electronic fix. Welcome to Scotland, James!

Thread Update

Several months ago I detailed a problem we experienced with threads and the apparent lack of cleanup the virtual machine had in this field. We discovered that if we didn't call stop(), the thread wouldn't be cleaned up when the garbage collector ran. When I announced this, I got a whole flood of e-mails denouncing me as a charlatan, or purely a fiction author. This upset

me more than a little. It has to be said that many of you still don't know good manners when it comes to composing e-mail. Let me give you a tip: if you want someone to reply to your e-mail, or at least read it, be polite. Try it – you may be surprised.

The major problem I faced was that I didn't have a simple test case that I could use to prove my point. My servlet environment highlighted it perfectly, but that was a scenario that couldn't be transported easily to another machine. The mailing list proved to be some comfort. Many of you had experienced the same problem, but in the Swing world. Again, no test programs were developed that could demonstrate this.

I was determined to simplify our code in such a way that we could prove it. I wanted to show the people who denounced me that the problem did indeed exist. Late one night I was sent a heavenly message from an angel in the strange form of Oracle. One of the core engineers found the problem and had a test program that proved it perfectly every single time. Hurray! Saved. I tested the program quickly on a number of JVMs and, sure enough, the mythical thread problem is not mythical.

So I can now state publicly that the thread does indeed exist, there is a test program, and it has been submitted to Sun as an official bug. I'll give you updates as and when I know more. So watch this space.

Mailing List

The mailing list is beginning to generate some seriously good threads of conversation. In the last month we've had a number of debates on the future of Java and whether it should be open-sourced or not. One correspondent posed the question, and we all answered it. Surprisingly, not many supported the open sourcing of Java, which was good. Personally, I had anticipated a much greater swing of support, but it was good to hear everyone's structured answers about why it should, for the time being, be left to Sun to manage.

Discussions on what we'd like to see in Java in the next wave have also taken place. One thing that came up time and again concerned operator overloading. I must say that I liked this facility in C++, and understand James Gosling's apprehension about not including it in Java. But we hear that it's seriously being considered.

If you want to be part of the discussion, send an e-mail to listserv@listserv.n-ary.com with subscribe straight_talking-l in the body of the e-mail. From there you'll get instructions on how to participate on the list. Thank you all for your continued posts – I thoroughly enjoy the variety of topics discussed.

Salute of the Month

This month the salute goes to the team behind Formula1. Some months ago I moaned about trying to read Excel spreadsheet files and the hassles we were going through. Many of you e-mailed me your stories, but none had any solutions we could actually use. One of the core developers of Formula1, Joe Erickson, read my woes, however, and e-mailed me detailing his joy in licking the Excel format. On his advice we looked at Formula1 – and I have to take my hat off to those chaps. A fine piece of software. If you ever need to read and write Excel files, you won't find a better solution than Formula1.

It's time to finish this up and head into another month of excitement. Last month, if you recall, I told you about the Neil Diamond phase we were going through. Well, this month we've moved on: I've discovered the wonders of Dolly Parton and, two CDs later, I am well and truly hooked. The irony of it is that this column is actually named after a movie from the selfsame lady. I feel bad that it's taken me nearly two years to actually discover her music. To tell you a wee secret, as I explore this style of music, I find myself getting into more and more country. Scary. Am I turning into my father?

alan@sys-con.com

Author Bio

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

AN ONLINE AIRLINE TICKET STORE PART 4

Leasing and Flying

This fictitious store allows passengers to lease equipment for the duration of a flight

WRITTEN BY AJIT SAGAR

Part 1 of this series appeared in **JDJ** *June, Part 2 in* **JDJ** *July, Part 3 in* **JDJ** *September*

This is the fourth in a series of articles focused on using Java and ColdFusion technologies to develop an Online Ticket Store application. As *JDJ*'s September issue had an XML focus, we went with the flow and discussed data formatting aspects of our store and developed XML objects to pass date structures between the Merchant Server and Service Access tiers.

This month's article focuses on the online store portion of our application, beginning with a description of the business offerings of the store. This is followed by a use-case analysis of the store, a methodology similar to the one in the July issue. We then design the classes for implementing the use cases and discuss the code needed to make our store a "reality." We'll also make use of the XML objects developed in the September issue to facilitate data transfer for the online store transactions.

The Online Airline Store Business Model

In addition to serving as a virtual travel agent, our application has an online store through which it sells merchandise. This includes the products offered via airline catalogs as well as unique items offered exclusively through the travel agency – books, magazines, clothing, gifts, souvenirs and computer equipment, for example.

So far, our store is like any other virtual store on the Internet that also sells airline tickets. But here's where it gets interesting. A unique feature our store offers is the ability for passengers to lease equipment for the duration of the flight.

This equipment includes:

- *Books and magazines*
- *Laptops for business or pleasure*
- *Portable CD players and music CDs*
- *Portable cassette players and audiotapes*

The idea is that someone who uses our Online Store to book a flight and purchase a ticket can instruct the store to have the leased equipment available on the flight he or she will be taking. Although the scenario is fictitious, I'd like to develop it further. I can think of several different options for our leasing operation:

1. The airline could keep an inventory of the equipment at the various airports and the passenger could pick up and drop off the equipment at the gate. The equipment may even be directly available on the flight. This, of course, would put a burden on the airlines to create the infrastructure and organization to support such a model.
2. The online store could be affiliated with a leasing store at the airport that holds the inventory. The passenger could pick up the inventory from the physical store.
3. The online store could have personnel who would be responsible for making the equipment available at the gate for the flight and picking it up at the other end of the flight.

If this kind of business becomes a reality, I can foresee airlines (Option 1) and independent agents (Options 2 and 3) competing for the business. However, one assumption we'll make for our application is that the equipment should always be leased via the online store. Thus, similar to a ticket purchase, payment for leased equipment would be made at the Merchant Server tier. The modules for this interaction (Shopping Cart, Catalog, Personal Profile Manager, Payment Manager, etc.) are thus implemented in ColdFusion and will be discussed in the December issue of ***ColdFusion Developer's Journal***. For an explanation of the tiers of this application and the software modules, please refer to the previous three articles. Figure 1 shows the four tiers in our application for the current reference. This is the same as Figure 2 in Part 1 of this series.

You may wonder what modules of the store will be housed in the Service Access tier. After all, that's the Java middleware tier in this application. Well, though the merchandise will be ordered and paid for at the Merchant Server tier, this tier doesn't have any knowledge of the local inventory and availability of the merchandise. The Service Access tier is responsible for this function – and for making sure that the order, whether it's a merchandise purchase or lease, gets "delivered" to the customer. Delivery may constitute shipping the actual product, making it available at the gate or keeping it ready for the customer at the physical store in the airport.

The UI design for this application isn't a major part of our Java-based design. A sophisticated UI will be developed in parallel using ColdFusion in the corresponding issues of ***ColdFusion Developer's Journal*** (Vol. 1, issues 4 and 6, and Vol. 2, issues 1 and 2).

COMPONENT	FUNCTION
Personal Profile Manager	Maintains a personalized profile for end customer, including data about the customer's flight preferences, frequency of transactions, history of goods purchased, etc. The customer may update this.
Catalog	Catalog of goods offered by the store, including merchandise available for purchase as well as lease.
Payment Manager	Manages payments made by the customer while interacting with the system. Consists primarily of credit card–based payment management.
Customer Profile Database	Contains the customer profile, including name, address, personal preferences, purchase history, purchasing trends, etc. Information can be used later to provide CRM (Customer Relationship Management).
Login Manager	Manages user login, authentication and passwords.
Shopping Cart	Keeps track of customer's current purchase as well as pending orders for purchased goods.
Merchandise Sales and Leasing Broker	The main operational module for conducting transactions related to the online sale and leasing of goods offered in the store. The Broker accepts orders from the customer, runs them against the Order Manager and returns a confirmation to the customer.
Products Database	Contains information about the products.
Order Manager	Interfaces with the back-office modules to check availability and to place an order. Placing an order results in creating a shipment (in the case of PURCHASE) or making the item available at the airport (in the case of LEASE).

TABLE 1 Description of application modules

Online Store Requirements

The Online Store entertains two types of transactions – purchase and lease. Once the customer selects the merchandise (this is handled by the Merchant Server tier), the order is sent to the Service Access tier. This leads to a check for availability against the Application Services tier. Note that the function of this tier is different from that in the previous articles. In this article the tier is responsible for satisfying a merchandise order instead of booking an airline ticket. In both cases, however, the Application Service tier acts as the back office for services offered by the middle tiers. If the purchase or lease were made directly against the airline, the Application Services tier modules would reside in the airline's back offices.

The application modules involved in this transaction are described in Table 1 and illustrated in Figure 2.

Once the "order" is placed, a confirmation is sent back to the Merchant Server tier, which is then responsible for getting the payment for the purchase. To keep the workflow simple, we're assuming a synchronous workflow through the system. In reality, part of this workflow will be asynchronous. The simplified workflow for this interaction is shown below:

1. Accept the merchandise order from the Merchant Server tier.
2. Determine whether it's a PURCHASE or LEASE.
3. In the case of LEASE, search for availability of the merchandise.
4. If not available, send back an exception.
5. If available, place an order with the Application Services tier. For a PURCHASE operation, this would be an order for a shipment. For a LEASE operation, this would be the order to make the equipment available at the airport.
6. Return the confirmation to the Merchant Server tier.

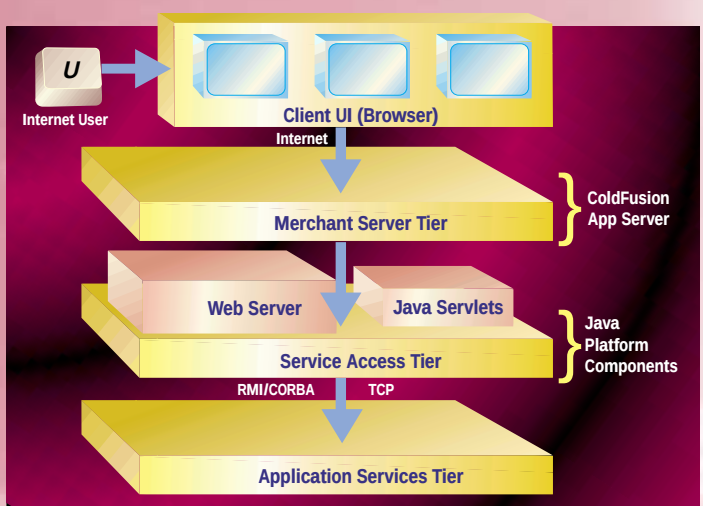


FIGURE 1 Application framework

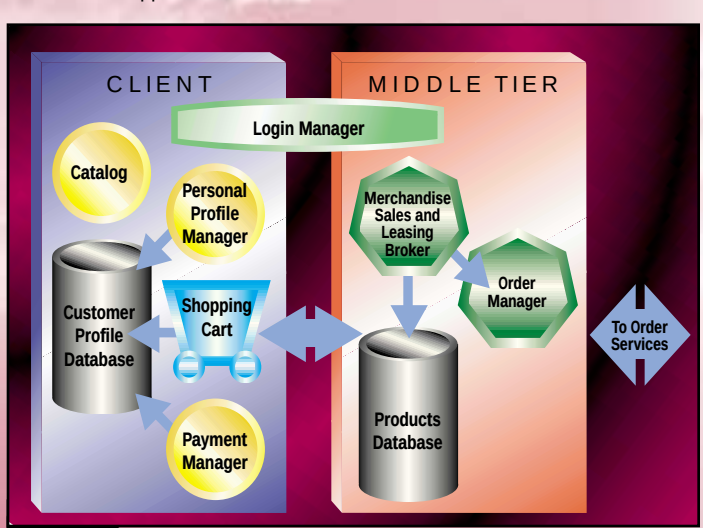


FIGURE 2 Online store modules

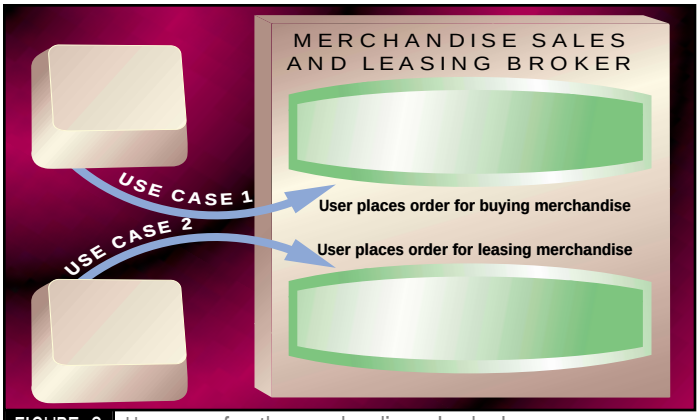


FIGURE 3 Use cases for the merchandise sales broker

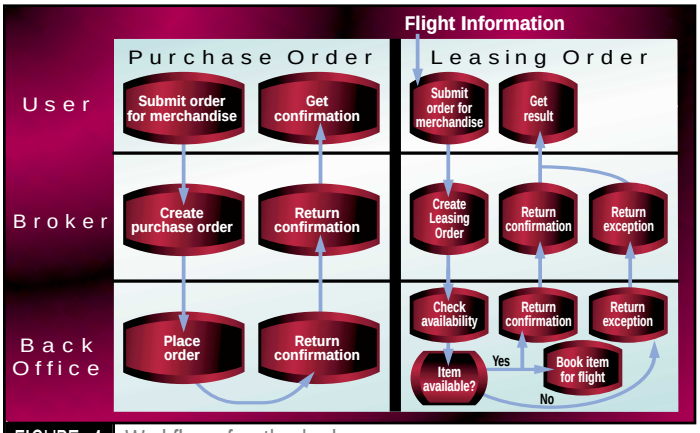


FIGURE 4 Workflows for the broker

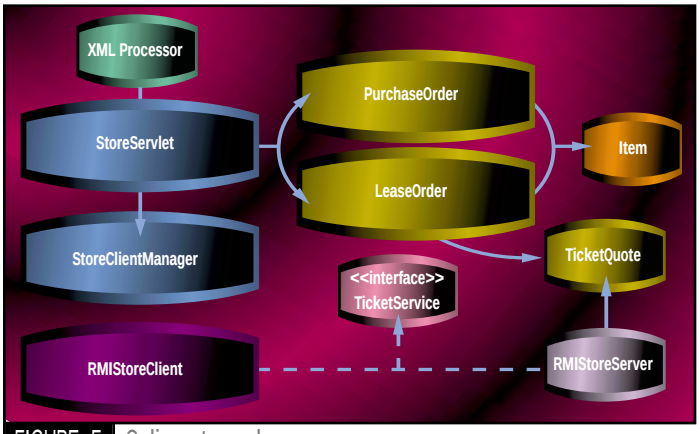


FIGURE 5 Online store classes

This is a simple workflow that runs through the system in sequential fashion. We'll implement this workflow in our system, ignoring other functionality such as canceling an order. The idea is to demonstrate how data flows through our Online Store from the end customer to the back office and back. Figure 3 illustrates the use cases for the Broker. Workflows are illustrated in Figure 4. The main difference between the two flows is that the Leasing Order workflow has a constraint on time because the items need to be booked for the flight and thus need to be at the airport at a specific time. Note also that the Leasing Order workflow follows in sequence from the Ticket Booking workflow described in the July ***JDJ*** article – that is, the items are leased only after the tickets have been reserved. The Purchase Order workflow is independent of any interaction the customer may have with the ticket reservation system. We assume that for a Purchase Order the customer will accept delivery whenever the merchandise can be sent by the system. This is similar to the regular catalog orders that state "Allow 4–6 weeks for delivery."

Data Interchange Formats

The following data objects were introduced in the September ***JDJ***. Some of the description is repeated here to put things in the appropriate context:

- *Lease Order*
- *Lease Confirmation*
- *Purchase Order*
- *Purchase Confirmation*

Let's go ahead and define the attributes of goods sold in the store. An item that may be offered by the store will have the following fields. Example values are assigned to the fields:

```
ITEM_NAME="CD Player"
ITEM_ID ="cd30056"
QUANTITY="2"
```

An order for the purchase of an item (or items) will also need information about the person making the purchase. This will consist of the following fields:

```
NAME="Clark Kent"
ADDRESS="123 Tiny Lane, Smallville, Kansas, 12345"
```

The fields listed above are common for both purchase and lease options. The lease operation requires one additional field. This is the reference number of the confirmed flight, and is required because it's the reference for the store to make the equipment available at the airport.

```
REFERENCE_NO="SA123456"
```

The hierarchy of the XML documents in DOM for the PURCHASE/lease was illustrated in the September ***JDJ***. The XML files are shown in Listing 1. I won't go into a detailed description of the code for the XML parsing for these objects as it's similar to the mechanism described for the ticket reservation part of our application as shown in September.

Class Design

Now let's define the classes involved in this set of transactions. For now, we'll forgo discussion on the UI and assume that the user's input somehow arrives at the Broker. In designing the store I'm restricting the discussion to an RMI-based connection to the back office. The previous articles have discussed the flexibility of using different transport protocols like CORBA, TCP/IP, etc. That design can easily be applied to our store. The classes for the store are described in the following sections. Figure 5 illustrates the class relationships.

StoreServlet

Listing 1 shows the StoreServlet class. This class receives a merchandise order request ("PURCHASE" or "LEASE") from the Merchant Server, packages it into a LeaseOrder (LEASE) or PurchaseOrder (PURCHASE) object, passes it on to the Order Manager and gets a confirmation (or an exception if it's a LEASE operation), and returns the result to the Merchant Server. The input to the servlet is in the form of an HTTP POST that carries one of the following XML data structures:

```
<LEASE>
<PURCHASE>
```

The LEASE structure is used only after the customer has purchased a ticket. The StoreServlet first checks to see whether it has a reference to the clientManager_ object. If not, it constructs a new StoreClientManager object. The request and response streams are copied into request_ and response_ variables for future use. Next, the method processParameters() is called. This method parses the input parameters and creates the appropriate Java object. The XML processing for this application is done in the XMLProcessor class. The methods for the LEASE and PURCHASE operations are shown in Listing 1. (The code for the ticket reservation objects is stubbed

Applied Reasoning

www.appliedreasoning.com

out to conserve space.) The StoreServlet first creates a new instance of the XMLProcessor object. The complete listing may be obtained from www.JavaDevelopersJournal.com. It then calls the initParser() method on the XMLProcessor class. The servlet's input stream is passed in as a parameter for parsing the XML document that the servlet received in its input stream. The XMLProcessor class is shown in Listing 2. The code for XMLParser.java is not discussed here, as it's similar to the code in the September article.

Next, the method getOperationType() is called on the XMLProcessor reference (xp) to check what kind of an operation was invoked on the StoreServlet. If the type is a "Purchase," the method processPurchase() is called on the object xp. This method returns a PurchaseOrder object from the XML document. This object is passed to the doPurchase() method, which in turn calls the doPurchase() method on the clientManager_. If the type is "Lease," the method processLease() is called on the object xp. Following this, the local method doLease() is called, which in turn calls the method doLease() on the clientManager_.

This is a very simplified version of an order. A real order would have greater detail, including telephone number and method of shipment.

XMLProcessor

The following methods were added to this class as compared to the September listing:

```
processPurchase()
processLease()
processConfirmation()
```

The complete code for these methods is given in Listing 2. The remainder of the listing was provided in September and thus is not repeated here. One new operation used in the parsing is the processListTag() utility method, which is used to obtain an array of item objects from the XML input file (submitted via the POST). This is used in both processLease() and processPurchase() methods. The processConfirmation() method is relatively simple in comparison to the processTicketQuote() method described in September. The complete listing for this file is available at www.JavaDevelopersJournal.com.

The next four classes – Item, PurchaseOrder, LeaseOrder and Confirmation – are shown in Listing 3.

Item

This class encapsulates an item. Basically, it consists of the following data fields and the getter and setter methods for them:
• *name* • *sku* • *quantity*

PurchaseOrder

This class encapsulates an item purchase order. It consists of the following fields and the corresponding getter and setter methods:
• *list of Items* • *customerName* • *address*

LeaseOrder

This class encapsulates an item leasing order. It consists of the following fields and the corresponding getter and setter methods:
• *list of Items* • *customerName*
• *address* • *referenceNo*

The LeaseOrder class inherits its first three fields from PurchaseOrder. The last field is the reference number of the associated tickets purchased.

Confirmation

This class encapsulates the result of an order for a purchase or a lease. It consists of the following fields and the corresponding getter and setter methods:
- status (confirm/reject)
- reference number

The last four classes – StoreClientManager, RMIStoreService, RMIStoreClient and RMIStoreServer – are shown in Listing 4.

StoreClientManager

This class receives a PurchaseOrder or a LeaseOrder from the StoreServlet, forwards it to the RMIStoreClient, gets back a Confirmation object and sends it back to the StoreServlet. It's also responsible for actually placing the order or arranging for the lease. This functionality is stubbed out in the application. It has two main methods – doPurchase() and doLease(). The doLease() method delegates the operation to the RMIStoreClient's doLease() method.

RMIStoreService

This is the interface for the services offered by the Application Services tier. It provides a single method, getLeaseAvailability().

RMIStoreClient

This is accessed by the StoreServlet for submitting the request to a Ticket Server. It provides the stub for the method getLeaseAvailability().

RMIStoreServer

This class processes a PurchaseOrder (or LeaseOrder) object and sends back a Confirmation object. It implements the method getLeaseAvailability(). The code always returns a "true" for availability. In a real application this would go against a local availability engine that would check whether the item was available in the store.

Running the Programs

The StoreServlet can be accessed via any client that can post the corresponding file for either the LEASE or the PURCHASE operation. I've included the files required to access the servlet from ColdFusion. Readers may use HTML, Java clients or VB clients. The code for this article is available at www.JavaDevelopersJournal.com. The code was compiled and tested on a Windows NT 4.0 workstation. To run the programs you'll need the following:
- JDK 1.1.x
- JSDK 2.0 (Java Servlet Development Kit)
- Your servlet engine and Web server

Conclusion

Developing this application has been a fun task for me. My intention was to provide a template for building *n*-tier applications in e-commerce. In essence, the idea was to decouple the Web storefront from the back-office services. If you undertake the development of such applications, the business logic for accessing back-office services may reside in middleware components written in EJB or COM. In that case you may end up using two categories of application servers – the Web storefront application server like ColdFusion and a middleware application server such as Netscape Application Server or WebLogic's application server.

As mentioned earlier, this article series has generated a lot of feedback from readers – so much so that I plan to include the concepts discussed here in a book I'm currently writing. I hope it's been a useful exercise for all of you too.

ASIDE FROM THE AUTHOR

I'd like to bring up a few interesting issues related to this article series. For one thing, the first article generated more feedback than I expected. It's been a pleasant surprise to know that some companies actually have airline store sites developed in ColdFusion similar to the fictitious one we've developed here. I also received some e-mail that indicated folks took this application more seriously than I thought they would. These readers concluded that this is a real-world application. While the design of the application outlines a real-world architecture for a business problem and serves as a template for a real-world solution, it's not a production-level system.

Another interesting development was that LiveSoftware (the makers of JRun) was acquired by Allaire Corporation (the makers of ColdFusion). This happened during the week of June 14, when JavaOne was in progress. It was my pleasure to talk to Jeremy Allaire (ColdFusion) and Paul Colton (JRun) at JavaOne regarding this development. In my opinion this has been a very smart move on Allaire's part. A few months ago Allaire acquired Bright Tiger Technologies, a company that specializes in clustering technology. The acquisition of JRun makes Allaire's app server story complete. ColdFusion is a great product and I've always felt they needed a story on the Java side. That was part of the motivation for building the Online Ticket Store. Coincidentally, the August issue of ColdFusion (Vol. 1, issue 4) has two features with two individual CF_SERVLET tags – the professional one from LiveSoftware and my own humble contribution.

Author Bio

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

ajit@sys-con.com

Listing 1: StoreServlet.java

```
import java.io.*;
import java.util.*;
import java.text.*;
import java.rmi.*;
import javax.servlet.http.*;
import javax.servlet.*;

public class StoreServlet extends
HttpServlet
{

  protected HttpServletRequest request_
  = null;
  protected HttpServletResponse
  response_ = null;
  protected StringBuffer result_ = null;

  XMLProcessor xp_ = null;
  PrintWriter toClient_ = null;
  StoreClientManager clientManager_ =
  null;
  Confirmation confirmation_ = null;

  // handle POST for servlet
  public void doPost (HttpServletRequest
  request,
           HttpServletResponse response)
    throws ServletException, IOException {

 request_ = request;
    response_ = response;

    toClient_ = new
PrintWriter(response_.getOutputStream());

    // Begin HTML
    toClient_.println("<HTML>");

 if (clientManager_ == null) {
  clientManager_ = new StoreClientManag-
  er(); }

 result_ = new StringBuffer();

 // Extract the arguments into local
 // variables
 processParameters();
 toClient_.println(result_.toString());
    toClient_.println("<HTML>");
    toClient_.flush(); }

  private void processParameters () {
    //XMLProcessor for converting XML
    //to Java
    xp_ = new XMLProcessor();

     try {
      InputStream in = request_.get
      InputStream();
      xp_.initParser(request_.getInput
      Stream(), toClient_); }

     catch (IOException ioe) {
ioe.printStackTrace(); }

   LeaseOrder leaseOrder = null;
   PurchaseOrder purchaseOrder = null;
      String queryType = xp_.getQuery-
      Type();

    if (queryType.equals("Lease")) {
       leaseOrder = xp_.processLease();
        confirmation_ = doLease(lease-
        Order); }

    else if (queryType.equals("Pur-
    chase")) {
        purchaseOrder = xp_.processPur-
        chase();
        confirmation_ = doPurchase(pur-
        chaseOrder); }

 else {
  result_.append("<HI>ERROR:: Unknown
  query type");
     return; }

    String xmlConfirmation =
    xp_.processConfirmation(confirmation_);
 result_.append(xmlConfirmation);
  }

  private Confirmation doLease(Lease-
  Order leaseOrder) {
 try { return
clientManager_.doLease(leaseOrder); }
 catch (Exception e) { return null; }
  }

  private Confirmation doPurchase(Pur-
  chaseOrder purchaseOrder) {
 try { return clientManager_.doPur-
 chase(purchaseOrder); }
 catch (Exception e) { return null; }
  }
```

Listing 2: .java: XML processor for the Online Store

```
<!-- Confirm.txt -->

<?xml version="1.0"?>
<Confirmation Type="Ticket">
  <ConfirmationNo>"SA2345678"</Confirma-
   tionNo>
</Confirmation>

<!-- Lease.txt>

<?xml version="1.0"?>
<Lease>
  <Passenger>
    <Name>
      <LastName>"Kent"</LastName>
      <FirstName>"Clark"</FirstName>
    </Name>
  </Passenger>
  <Flight>
   <ReferenceNo>"SA123456"</ReferenceNo>
   </Flight>
    <ItemList>
      <Item>
       <ItemName>"CD Player"</ItemName>
       <ItemId>cd00321</ItemId>
       <Quantity>1</Quantity>
      </Item>
      <Item>
       <ItemName>"Book"</ItemName>
       <ItemId>bookx453</ItemId>
       <Quantity>1</Quantity>
     </Item>
   </ItemList>
 </Lease>


<!-- Purchase.txt>

<?xml version="1.0"?>
<Purchase>
 <Passenger>
  <Name>
    <LastName>"Kent"</LastName>
    <FirstName>"Clark"</FirstName>
  </Name>
  <Address>
    <Street>"123 Tiny Lane"</Street>
    <City>"Smallville"</City>
    <State>"Kansas"</State>
    <Country>USA</Country>
    <ZIP>12345</ZIP>
   </Address>
  </Passenger>
    <ItemList>
      <Item>
       <ItemName>"CD Player"</ItemName>
          <ItemId>cd00321</ItemId>
          <Quantity>1</Quantity>
      </Item>
      <Item>
          <ItemName>"Book"</ItemName>
          <ItemId>bookx453</ItemId>
          <Quantity>1</Quantity>
      </Item>
    </ItemList>
</Purchase>

}
```

Listing 3: Item.java, PurchaseOrder.java, LeaseOrder.java, Confirmation.java

```
import java.io.*;

public class Item implements Serializable {

 private String name_ = null;
 private String sku_ = null;
 private int quantity_ = 0;

 public Item (String name, String sku,
 int quantity) {
  name_ = name;
  sku_ = sku;
  quantity_ = quantity; }

 public String getName () { return
 name_; }
 public String getSKU () { return sku_;
}
 public int getQuantity () { return
 quantity_; }
}

// PurchaseOrder.java
import java.io.*;

public class PurchaseOrder implements
Serializable {
 private Item[] items_ = null;
 private String customerName_ = null;
 private String address_ = null;

 public PurchaseOrder (Item[] items,
           String customerName,
           String address) {
   items_ = items;
   customerName_ = customerName;
   address_ = address;
 }
 public Item[] getItems () { return
 items_; }
 public Item getItem(int index) { return
 items_[index]; }
 public String getCustomerName () {
 return customerName_; }
 public String getAddress () { return
 address_; }
}

// LeaseOrder.java
import java.io.*;

public class LeaseOrder extends Pur-
chaseOrder implements Serializable {

 private String referenceNo_ = null;

 public LeaseOrder (Item[] items,
        String customerName,
        String address,
        String referenceNo) {

  super(items, customerName, address);
  referenceNo_ = referenceNo;
 }
 public String getReferenceNo () {
return referenceNo_; }
}

// Confirmation.java
public class Confirmation {
 String type_ = null;
 String confirmationNo_ = null;

 public Confirmation (String type,
String confirmationNo) {
  type_ = type;
  confirmationNo_ = confirmationNo;
 }

 public String getType () { return
type_; }
 public String getConfirmationNo () {
return confirmationNo_; }

}
```

Listing 4: StoreClientManager.java, RMIStoreService.java, RMIStoreClient.java, RMIStoreServer.java

```
public class StoreClientManager {

 RMIStoreClient rmiClient_ = null;

 public StoreClientManager () {
    if (rmiClient_ == null)
        rmiClient_ = new RMIStore-
Client();
 }

 public Confirmation doLease (LeaseOrder
 leaseOrder)
   throws Exception {
  Boolean result =
rmiClient_.getLeaseAvailibility(lease-
Order);

  if (result.booleanValue())
    return new Confirmation("Lease",
    "AS1234");
  else
    return new Confirmation("Lease",
"NIL");
 }

 public Confirmation doPurchase (Pur-
 chaseOrder purchaseOrder)
   throws Exception {
  return new Confirmation("Purchase",
"AS1234");
 }
}

// RMIStoreService.java
import java.rmi.*;

public interface RMIStoreService extends
Remote {
  public static final String
SERVER_NAME =
"rmi://t8000x321/servlet/rmiStoreServer";
  public Boolean getLeaseAvailibility
(LeaseOrder leaseOrder) throws
RemoteException;
}

// RMIStoreClient.java
import java.rmi.*;

public class RMIStoreClient implements
RMIStoreService {

  private RMIStoreService server_ =
  null;

  public RMIStoreClient() { connect
  ToServer(); }

  public Boolean getLeaseAvailibility
(LeaseOrder leaseOrder) throws
RemoteException {
 Boolean result = new Boolean(false);
 try { result = server_.getLeaseAvaili-
 bility(leaseOrder); }
 catch (Exception re) { re.printStack-
 Trace(); }
    System.out.println("result = " +
    result);
 return result;
  }

  public void connectToServer () {
 if (System.getSecurityManager() ==
 null)
   System.setSecurityManager(new RMISe-
   curityManager());

    try { server_ =
(RMIStoreService)(Naming.lookup(RMIS-
toreService.SERVER_NAME)); }
 catch (Exception e) { e.printStack
 Trace(); }
  }
}

// RMIStoreServer.java
import java.rmi.server.*;

public class RMIStoreServer
  extends UnicastRemoteObject implements
  RMIStoreService {

  public String RMI_SERVER_NAME =
"rmi://t8000x321/servlet/rmiStoreServer";

  public RMIStoreServer() throws Remote-
Exception {
 super();
  }

  public Boolean getLeaseAvailibility
(LeaseOrder leaseOrder) throws
RemoteException {
 return new Boolean(true);

  }

 public static void main (String[] args) {
 System.setSecurityManager(new RMISecu-
 rityManager());
 try {
      RMIStoreServer server = new RMIS-
      toreServer();
   Naming.rebind(server.RMI_SERVER_NAME,
   server);
   System.out.println("RMIStoreServer
   ready ..."); }
 catch (Exception e) { e.printStack-
 Trace(); }
  }
}
```

How Not to Trip Over Your Own Footprint!

Minimizing the memory footprint to suit a range of environments

WRITTEN BY
JEFF RICHEY AND
JEFF SCROGGIN

Today developers are creating a full spectrum of Internet applications and systems ranging from enterprise servers to handheld devices that manifest a number of unique requirements. Although these applications and systems are commonly written in Java, they have different footprint requirements depending on the platform they run on. However, they need to have the same look and feel regardless of where they're deployed.

This article will describe the technical considerations that enable an application or system to run on a broad range of platforms. Minimizing the memory footprint to suit a range of environments, from network servers to PDAs and other devices, is the core of this issue. Applications that require a minimum memory footprint will benefit from two important factors: (1) the JVM (Java Virtual Machine) class loader and (2) a development strategy for building applications or systems using Java "factories."

How to Build Dynamic Footprint Applications and Systems

The JVM class loader has the ability to load and unload from memory the subset of class files needed at any given time by the application or system (see Figure 1). By contrast, Algol-based applications, including those written in C or C++, require the entire executable to be brought into memory in order for any part of the application or system to execute, even if only a subset of the functionality is required at runtime (see Figure 2). This is an important and significant difference that can dramatically impact the runtime memory footprint.

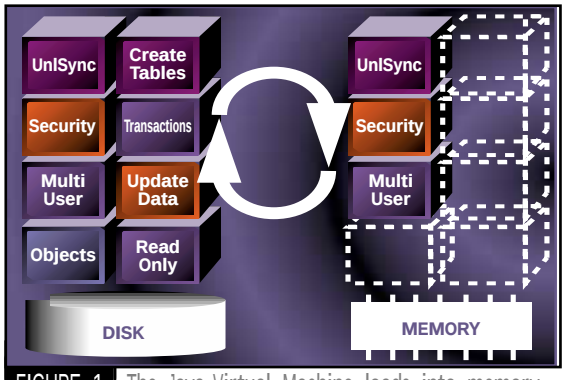


FIGURE 1 The Java Virtual Machine loads into memory only that subset of class files needed by the application at runtime.

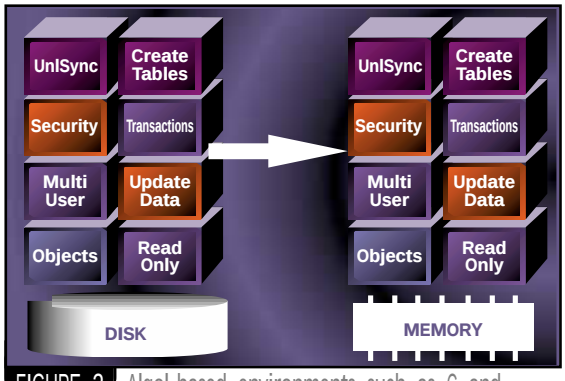


FIGURE 2 Algol-based environments such as C and C++ load an entire application into memory, even if only a subset of the functionality is required at runtime.

To achieve the greatest possible benefit from the JVM class loader, PointBase has employed Java "factories" in the design and architecture of its database management system. Using factories allows developers to minimize class file size (and disk requirements) by including only those factories needed by the application. More significant, a disciplined implementation of factories in the database ensures a minimum memory footprint – the JVM class loader keeps in memory only those classes needed at any one time by the application.

Using the discipline of factories and the JVM class loader allows you to create dynamic footprint applications and systems that won't require special tailoring by an end user to meet limited memory requirements, especially for memory-constrained devices and systems such as PDAs.

Code Snippets

The following code snippets provide a simple example of the method that PointBase has used to create Java factories. This particular section of code demonstrates the compilation factory for a SQL Create Table statement.

```
public class createTableCompilation-
Factory
{
public compilerInterface getCompiler(
)
{
 return new createTableCompiler( );
}
public definerInterface getDefiner( )
{
 return new createTableDefiner( );
}

public compilerInterface getExecu-
tion( )
{
 return new createTableExecution( );
}

public parserInterface getParser( )
{
 return new createTableParser( );
}
}
```

The methods in this class are invoked when an SQL Create Table statement is executed in the application.

With most SQL statements a database management system needs to parse the statement, check definition information in the system catalogs and then compile it into an internal format for execution. With PointBase, when an SQL statement is encountered the appropriate factory will be used to invoke constructors that cause the parsing, definition, compilation and execution aspects of an SQL statement. This approach allows you to separate unassociated functionality, and dynamically limit or add functionality without causing linking problems commonly encountered with other development languages.

Benefits of Maintaining Applications Written Using Java Factories

Algol-based applications and systems don't have the ability to dynamically load objects. The entire executable must be loaded into memory in order for even one object to execute. Additionally, most Algol-based languages don't support or allow the factory concept.

Many developers may attempt to circumvent the restrictions of Algol-based applications by recompiling the application for each environment with only those features that they

AUTHOR BIO

Jeff Richey, vice president of engineering and cofounder of PointBase, is a recognized leader in database product development. Jeff has over 15 years of database experience, working as a core architect and development manager for IBM/DB2, HP, Oracle and Sybase. He is a patent holder of two key innovations in SQL performance.

Jeff Scroggin, director of marketing at PointBase, is an experienced marketing manager with over 10 years working in the data management and applications market segments. His assignments have included product marketing, product management and business development responsibilities.

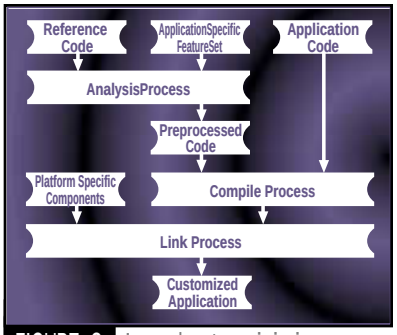


FIGURE 3 In order to minimize memory footprint in Algol-based environments, developers must commonly follow a complex series of steps to create a customized version of the application containing only the necessary functionality.

need. For example, some C-based database systems require the user to go through numerous steps with their application and database in order to generate a reduced runtime footprint (see Figure 3).

This methodology has a significant limitation when the application needs to be modified as these steps must be repeated and the application redeployed to all installations. This approach is extremely expensive and labor intensive for the application developer and the end user who must redeploy the application.

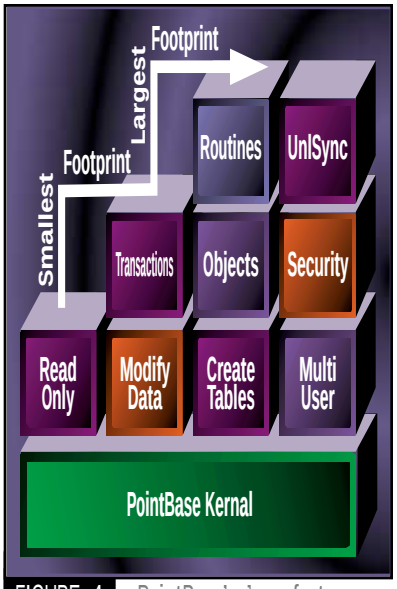


FIGURE 4 PointBase's Java factory design implements SQL functionality as distinct class files or modules that the JVM class loader invokes automatically at runtime, providing the minimum database footprint for each application and installation.

However, Java dynamically adapts to the applications and systems without end-user intervention. Algol-based products require end users to adapt their applications and systems. This difference is fundamental and crucial to deploying and managing small footprint applications in the field.

PointBase and Dynamic Footprint

PointBase delivers an object-relational database management system written in 100% Pure Java. Developers who wish to embed the PointBase database system within their application typically include the classes from the PointBase JAR file in their own application's JAR file. The developer can then control and monitor the runtime memory and disk footprint of the entire solution, including the database (see Figure 4).

For customers who don't need the full range of database functionality, PointBase's Java factory architecture provides a simple, automated way to customize the database for each application. For example, many applications don't need SQL security (privileges). Others, such as "palm" applications, only need SQL DML (Delete, Insert, Update), SQL Queries (Select) and Transaction management (Commit and Rollback) functionality. PointBase supports a wide range of application requirements and will dynamically minimize the footprint for a full range of applications and environments.

jeff.scroggin@pointbase.com jeff.richey@pointbase.com

Components, and Creating a Custom Property Editor

Adding more features to the CodeDocument class

WRITTEN BY
JIM CRAFTON

This column discusses property editors and how to implement one for Java – specifically, how to make one work for our CodeDocument class. When last we talked, we saw how to build a CodeDocument class, but it wasn't something we could work with in a visual designer like JBuilder, say, or Visual Café. In this column we'll build a special case of the JTextArea component and add some specialty properties and property editors to support the CodeDocument class we worked on before. The code listings at the end of the article are excerpted from the complete code, which you can download from www.JavaDevelopersJournal.com.

Property Editors and How They Work

Basically, property editors allow you to edit a property. Wow! What a concept! Except that to make this work the IDE has to know what the property is. Luckily, since we're talking about Java, we have a neat little tool called Reflection that we can use to ask any given object, "What are you?" and get back a response like, "I'm a java.lang.String class object." Great, now we know what we're dealing with…but wait a minute…hold the bus. How do we know how to edit the object? "It's a string. How hard can it be?" you say. Well, if everything were a string, we'd have no problems, but things are a little more complicated than that. Granted, a string is easy to edit, but what about a color object, or a collection of items (like a vector)? Since a property can be any kind of object, we need to have a uniform way of explaining to the IDE how to edit that object, and, if necessary, supply a custom UI with which to edit the property. So the property editor interface provides just the thing we need, with methods to get and set the object in question, as well as methods for telling whether the property editor supports a custom UI editor.

Property editors by themselves don't do much good. For the IDE to "know" about them, the component whose properties we're interested in must publish BeanInfo. The BeanInfo interface describes the component to the IDE with information such as the icon to display as a visual representation of the component and – you guessed it – a list of property descriptors. What do the property descriptors do? They describe properties! Each property descriptor contains information such as the name of the get and set methods, the name of the property, the class the property belongs to and, finally (and this is optional), the PropertyEditor class itself. Thus, when the user asks to edit a property, the IDE first retrieves the component's BeanInfo class and then the property descriptor for the desired property. If the property descriptor supports a custom editor, the custom editor is then loaded and the custom editor component is loaded and displayed for the user to work with (this usually takes place in a dialog of some sort – at least this is how JBuilder handles things).

Adding Our Own Property Editor for the CodeDocument

Let's take this knowledge and do something practical with it. Let's add a new component based on JTextPane, which automatically uses a Document model based on our CodeDocument class. We'll start with the code shown in Listing 1.

All right, that was pretty painless. What we're doing is calling the super class in the constructor and setting the document to a new instance of the CodeDocument class we developed previously. We're going to add a single property – call keywords – that will allow the developer to modify the CodeDocument keywords attribute. Doing this changes the set of keywords the CodeDocument looks for in its syntax highlighting process. Let's look at another example demonstrating this, in Listing 2.

So now we've added two methods that allow us to get and set keywords. We overloaded the setKeywords method to allow for either a vector of keywords or an array of strings (the array of string method simply converts the array to a vector and calls the vector version of the method). This will become more important when we add our property editor later on. Notice that we didn't add an attribute of type Vector to the code – we're simply mapping any calls to get or set keywords directly back to the CodeDocument class. Now we have our component, and if we wanted to we could leave it at that. However, we wouldn't have a property editor working yet, thus defeating the point of this article!

The next thing we need to add is the BeanInfo class. BeanInfo is actually an interface that you need to implement in some other class to make things work. So let's create our BeanInfo implementation class, as seen in Listing 3.

So far this is pretty basic. As mentioned before, the BeanInfo methods allow an IDE to determine all sorts of information about the component and the properties it supports. In our implementation you'll notice we extend SimpleBeanInfo, a class that has all the methods of the BeanInfo stubbed out. Since there are a few other methods that aren't neccessary for us at this point, extending SimpleBeanInfo makes our life a little easier, allowing us to focus on just the parts we need. As mentioned earlier, getPropertyDescriptors() is the method called by the IDE to get all the property info. We'll return an array of property descriptors only for the properties we're interested in exposing to the IDE for user manipulation. In our case there will be only one element, for the keywords property. The getIcon() method will return either null or an image containing a 16x16 icon (black and white or color) that represents the component or a 32x32 icon (also black and white or color). The kind of icon is determined by the iconKind parameter passed into the method (the choices are ICON_COLOR_16x16, ICON_COLOR_32x32, ICON_MONO_16x16, and ICON_MONO_32x32). The getAdditionalBeanInfo() is supposed to call the super class and request more bean info from that class.

For now, let's look at the getPropertyDescriptors() method. In our implementation we need to supply only one element in the array. We don't need to worry about any of the super class properties. If we had another property that was a string – say, DocumentName – we wouldn't have to worry about it either. Why? Because objects like string and integer almost always have default property editors registered in the IDE and there's usually no need to register something else. But let's say that,

instead of DocumentName, our property was called FileName, and we wanted to bring up some kind of file open dialog when the user tried to edit it. Now we'd want to supply a custom property editor for the string, causing a file open dialog to pop up when the property is edited. Back to our implementation. To set up the element properly, let's look at Listing 4.

The first thing we do is create a new instance of a PropertyDescriptor class by passing in the name of the property, the class it belongs to (in our case CodeTextPane) and the names of the get and set methods. The setDisplayName() method is used to define the text that will be displayed for your property, and the setShortDescription is used to populate tool tips with a short description of your property. The setPropertyEditorClass() method is used to set the class type for your property editor. It's important to set this up correctly; otherwise the property editor won't be picked up by the IDE. The class KeywordsEditor is the one we're going to create next, in case you're wondering where it came from. After this is done we'll create a new array of PropertyDescriptor objects (the propertyDescriptors variable) and put the new instance we created previously into the array. Then we just return the array.

The only other method we'll bother implementing in our implementation of BeanInfo is the getAdditionalBeanInfo() method. This is given in Listing 5.

By providing additional BeanInfo, we're giving the IDE more information about our object – specifically, about the object's super class. Similar to the PropertyDescriptor, we're returning an array of objects, each a reference to BeanInfo. In our implementation we call the Introspector's getBeanInfo() method to get the super class's bean info.

So far we've gone through all the steps neccessary to create a simple component (CodeTextPane) and supply an implementation of BeanInfo that properly exposes our property (keywords) and its custom editor class (KeywordsEditor). Now it's time to look at the property editor class directly. The class will actually have two parts: the first is the property editor class itself, KeywordsEditor (Listing 6a); the second is the custom editor component, KeywordsEditorComponent (Listing 6b).

The property editor interface is fairly easy to implement, especially when we extend from the PropertyEditorSupport class. Like the SimpleBeanInfo, it provides either stubs or simple implementations of the methods so we can concentrate on the few that are important to us. Since we want to support a custom editor component, we return true from the supportsCustomEditor() method that's called by the IDE. We also return an instance of our editor component in the getCustomEditor() method. Since getCustomEditor() returns a component, we can base our editor component on practically anything. Usually the component gets nested into some other container, like a dialog (this is the case with JBuilder), so basing it on JPanel is usually a good idea. The getJavaInitializationString() method returns a string that represents the Java code required to set the given property from the code editor. Whenever you change the property via your custom editor, this string will need to change as well. An example would be a property that is a font object. In your IDE's code editor, you might see something like this:

```
public void myInitMethod(){
  //more code
  this.setFont(new Font("SansSerif", 1, 12));
  //more code...
}
```

The line "new Font("SansSerif", 1, 12)" would be provided to the IDE by whatever property editor is registered for the font property of the object. This string would be the return of the property editor's getJavaInitializationString() method. In our case the string might look like the following:

```
public void myInitMethod(){
  //more code
  codeTextPaneObj.setKeywords(new
String[]{"foo","bar","blah",});
  //more code...
}
```

Since there's no convenient way to represent a vector like this, we can use our second version of the setKeywords() method to pass in the new set of keywords. The implementation code for the getJavaInitializationString() method would look like Listing 7.

The getValue() method returns the current value of the property represented by the property editor (it's already implemented for us by the PropertyEditorSupport class). We then go through all the elements in the vector and assemble them into a string, returning this string when we're finished.

With this done we have our PropertyEditor class ready to go. Now we can take a look at designing the custom editor component. For starters let's take a look again at Listing 6b. The class is simple. It extends Panel (or JPanel if you want to have a Swing look) and has a constructor that takes one argument – a property editor object that's set to a private variable. As it stands, the component will do absolutely nothing, so let's add a listbox to view all our keywords, an edit box to type in new keywords, two labels – one for the list and one for the edit box – and, finally, two buttons – one to add the information in the edit box and one to delete selected items from the keywords list. Take a look at Listing 8.

The keywords variable is a DefaultListModel, Swing's default implementation of a suitable list model for use in JList controls. So when we add items to the list, we don't bother calling the control methods; instead, we actually manipulate the keywords object. The JList is constructed by passing in the keywords object as its model, so everything is hooked up for us. The next thing we need to add is a private initialization method to set all the widget properties and hook up event listeners, as well as to configure a GridBagLayout to position all the controls properly. We'll call this method initEditor(), and instead of showing all the code here (you can see it on the ***JDJ*** Web site), I'm just going to present the sections relevant to the article. The main thing we need to catch are changes to the keywords list. Each time it changes, we need to call the property editor's setValue() method to inform it of a change, which in turn notifies any registered listeners that the property has changed. That way, when our editor component is dismissed, the property is in a valid state and the getJavaInitializationString() will return correct data. To accomplish this we register a listener with the keywords object as seen in Listing 9 (this takes place in the initEditor() method).

This ensures that any change to the keywords object will call the keywords_changed() method (a private method of the editor component), which properly updates the property editor. According to the Java documentation, you shouldn't actually modify the property editor value directly; instead, you should just send a new instance. To do that we end up with the code in Listing 10.

The method simply makes a copy of the elements in the keywords list and places them in a new instance of a vector that can then be sent to the property editor via the setValue() method.

One other thing of interest in the initEditor() method is the setting of the listbox to the current value of the property editor. This is accomplished in Listing 11.

The keywords object is cleared using the removeAll() method, and the property editor's value is retrieved through the getValue() method and then enumerated; the new elements are then added to the keywords list.

That's It, Folks!

We're done!

- We've covered creating a new text component with support for our previously created CodeDocument class.
- We've created a BeanInfo class to ensure that IDEs such as JBuilder are aware of the properties we're exposing and any custom features, like editors, we support.
- We've created a property editor class to support our new property.
- We've created a custom editor component that will allow us to edit that property in a visual way.

Whew! I hope this was helpful – let me know how it works for you by e-mailing me at ddiego@diegoware.com.

Author Bio

Jim Crafton is part of the R&D team at Improv Technologies (www.improv-tech.com), helping to develop a new production-quality animation tool. In his spare time he develops advanced graphics software (you can see it at www.diegoware.com).

ddiego@diegoware.com

Listing 1

```
package CodeEditor;

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

public class CodeTextPane extends JTextPane {
  public CodeTextPane() {
    super();
    //now we set our Document model
    setDocument(new CodeDocument());//no keywords at this
point
  }
}
```

Listing 2

```
public class CodeTextPane extends JTextPane {
  public CodeTextPane() {
    super();
    //now we set our Document model
    setDocument(new CodeDocument());//no keywords at this
point
  }

  public void setKeywords(Vector aNewSetOfKeywords){
    Document doc = getDocument();
    if (doc instanceof CodeDocument) {
      CodeDocument codeDoc = (CodeDocument) doc;
      CodeDoc.setKeywords(aNewSetOfKeywords);
    }
  }

  public void setKeywords(String aNewSetOfKeywords[]){
    Vector v = new Vector();
    for (int i=0;i< aNewSetOfKeywords.length;i++){
      v.addElement(aNewSetOfKeywords[i]);
    }
    setKeywords(v);
  }
  public Vector getKeywords(){
    Vector keywords = null;

    Document doc = getDocument();
    if (doc instanceof CodeDocument) {
      CodeDocument codeDoc = (CodeDocument) doc;
      keywords = CodeDoc.getKeywords();
    }

    return keywords
  }
}
```

Listing 3

```
package CodeEditor;

import java.beans.*;
import java.awt.*;

public class CodeTextPaneBeanInfo extends SimpleBeanInfo {

  Class theBeanClass = CodeTextPane.class;

  public CodeTextPaneBeanInfo() {
  }

  public PropertyDescriptor[] getPropertyDescriptors() {
    return null;
  }

  public Image getIcon(int iconKind) {
    return null;
  }

  public BeanInfo[] getAdditionalBeanInfo() {
    return null;
  }
}
```

Listing 4

```
public PropertyDescriptor[] getPropertyDescriptors() {
    try  {
      PropertyDescriptor keywordsProperty = new PropertyDe-
scriptor("keywords", theBeanClass, "getKeywords", "setKey-
words");
      keywordsProperty.setDisplayName("keywords");
      keywordsProperty.setShortDescription("keywords");
      keywordsProperty.setPropertyEditorClass(CodeEditor.Key-
wordsEditor.class);

      PropertyDescriptor[] propertyDescriptors =
        new PropertyDescriptor[] {keywordsProperty};
      return propertyDescriptors;
    }
    catch (IntrospectionException ex) {
      ex.printStackTrace();
      return null;
    }
  }
```

Listing 5

```
public BeanInfo[] getAdditionalBeanInfo() {
    Class superclass = theBeanClass.getSuperclass();
    try  {
      BeanInfo superBeanInfo =
Introspector.getBeanInfo(superclass);
      return new BeanInfo[] { superBeanInfo };
    }
    catch (IntrospectionException ex) {
      ex.printStackTrace();
      return null;
    }
  }
```

Listing 6a: The Property Editor

```
package CodeEditor;

import java.awt.*;
import java.beans.*;

public class KeywordsEditor extends PropertyEditorSupport {
  private KeywordsEditorComponent editorComponent;


  public KeywordsEditor() {
  }

  public boolean supportsCustomEditor() {
    return true;
  }

  public Component getCustomEditor() {
    if (editorComponent == null) {
      editorComponent = new KeywordsEditorComponent(this);
    }
    return editorComponent;
  }

  public String getJavaInitializationString() {
```

```
    return "";
  }
}
```

Listing 6b: The Editor Component

```
package CodeEditor;

import java.awt.*;
import java.beans.*;

public class KeywordsEditorComponent extends Panel {
  private PropertyEditor propEditor;


  public KeywordsEditorComponent(PropertyEditor anEditor) {
    this.propEditor = anEditor;
  }
}
```

Listing 7

```
public String getJavaInitializationString() {
 Vector v = (Vector)getValue();
 String s = "new String[] {";
 Enumeration enum = v.elements();
 while (enum.hasMoreElements()){
   String keyword = (String)enum.nextElement();
   if ( enum.hasMoreElements() ){
     s += "\"" + keyword + "\",";
   }
   else{
     s += "\"" + keyword + "\"}";
   }
 }
 return s;
}
```

Listing 8

```
public class KeywordsEditorComponent extends Panel{
  private PropertyEditor editor;
  JLabel label1 = new JLabel();
  JLabel label2 = new JLabel();
  DefaultListModel keywords = new DefaultListModel();
  JList keywordList = new JList(keywords);
  JTextField keywordEdit = new JTextField();
  JButton addBtn = new JButton();
  JButton deleteBtn = new JButton();
  JScrollPane listScroller = new JScrollPane(keywordList);
```

Listing 9

```
keywords.addListDataListener(new ListDataListener(){
    public void contentsChanged(ListDataEvent e){
      keywords_changed(e);
    }
    public void intervalAdded(ListDataEvent e){
      keywords_changed(e);
    }

    public void intervalRemoved(ListDataEvent e){
      keywords_changed(e);
    }
  });
```

Listing 10

```
private void keywords_changed(ListDataEvent e){
  if (editor != null){
    Vector v = new Vector();
    Enumeration enum = keywords.elements();
    while (enum.hasMoreElements()){
      v.addElement(enum.nextElement());
    }
    editor.setValue(v);
  }
}
```

Listing 11

```
if (editor != null){
  keywords.removeAllElements();
  Vector v = (Vector)editor.getValue();
  Enumeration enum = v.elements();
  while (enum.hasMoreElements()){
    keywords.addElement(enum.nextElement());
  }
}
```

Real-Time Web-Based Applications with Java and CORBA

CORBA's idea of separating interface from implementation is well suited to Java

WRITTEN BY
ROLF KAMP AND
THOMAS CZERNIK

Common Object Request Broker Architecture and Java are among the newest emerging technologies revolving around IP and Internet applications. The CORBA specification defines an industry-wide standard infrastructure that simplifies the integration of software systems using object-oriented techniques. CORBA separates architecture and implementation from interface specification, allowing clients and servers to be implemented in any language, on any platform.

Java is an excellent, network-savvy, object-oriented language that's well suited for implementing CORBA components. Java's object-oriented paradigm separates an object's interface from its implementation, as does CORBA. Its machine independence and wide availability allows CORBA objects to execute on any platform running a Java Virtual Machine. Today JVMs are available on many systems, from mainframes to microprocessors. This combination of CORBA and Java creates an optimal environment for implementing real-time Web-based applications requiring no client software except a browser.

CORBA Overview

CORBA's Interface Definition Language, used to define object interfaces, is where the separation between interface specification and implementation occurs. IDL is a strongly typed, declarative-only language resembling C++. IDL doesn't define object implementations, only object interfaces. An IDL compiler translates the IDL into a set of stubs and skeletons implemented in a target language such as Java. These stubs and skeletons are a set of Java interfaces and classes from which object implementations and complete applications are built.

Following is an example of IDL for the object called MyObject, which has one method called getTimeStamp.

```
module MyPackage {
  interface MyObject {
    string getTimeStamp();
  };
};
```

Running this IDL through an IDL-to-Java compiler generates the Java classes and interfaces containing stubs and skeletons in a Java package called MyPackage.

CORBA's Object Request Broker (ORB) locates objects, manages connections and communications between stubs and skeletons, and invokes object methods on behalf of the client. The stubs and skeletons perform various functions such as marshaling data and operations. The client invokes methods on remote objects via the stub interface. In turn, the server receives client requests through the skeleton. Stubs and skeletons can be implemented statically or dynamically.

The Dynamic Invocation Interface is an approach that permits clients to discover information about objects at runtime that may not exist at compile time. CORBA's Interface Repository can be queried to discover, at runtime, a remote object reference, its methods and method parameters. DII provides a greater level of flexibility than static IDL stubs. The latter, however, are simpler and offer better performance. The Dynamic Skeleton Interface is the skeleton equivalent of DII. DSI provides a runtime binding to an object that may not be known at compile time. DSI allows the server to determine a requested method's signature and implement that method at runtime. An object receiving a request doesn't know if the client request originated from a DII or a static IDL stub. Similarly, the client doesn't know if the object implementation fulfilling the request uses an IDL skeleton or DSI.

Implementing CORBA Applications

Two implementation approaches exist for building on these stubs and skeletons, the IS-A approach and the HAS-A approach. The IS-A, or ImplBase approach, builds on the stubs and skeletons using inheritance. The HAS-A, or Tie approach, builds on the stubs and skeletons using delegation. In the Tie approach the object implementation must implement the IDL interfaces. Since Java doesn't support multiple inheritance, a class using the ImplBase approach is limited to inheriting only from the stub and skeleton classes. If the object implementation must inherit from any class other than the ImplBase class, the Tie approach must be used.

- *Java implementation of MyObject using the ImplBase approach*:

```
class MyObjectImp extends _MyOb-
jectImplBase {
  public String getTimeStamp() {   //
  method body  }
}
```

- *Java implementation of MyObject using the Tie approach*:

```
class MyObjectTie implements _MyOb-
jectOperations {
  public String getTimeStamp() { //
  method body }
}
```

Objects are made available through a server process that's registered with the ORB. Note that registering a server with an ORB is vendor specific. The server initializes the ORB, instantiates the

CORBA objects and passes control to the ORB to handle incoming requests.

- *Server implemented in Java using ImplBase approach*:

```
org.omg.CORBA.ORB orb =
org.omg.CROBA.ORB.init(args,null);
//initialize the ORB

// instantiate the CORBA object
MyObject ObjectRef = new MyObject-
Impl();

orb.connect(ObjectRef);  // pass con-
trol to the ORB
```

- *Server implemented in Java using Tie approach*:

```
org.omg.CORBA.ORB orb =
org.omg.CROBA.ORB.init(args,null);
//initialize the ORB

MyObject ObjectRef =
new _tie_MyObject(new MyObjectTie());

orb.connect(ObjectRef); //pass con-
trol to the ORB
```

Clients initialize the ORB and obtain an object reference to the server by calling the bind method. Using this object reference, requests to the server can be made as if the object resided in the client's address space.

- *Client implemented in Java*:

```
org.omg.CORBA.ORB.init(this,null);

// bind to server
MyObject ObjRef =
MyObjectHelper.bind(":MyServer","host
name");

// executes method getTimeStamp on
server
String TheTime = ObjRef.getTime-
Stamp();
```

CORBA Callback Methods

Applications, such as real-time stock updates, inventory management and network surveillance, require clients to react in real time to changes or updates that occur in the server. Callbacks are a technique that makes it possible for the server to execute methods on the client as changes occur in the server.

Callbacks are a well-defined, easy-to-implement, effective technique for developing real-time, Web-based clients. When using callbacks, the client/server relationship is reversed. Clients wanting to receive real-time data register with the server by passing a client object reference to the server (see Figure 1). A proxy for the client's object is created in the server.

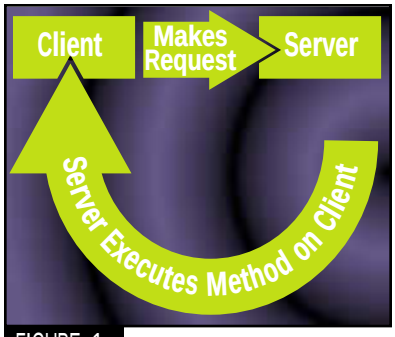


FIGURE 1

This proxy serves as a handle to the client that the server uses to invoke methods on the client. Servers wishing to use callbacks with many clients need to store the client object references as a data structure, such as Java's Vector class.

By default, when executing CORBA methods, callers block until the called method returns. When using callbacks, a deadlock situation can occur since clients may invoke server methods from within the callback method. To avoid this possible situation, callback methods should be qualified with the IDL keyword oneway. Calling objects don't block – instead, they continue immediately after invoking oneway methods. Using these methods, servers don't wait until a method completes and may continue to accept method invocations. For this reason oneway methods must have a return type of void and should not throw an exception.

The following IDL defines the interfaces for the objects MyCallback and MyObject. Clients register with a server by invoking MyObject's registerCallback and remove themselves from the server's client list by invoking MyObject's removeCallback. MyCallback declares the callback method receiveTimeStamp, which the server invokes to update the client's timestamp.

```
module MyPackage {
interface MyCallback {
  oneway void receiveTimeStamp(in
string servertime);
};

interface MyObject {
  string getTimeStamp();
  void registerCallback(in MyCallback
obj);
  void removeCallback(in MyCallback
obj);
};
};
```

The client initializes the ORB, binds to the server and registers by executing registerCallback with an argument of this (an instance of MyCallback). The server uses the MyCallback object reference as a proxy to the client.

```
class MyClient extends _MyCallback-
ImplBase {
public MyClient(){
  org.omg.CORBA.ORB.init(this,null);
  MyObject ObjRef =

MyObjectHelper.bind(":MyServer","host
name");
  ObjRef.registerCallback(this);
}
public void receiveTimeStamp(String
serverTime) {
  // method body
  }
}
```

The server's registerCallback method receives an instance of MyCallback and stores it in its instance variable client. The doCallback method executes receiveTimeStamp on the client by making reference to the client instance variable. The server may execute doCallback whenever it wants to update the timestamp on the client.

```
class MyObjectImp extends _MyOb-
jectImplBase {
MyCallback client;
public String getTimeStamp() {
  // method body
}
public void registerCallback(MyCall-
back obj) {
  client = obj;
}
public void removeCallback(MyCallback
obj) {
  // method body
}
public void doCallback(){
  // define method what will execute
on client
  client.receiveTimeStamp("10:10
AM");
}
}
```

Understanding Network Traffic Generated by CORBA

One key element to understanding performance in a CORBA system is the network traffic generated by method calls. Developers need to understand when they're using object references versus object instances. CORBA objects always reside on the server and are accessed by the client through methods defined in the IDL. Prior to CORBA 3.0, only object references could be passed between servers and clients. CORBA copies by value nonobjects such as Java primitive data types. An object's attributes are retrieved from the server by using an object's accessor (get) method.

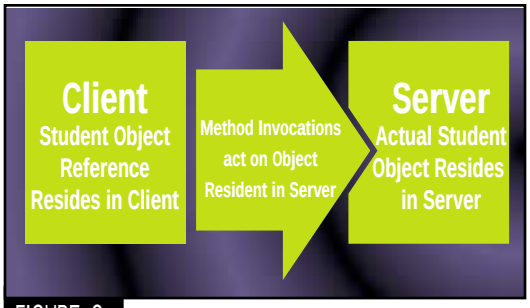


FIGURE 2

Each invocation of an accessor method generates network traffic. The amount of network traffic generated depends on the amount of data retrieved and the number of accessor methods invoked. If a client needs to make frequent reference to object attributes that don't change often, network traffic can be kept to a minimum by making a local copy of the attributes on the client. Callbacks shouldn't be used to update the client-side data.

The IDL compiler generates a pair of Java methods for IDL variables tagged with the attribute keyword. Each IDL attribute generates accessor and mutator (set) methods. The accessor method returns the value contained in the class variable. The mutator method is used to modify the value of the class variable. Only the accessor method is generated for IDL attributes marked as read-only. Method parameters may be marked with the IDL keyword in, out or inout. Parameters marked in are passed only from the client to the server and are viewed as immutable to the called method. The inout keyword is used to declare parameters modified by the server. Parameters marked inout are passed from the client to the server and from the server back to the client. The out keyword declares parameters returned to the client from the server. Since Java doesn't support passing out and inout parameters by reference, the IDL compiler generates a Java Holder class, which is instantiated to simulate passing parameters by reference. The best performance is achieved by using parameters marked with the in keyword.

Following are IDL-defining attributes that create a Java interface containing accessor and mutator methods. These methods are generated for the name and gpa attributes. Since the ID attribute is read-only, only an accessor method is created.

```
interface Student {
  attribute string name;
  attribute float gpa;
  readonly attribute long id;
};
```

```
interface MyObject {
  Student getStudent(in long idNum-
ber);
};
```

A developer implements the Student object by extending the IDL generated _StudentImplBase class. StudentImpl defines the attributes declared in the IDL as private instance variables, along with the accessor and mutator methods.

```
public class StudentImpl extends
_StudentImplBase {
private String name;
private float gpa;
private int id;
public StudentImpl(String pname, int
pid) {
  name = pname;
  id = pid;
}
public String name() {
  return name;
}
public void name(String value) {
  name = value;
}
public float gpa() {
  return gpa;
}
public void gpa(float value) {
  gpa = value;
}
public int id() {
  return id;
}
}
```

The following client code demonstrates the use of object references. After binding to MyServer, the client obtains a reference to the Student object by invoking MyObject's getStudent method. The execution of the student.id, student.-name and student.gpa methods generate network activity. Although invocations of these methods appear to operate on a local instance of class Student, they actually access the Student object that resides on the server (see Figure 2).

```
org.omg.CORBA.ORB.init(args, null);
MyObject ObjectRef =
MyObjectHelper.bind(":MyServer",
"hostname");

Student student = ObjectRef.getStu-
dent(1234);
System.out.println("Student's id: " +
student.id()
+ "Student's Name: " +
student.name());
student.gpa(3.9);
System.out.println("Student's gpa: "
+ student.gpa());
```

Using CORBA in a Web Browser

Java-enabled Web browsers act as a common platform from which CORBA applets may be launched. An ORBlet downloaded to the Web browser permits the execution of CORBA-enabled applets. Browsers that are CORBA enabled, such as Netscape's Communicator, provide efficient use of CORBA in applets since the need to download an ORBlet is eliminated. Netscape's Communicator contains Inprise's Visigenics ORB. If an applet requires an ORB other than the one contained in the Web browser, the required ORBlet must be downloaded when the applet is downloaded. The HTML <param> tag's name and value attributes direct the Web browser to use the downloaded ORB.

The following HTML code downloads the CORBA-enabled applet Grid.class contained in the file Grid.jar. IONA's OrbixWeb ORB is downloaded in the file OrbixWeb301.jar. The HTML <param> tags direct Netscape's Communicator to use IONA's ORB rather than Inprise's built-in ORB.

```
<HTML>
<BODY>
<APPLET CODE=Grid.class
ARCHIVE=Grid.jar,
OrbixWeb301.jar CODEBASE=java_output>
<param name="org.omg.CORBA.ORBClass"
value="IE.Iona.OrbixWeb.CORBA.BOA">
<param name="org.omg.CORBA.ORBSingle-
tonClass"
value="IE.Iona.OrbixWeb.CORBA.BOA">
</APPLET>
</BODY>
</HTML>
```

Typically, applets initialize the ORB and bind to the server in the applet's init method. Thereafter, applets may invoke server methods in response to user actions such as a button click. The server may run callback methods on the client anytime after the applet's init method completes. In this way clients receiving real-time updates from a server may be deployed requiring only a Java-enabled Web browser on the client.

Summary

CORBA's powerful concept of separating interface from implementation is well suited to Java. This concept allows software systems to communicate with each other without regard to the client's actual platform. Real-time distributed systems executing on a wide variety of clients can be developed using the techniques discussed here. These applications execute in the ubiquitous Java-enabled Web browser, reducing client-side administration.

rfk@att.com tjc@att.com

AUTHOR BIOS

Rolf Kamp is a senior technical staff member with AT&T's Operations Technology Center, working on network operation support systems. He is also an adjunct faculty member at Brookdale Community College.

Thomas Czernik is also a senior technical staff member with AT&T's Operations Technology Center, working on network operation support systems.

Jtest!

by ParaSoft

REVIEWED BY JENNIFER NESTOR

AUTHOR BIO

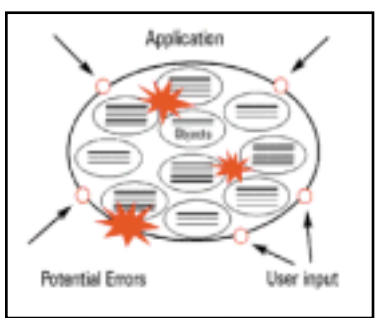
Jennifer Nestor, a staff consultant with Computer Sciences Corporation, has been programming in Java for over two years. She has a BA in mathematics and is currently pursuing an MBA. She can be reached at jnester@csc.com

jnester@csc.com

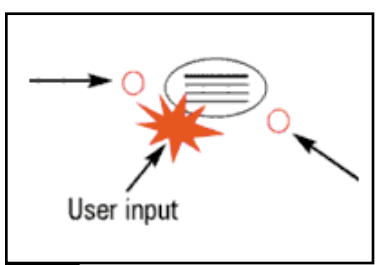
Jtest!: ParaSoft
Web: www.parasoft.com
Phone: 888 305-0041

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

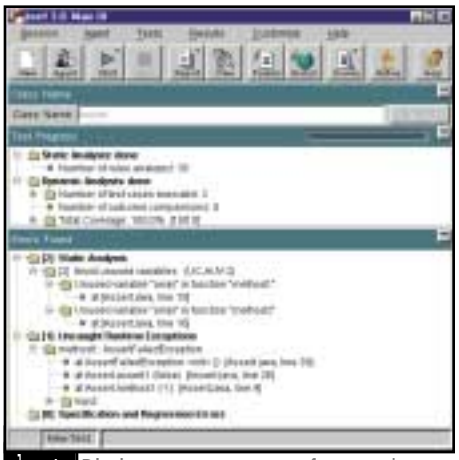
Pricing: $3,500.00



Jtest! Testing an application



Jtest! Testing a class



Jtest! Displays error messages from each type of testing in the bottom of the GUI

ParaSoft's Jtest 3.0 is a powerful automated tool for testing Java classes. Developers can unit-test their code for completeness and standards compliance and conduct regression tests to ensure that changes they've made to their code haven't introduced errors.

Environment

I installed Jtest on an IBM Thinkpad 600E running Windows NT 4.0 Workstation, Service Pack 5, with 128MB of RAM. According to Jtest's documentation, it'll run on Windows NT/95/98, with support for Solaris and Linux coming in the next few releases. Minimally, Jtest recommends you run with a Pentium 233 with 128MB of RAM. It also now supports Java Development Kit version 1.1.x and version 1.2. One of these JDKs must be in your path in order to perform all of the tests available.

Installation

Installation was simple; within minutes I was able to execute tests using the demo class installed with the application. The installation comes with a well-written tutorial that explains in easy-to-follow terms all of Jtest's features. That said, the application still relies on clumsy message boxes to explain what the application is doing and what the user may do next. Thankfully, these messages can be turned off.

Static Analysis

Performing code reviews is a part of any successful project. These reviews can catch many logic errors as well as ensure adherence to standards. Jtest performs these reviews by analyzing your code and comparing it to a set of rules. The user can customize these rules by clicking the Global button, opening the Static Analysis node, opening the Rules node and then opening the Built-in Rules node. From here you can enable or disable specific rules or entire rule categories by their severity levels.

White Box Testing

Anyone who has survived a long testing project knows that one of the most tedious processes is writing test cases. Jtest is the first testing application that generates unit test cases based on the internal structure of your classes. Using patented technology, Jtest examines bytecode, trying to break the class by attempting to pass unexpected variables to its methods.

To begin white box testing, open Jtest and browse to the class you'd like to test. To test multiple classes, go to the Project Testing UI and select the directory, zip or JAR file of classes. After this is completed, press the start button and wait for Jtest to conduct its tests.

Once the test is completed, Jtest will display the list of errors it generated. Right clicking on any of these errors will allow you to view the code on which the error was generated. This error can also be suppressed in the future if, for example, you know that a null would never be passed into these methods.

Black Box/Regression Testing

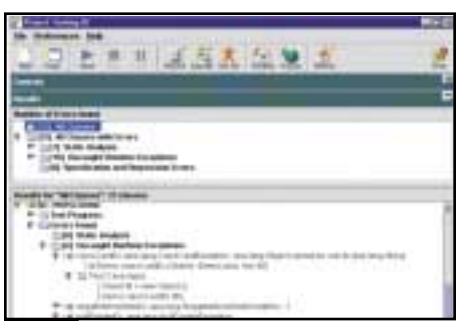
Black box testing checks to see whether the class is behaving according to its specification. During a normal testing phase, these test cases would have to be written by poring through written specifications and compiling combinations of possible inputs. Jtest provides these inputs for you and tests all possible combinations.

After running your first test on the class, you can view which inputs Jtest used and add any custom ones you think are necessary by selecting the View Test Cases button. After all the necessary test cases have been run, you can validate the output.

This validated output will serve as a baseline every time Jtest is run on the class and will perform regression testing by ensuring that any changes made to the class since the last test haven't changed these outcomes.

Summary/Recommendations

Jtest is unique in its ability to generate testing inputs and will save any development project a significant amount of time. Its automated execution of both white and black boxes sets it apart as a testing tool. Its testing should strengthen your project's code and allow developers to spend more time on coding and less time writing test cases. Jtest's only drawback is its price, which still remains slightly high at nearly $3,500 per seat.



Jtest! White-box testing

Broadcast live from San Francisco at JavaOne

INTERVIEW WITH... PETER COAD OF OBJECT INTERNATIONAL

JDJ: ***We're here to talk about Object International, TJ3, your classes and the book that that was recently released.***

Coad: Object International is focused on helping teams deliver frequent tangible working results around the globe.

JDJ: ***Tell us about some of the new features of Together/J3 that you'll be releasing soon.***

Coad: Together/J is something I've been working on for 10 years. We built earlier versions of this technology in C++, but not until Java have we been able to do all the things we wanted to do to give the developer a creative environment to work in. The field test for TJ3 began June 21, 1999.

JDJ: ***Can you give us an idea of some of the enhancements from previous versions, some of the new features?***

Coad: A hallmark of Together/J is simultaneous round-trip engineering, which is still there. New things in TJ3 include Gang of Four Pattern Support and Java modeling components.

JDJ: ***Your book's name is* Java Modeling in Color with UML: Enterprise Components and Process, *which you coauthored with Eric LeFebvre from Montreal, Canada, and Jeff De Luca from Melbourne, Australia. Is any of the material available online?***

Coad: A lot of the content is posted at www.oi.com (that's for Object International). I try to put as much content on the Web site as I can without my editor getting mad, so you can read freely and learn a lot about this approach.

JDJ: ***Why did you write the book? What did you see as an issue?***

Coad: I started modeling with a team in Singapore in September 1997. And this team was stuck. They had spent two years with a well-known author and Java design practitioner, but they had real problems.

The project manager realized that after two years they had large object models with just data and no methods and they hadn't delivered a single line of source code. My job was to come in and flip this team in a five-week period. I knew I wanted to teach on four categories, four archetypes, and that day on the table there were four colors of Post-it notes. I grabbed the yellow one and said, "This is a role." Took the green one and said "This is a person, place or thing." And so on. We went through four colors and started building a model.

The fascinating thing was after a week or two, we could see, on the wall, a model with a wave of color going from yellow to pink to green to blue. Even though we couldn't read the labels, we could tell things about the shape of the overall model.

During those weeks both newcomers to modeling and business experts came to me again and again and said, "Pete, we understand that you've never modeled in color before, but can you tell us, please, how it's possible that you could build effective models if you didn't have color." Through their eyes I saw the potential importance, in terms of what it would mean in model content, if we could really develop these ideas in practice. That's what we really worked to do.

JDJ: ***We're going to switch gears again and go back to TJ3. Could you give us some more info?***

Coad: We have the white board edition that developers use around the globe. There are no size or time limits in this product. We have other features aimed at corporate developers, such as nine UML 1.3 diagrams supported in the product. Documentation generation is especially important with our corporate clients like Home Depot.

You can actually launch other tools from within TJ3, e.g., you can launch an external editor, work in it and come back. TJ notices the update and auto-updates. In addition, you can invoke a compiler from TJ3 and if the compiler were to turn back error messages, TJ3 captures them; you can then navigate to each point in source code and models to edit your source, make the corrections and then fire off the compiler once again.

JDJ: ***What do you see as the most innovative, most interesting aspect of Together/J?***

Coad: Three things – patterns, components and simultaneous round-trip.

JDJ: ***What is in the future for Together/J?***

Coad: We're keenly interested in what makes sense with EJB. This summer we released Together Enterprise, which has Java and C++ support in it. It also has custom diagraming so if you're interested in real time and you're not happy with UML, you can actually adapt the diagrams and customize them. In addition, if there are more diagrams beyond UML (you could imagine someone actually coming up with more diagrams), you can actually define your own diagrams in Enterprise and have support.

Another thing we did in late summer with the Enterprise product is moving into the data modeling space so we have an ER diagramming tool that for JDBC you can forward-engineer into DDL and then reverse-engineer from the data dictionary.

JDJ: ***That's great. Now before we close up, do you have any final words of wisdom? Anything to wrap up the world-according-to-Peter-Coad kind of thing?***

Coad: What I would ask is that people consider going to oi.com, then go to the JM-book.HTM page and download the first chapter on *Java Modeling in Color*. The archetypes and color are explained there with examples. There are sequence diagrams that show the interactions. There is a template in there of 12 classes in four colors and if people really understand that one template, they'll have a template of how I've built models over the past 10-plus years. That same template also applies to the 61 examples in the book. That content is at our Web site and it's free. You can download it, get some color Post-it notes and put it to work.

So please try out this color modeling for yourself and see how much more effectively you might be able to work with the experts, and the developers.

InetSoft Technology Corp

www.inetsoftcorp.com

Design

sign.com/javlin

Developing with DCOM and Java

Part Two

Part One of this article appeared in the July issue of **JDJ** *(Vol. 4, issue 7)*

WRITTEN BY RICK HIGHTOWER

How can Java classes be used as scriptable components? DCOM, like CORBA, provides both static and dynamic invocation of objects. DCOM uses type library to provide metadata to do the dynamic invocation and introspection similar to CORBA's interface repository or Java's introspection mechanism.

IDispatch, a standard COM interface that supports Automation (late binding), is great for scripting languages such as Perl, Python, VBScript, JScript and so on. The Microsoft JVM implements IDispatch automatically for all Java objects. Microsoft refers to this feature as AutoIDispatch. Before AutoIDispatch, Java programmers had to write their own IDL files. Now all we have to do is create a Java class. Any public methods or member variables are automatically exposed via Automation. Anything that makes my job easier, I like.

Java programmers don't need to create interface definition files (IDL/ODL) to create COM objects. In addition, special tools like IDL compilers aren't necessary when doing Java COM development.

To implement a COM object in Java using AutoIDispatch, follow these steps:

1. Write a Java class and compile it. The class must have public members and methods.
2. Register the class using JavaReg.

DCOM Servers

Once you create a COM object, you need a server to serve the objects up to the world. A COM server is a container that holds COM objects. In essence, it's a class, a dynamic link library or an executable file that contains COM classes, which in the case of Java relates to Java classes. The COM server has the ability to turn classes into objects. The server implements the IClassFactory interface, which provides a standard way to request having a COM class instantiated into a COM object.

You won't actually need to create servers; they can be created via MIDL (Microsoft IDL compiler) or you can use the default one provided by the Microsoft Java SDK. In addition, COM clients don't need to deal with IClassFactory directly because the COM library handles this when you call CoCreateInstance or the extended version, CoCreateInstanceEx.

The COM client asks the COM library for a given class via a CLSID. The CLSID can be looked up in the registry. The COM library goes to the registry and looks up the CLSID, then instantiates a server, which must provide an IClassFactory interface so that the COM server can create the object on behalf of the COM library. For the COM client to ask the COM library for a class, the COM client has to find the CLSID in the registry. Therefore, the server must register a CLSID for every COM object it's able to create. Here are the three types of COM servers:

1. In-process servers execute in the same process (DLL, OCX, INPROC) context as the client. Local Java classes exported as COM objects run in this mode unless they're using a surrogate process that they need to run remotely.
2. Local servers run in a separate process (EXE) from their clients but still on the same machine. Basically, local servers run over LPC.
3. Remote servers execute in a separate process on a remote machine. The clients use RPC to talk to the remote server. However, a client doesn't have to know that the server is remote. This may be completely transparent to the client. Java classes exported as COM objects can be run in this mode only if they use a surrogate process, which can be the default surrogate process provided by the COM library. This process is covered in the code examples.

The COM library supplies three ways for COM clients to attach to a remote server and request a COM object:

1. The server name is associated with the ProgID of the object in the Windows Registry.
2. An explicit parameter is passed to CoCreateInstanceEx, CoGetInstanceFromFile, CoGetInstanceFromStorage or CoGetClassObject specifying the server name.
3. Monikers are used.

Newer clients that are DCOM savvy have the flexibility to specify the server they want to connect to, which is essential with some applications. The newer servers can specify the server name as a parameter to CoCreateInstanceEx.

from the Microsoft JVM,

Creating a Java COM Object

This code example assumes you have Microsoft's Java SDK 3.1 or higher. The Microsoft Java SDK is freely downloadable from its Web site. If you don't have it, get it from www.microsoft.com/java. Once you've downloaded it, follow the install instructions closely.

For the first code example we're going to create a simple COM server, then register it in the system Registry so other programs can find it.

In your favorite editor, enter the code to create the HelloCOM COM Program class HelloCOM:

```
{
  public int count = 0;

  public String getHello()
  {
    count ++;
    return "Hello from COM " + count;
  }
```

Save this in a file as HelloCOM.java. Now you need to compile it. Use the Microsoft compiler; from the command line, type:

```
C:\jsdk>jvc HelloCOM.java
```

Next, you need to register it with JavaReg by typing the command exactly as shown in the following command line. If successful, you should get a dialog box.

```
C:\jsdk>javareg /register /class:HelloCOM /progid:Acme.HelloCOM
```

JavaREG is a tool for registering Java classes as COM components in the system Registry. This tool also enables you to configure the COM classes you create to execute remotely.

You now need to copy the HelloCOM.class file to \java\lib in the Windows directory. You'll need to substitute drives and directories as needed:

```
C:\jsdk>copy HelloCOM.class c:\winnt\java\lib\HelloCOM.class
```

That's it! You just created your first COM object. Remember, the only difference between COM and DCOM, from the programmer's perspective, is a few Registry settings and the length of the wire. Let's do some poking around and see this firsthand.

Exploring COM with OLEVIEW

OLEVIEW, the OLE/COM object viewer, is a development, administration and testing tool. Let's work with it to get a feel for how HelloCOM is configured in the registry.

1. If OLEVIEW is on your path, type OLEVIEW at the DOS prompt. Otherwise, find it and execute it.
2. Select View, ExpertMode.
3. Select Object, CoCreateInstanceFlags, CLSCTX_INPROC_SERVER.
4. Expand the node labeled Grouped by Component.
5. Expand the Java Classes mode under Grouped by Component.
6. Find the class you created in the previous exercise (HelloCOM).
7. Click this node once.

The CLSCTX_INPROC_SERVER component won't run as a remote server. Local and remote servers run in separate processes. Inproc is associated with a DLL, so when the class is registered it's associated with the DLL that runs Java classes (namely, msjava.dll).

Notice that the CLSID and AppID are listed under the Registry tab on the right side. When we executed JavaREG earlier, we used the following for the class and ProgID arguments:

```
/class:HelloCOM /progid:Acme.HelloCOM
```

Notice how the names map to this ProgID.

Now let's look at the information we can glean from the Registry tab:

- The inproc server is listed as msjava.dll.
- The threading model is specified as Both.
- The Java class associated with this COM object is HelloCOM.
- The ProgID is Acme.HelloCOM.

Let's test this COM object. When you double-click the HelloCOM node, OLEVIEW will try to instantiate the COM class associated with the CLSID listed (an easy way to test if everything is set up). The following events will happen when you double-click this node:

- OLEVIEW takes the ProgID and finds the CLSID (remember, the CLSID is a 128-bit GUID that identifies our COM class) for this COM class.
- OLEVIEW calls CoCreateInstance, passing it the CLSID and the CLSCTX_INPROC_SERVER.
- The COM library loads msjava.dll and negotiates with its IFactory to get a COM object representing our Java class. See Figure 1.

The bottom line is that OLEVIEW is actually loading a live version of our class. This is a powerful tool for testing COM classes. Next, notice that the node has been expanded to show all the interfaces that this

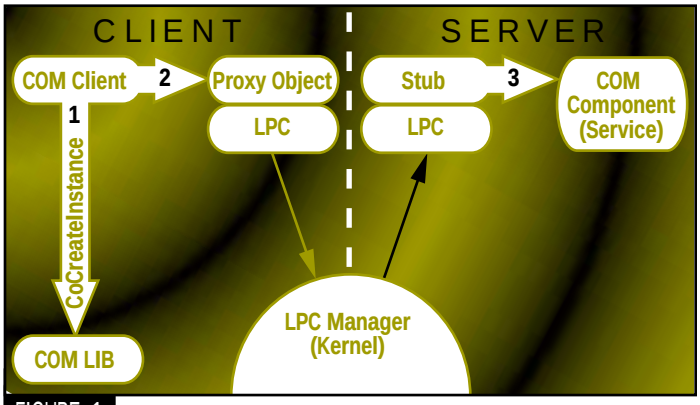


FIGURE 1

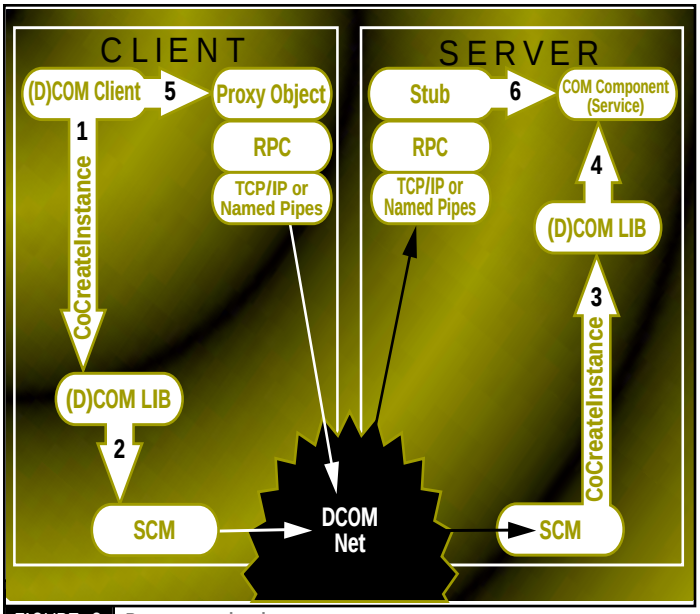


FIGURE 2 Remote activation

COM class supports. Notice also that it supports the following interfaces:

- ***IUnknown***: Interface negotiation and lifecycle management
- ***IDispatch***: Dynamic invocation for scripting languages
- ***IMarshal***: Custom marshaling

You should be familiar with these interfaces from the earlier discussion of the architecture in Part One of this series. From there we can ascertain that Microsoft has defined some kind of custom marshaler to marshal parameters to methods and return types between processes and machines.

Scripting the HelloCOM Object

We're going to use the COM server you just created with Visual Basic. This code example assumes you have some form of scripting language that works as an Automation controller. If you don't, don't worry about it. An example in the book mentioned at the end of this article shows you how to create an Automation controller in Java without a type library, i.e., using raw IDispatch.

I chose Visual Basic for this example because many people know how to use it, and it's widely available. If you can't find a scripting language that suits your fancy, don't sweat it. Just skip the scripting section of this example.

Any Automation-capable scripting language will do. You can use LotusScript (which comes with Notes and Lotus 1-2-3), VBScript, and Visual Basic for Applications (VBA) (which comes with Word and Excel). Two of my favorite Automation-capable scripting languages are Perl and Python. Both provide examples for doing Automation. They're freely available at www.perl.org and www.python.org, respectively.

From your favorite Automation controller scripting language, add a button to a form called Command1. Then add the following code.

```
Private Sub Command1_Click()
    Dim helloCOM As Object
    Dim count As Integer

    Set helloCOM = CreateObject("Acme.HelloCOM")
    MsgBox helloCOM.getHello
    count = helloCOM.count
End Sub
```

Notice that we use Visual Basic's CreateObject to instantiate the COM class into an object. Also notice that we pass CreateObject the ProgID of the COM class. You can probably guess what Visual Basic is doing underneath, but let's recap to make sure:

- CreateObject takes the ProgID and finds the CLSID for this COM class in the system Registry.
- CreateObject then calls CoCreateInstance, passing it the CLSID and the CLSCTX_INPROC_SERVER, which tells CoCreateInstance to create this object as an in-process COM server.
- The COM library loads msjava.dll and negotiates with IUnknown to get an IFactory interface.
- The COM library then uses the IFactory to get a COM object representing the Java class. (Actually, at this point IFactory returns an IDispatch that represents the interface to the COM object.) See Figure1.

Now let's consider what happens when the code example makes the following call:

```
MsgBox helloCOM.getHello
```

Here we're taking what helloCOM.getHello returns to use as a parameter to MsgBox. MsgBox is just a Visual Basic method that pops up a message box with a string that you give it. Underneath, Visual Basic is calling the IDispatch.Invoke method with the name of the method we defined in the Java class getHello. IDispatch.Invoke passes back a return type of variant. A variant, as you'll remember from Part One, can hold any type of value. Visual Basic then works with the variant to see what it contains (in this case, a string).

You may wonder why you're being prepped with all this background information. AutoIDispatch doesn't provide a type library. If it did, you could use JActiveX (a tool from the Microsoft Java SDK that wraps COM objects into Java classes) to create wrapper functions around the COM class you created (which we'll show you how to do in later lessons). Without JActiveX, creating wrapper classes can be a little tough.

Creating a Java DCOM Object

In this exercise, a duplicate of the first, we're going to create a remote COM object. We're going to use a different class name from the HelloCOM Java class – HelloDCOM – so you can compare the Registry settings of HelloDCOM with those of HelloCOM in the next exercise. As you did in the first example, using your favorite editor, enter the code below to create the HelloDCOM DCOM object.

```
class HelloDCOM
{
  public int count = 0;

  public String getHello()
  {
    count ++;
    return "Hello from COM" + count;
  }
}
```

Also, as before, you need to compile. Use the Microsoft compiler as follows:

```
C:\jsdk>jvc HelloCOM.java
```

Next, you need to register it with JavaReg:

```
C:\jsdk>javareg /register /class:HelloDCOM /progid:My.HelloDCOM
/surrogate
```

The only real difference between this step and what you did in the first exercise is the addition of the /surrogate parameter, which is essential for doing remote objects. The JavaReg /surrogate parameter allows the system-provided surrogate process to wrap the msjava.dll file in a process. This is needed – otherwise the DLL would have to run inproc, which can't be used with remote objects.

Again, as before, you need to copy this HelloDCOM.class file to the Windows directory:

```
C:\jsdk>copy HelloDCOM.class d:\winnt\java\lib\HelloDCOM.class
```

To test this setup, let's run the OLEVIEW program from the second exercise. Go to the HelloCOM class and try to instantiate. This will test to see if everything is working okay.

From OLEVIEW select View.ExpertMode and also set the Object.CoCreateInstance flags to CLSCTX_LOCAL_SERVER and CLSCTX_REMOTE_SERVER. It's essential that CLSCTX_INPROC_SERVER not be selected.

Expand the Java Classes node, then the Class HelloDCOM node. If the node opens, the COM class you created was instantiated to a COM object. This essentially tests that everything is running okay. Now compare all the parameters and settings in the Registry with the settings to HelloCOM. Go through the steps from the second exercise with HelloDCOM.

To actually use the COM object remotely, you're going to need to familiarize yourself with another tool: DCOM Configuration (or DCOMCNFG). DCOMCNFG is included with DCOM for Windows 95 and Windows NT with Service Pack 2 (SP2) or Service Pack 3. You can use it to set application properties, such as security and location.

On the computer running the client application, you must also specify the location of the server application that will be accessed or started. For the server application you'll specify the user account that has permission to access and start the application to the client computer.

Configuring the Server

Here are the steps needed to configure the server:

1. At the DOS prompt, type DCOMCNFG (or launch it any way you prefer; it's in the Windows \System32 directory).
2. Select Java Class: HelloDCOM in the Applications tab's Application list box.
3. Double-click Java Class: HelloDCOM or press the Properties button.
4. Another dialog box pops up called Java Class: HelloDCOM.

Ensure that the words DLLSurrogate appear next to the application type in the General tab. This is essential for remote operation of Java classes, as mentioned previously.

5. Go to the Security tab.
6. Select the Use Custom Access permission.
7. Press the Edit button to edit the access permissions.
8. Add the user name with the Add button. (Press Show Users in the Add Users and Groups dialog box.)
9. Set their permissions to Allow Access in the type of permission.
10. Select the Use Custom Launch permission.
11. Press the Edit button to edit the access permissions.
12. Add the user name with the Add button. (Press Show Users in the Add Users and Groups dialog box.)
13. Set their permissions to Allow Access in the type of permission.

Configuring the Client

Here are the steps needed to configure the client:

1. Run the JavaReg tool on the client machine. The following line is entered as a continuous line at the DOS prompt without a line break:

```
C:\jsdk>javareg /register /class:HelloDCOM /progid:My.HelloDCOM
/clsid:{CLSID}  /remote:servername
```

2. Set the CLSID to the 128-bit CLSID associated with this class. You can get this value by looking it up in OLEVIEW.
3. Set the server name to the name of your remote machine. Here's an example of what it might look like:

```
javareg /register /class:HelloDCOM /progid:My.HelloDCOM
/clsid:{064BEED0-62FC-11D2-A9AF-00A0C9564732} /remote:azdeals08
```

4. Next, you can use OLEVIEW on the client to ensure that you can connect to the remote server. This step should be familiar to you by now. It's the third time we've used OLEVIEW. Go through the steps from the second exercise with OLEVIEW and note the differences between HelloDCOM on the client and HelloDCOM on the server.

Demonstrating Access

From your favorite Automation controller scripting language, add a button to a form called Command1 and then add the code shown below, which demonstrates helloDCOM Class Access.

```
Private Sub Command1_Click()
    Dim helloDCOM As Object
    Dim count As Integer

    Set helloDCOM = CreateObject("My.HelloDCOM")
    MsgBox helloDCOM.getHello
    count = helloDCOM.count

    MsgBox helloDCOM.getHello
    count = helloDCOM.count
End Sub
```

We now pass the name of the COM class's ProgID to CreateObject. The Visual Basic runtime library initiates a similar process, as described in previous exercises. Notice that we use Visual Basic's CreateObject to instantiate the DCOM class into an object. Also notice that again we pass CreateObject the ProgID of the COM class. By this point you should be able to guess what Visual Basic might be doing underneath, but let's recap one more time:

- CreateObject takes the ProgID and finds the CLSID for this COM class in the system Registry. The Visual Basic runtime notices that the object being requested is a remote object.
- CreateObject then calls CoCreateInstance, passing it the CLSID, which tells CoCreateInstance to create this object as a remote process COM server.
- The local machine's SCM contacts the remote machine's SCM.
- The remote machine's SCM uses the COM library to load msjava.dll in a surrogate process and then negotiates with IUnknown to get an Ifactory interface.
- The COM library then uses the IFactory to get a COM object representing the Java class. (Actually, at this point, IFactory returns an IDispatch that represents the interface to the COM object.)
- The IDispatch reference gets marshaled back to the local machine so that the VB application (the COM client) can begin making calls. See Figure 2.

Conclusion

You know how to create a local COM server and a remote COM server. You know how to test both a local and a remote COM server with OLEVIEW. You know how to configure a COM server to be a remote server with JavaReg and DCOMCNFG. If you read this article and the previous one, you should have a feel for the difference in the architectures of DCOM, CORBA and RMI.

Reference

In the book *Java Distributed Objects* by Bill McCarty and Luke Cassady-Dorion, the subject of Java DCOM programming is covered in more detail with more examples. The book also covers RMI as well as CORBA (with an emphasis on CORBA). I wrote chapter 20 on DCOM, which covers Java and DCOM in greater detail, how to create callbacks in DCOM and how to use JActiveX to create Java wrappers around existing COM components. Also, I have an example that uses late bound calls using IDispatch.

Author Bio

Rick Hightower, a senior software engineer at LookSmart, a category-based Web directory, has been writing software for a decade, from embedded systems to factory automation solutions. Rick recently worked at Intel's Enterprise Architecture Lab, where he researched emerging middleware and component technologies.

rick_m_hightower@hotmail.com

 —continued from page 7

- Support for Windows NT only (not cross-platform; UNIX still preferred for high-end)
- Proprietary; lock-in to Microsoft as vendor
- No support for standards like CORBA

J2EE Levels the Playing Field

As vendors announce J2EE capabilities in their servers, we'll see an interesting phenomenon: every server will initially appear the same. Clearly, vendors will need to seek ways to differentiate their products. We predict that this differentiation will center on three layers (see Figure 1).

In the bottom layer vendors will try to differentiate their base J2EE capabilities by touting higher performance, better reliability or management tools. Differentiation will be minimal as many vendors make the same claims, which will be hard for customers to evaluate. As a result, we see the J2EE layer soon becomes a commodity.

In the value-add layer above the J2EE standard, vendors can differentiate by adding technology value such as integrated development tools, connectors or data architectures for dealing with legacy and nonrelational data. Analysts, including Gartner and Meta, recognize integrated development tools as a key differentiating feature for application servers. Development projects are typically carried out by a mix of expert 3GL Java developers (few and hard to come by) and 4GL-style developers (the majority of corporate developers) migrating from tools such as PowerBuilder and Visual Basic. Expert Java developers frequently have a preferred code editor and avoid productivity tools, but 4GL developers look for tightly integrated tools that can provide huge productivity gains as well as help them to develop according to "best practices" that aren't otherwise obvious. The bottom line: integrated development and deployment tools will speed time-to-market, which is key to customers.

Content management is another value-added area where vendors differentiate their servers. Vendors realize that, in a large percentage of Web applications, application content drives users to do transactions. For example, in e-commerce sites product catalogs or investor research drives the purchases and trades. Content management features include the ability to author and create content easily, store it in a database, publish database fields to URLs on the Web, version control, approval workflows and full-text indexing.

The application layer is the third place where differentiation will occur. Smart vendors realize that the key reason customers buy application servers is to solve business problems. A few categories of business problems are common, and as such it's possible to save customers large amounts of time by offering an application framework that provides prebuilt components that can be easily customized and extended to meet customer-specific requirements. This provides a faster time-to-market than any feature in an application server, and is valuable to customers.

Conclusions

Buyers and users of application servers will benefit from the J2EE standard. It will provide a clear standard for developing three-tier distributed applications as well as an industry-standard skill set that developers will learn. We highly recommend that developers learn how to make use of the J2EE standard as it will rapidly become the most highly valued skill set in organizations wanting to use the Web in their business.

J2EE "best practices" aren't yet widely known or documented. We strongly recommend that organizations use experienced consultants on the project teams of their first J2EE projects to learn these best practices and to avoid the pitfalls of poor development.

J2EE will "commoditize" the base application server functionality, causing smart vendors to differentiate their products in either the technology value-add and/or application layer. Buyers should evaluate vendor strategies according to their own set of priorities.

For a resource area on J2EE and Java development, visit devcenter.sil-verstream.com.

CORBA Project Survival

. . . Or how to be sure your first CORBA project isn't your last

WRITTEN BY
STEVE TOCKEY

Congratulations! You've just been designated the project manager of your first CORBA project! "Help!" you say? Even though you may not have any CORBA experience at all, you needn't panic. This article describes how you can grab this bull by the horns and guide your project to a successful completion. Not only can you survive your first CORBA project, but you can do everything in your power to make it a success. Luck isn't the missing ingredient – knowledge is.

This article describes how I approached my first CORBA project – and my second. It also describes how I'd approach my next CORBA project, the next one, and so on. This article is based loosely on Steve McConnell's award-winning books *Software Project Survival Guide* and *Rapid Development* (see References 1 and 2). They provide some additional discussion of the approach described here.

The Project Charter

I start a project by clarifying its "charter." At a minimum, the project's charter defines its goals, objectives, completion criteria, resources and constraints. It really doesn't matter what form the project charter takes, such as a formal document, a memo or even an e-mail message. The critical point is that the project manager and the project team have an explicit agreement with the project stakeholders over what constitutes project success.

The Risk Assessment

The second step in the project is the risk assessment. In this step we try to identify the major stumbling blocks that could prevent the project team from achieving success.

Risk can be defined as "the possibility of an unwanted consequence of an event or decision." There are two important components of this definition. First, unlike the financial markets, software project risk is not a good thing – there's no upside to it. Second, software project risk is a probability. A "problem," however, is a certainty; it's a previously known or unknown risk that has already materialized.

The steps in a risk assessment are as follows:

1. *Risk identification*
2. *Risk analysis*
3. *Risk prioritization*

Risk Identification

The most abstract, highest-level risks are the same on every project:

- *Cost overrun*
- *Schedule overrun*
- *Quality underrun*
- *Functionality underrun*

Except for outsourced projects, it ends up being impractical to deal with software project risk at this high a level. It's necessary to drill down one level and ask, "Why might the project experience a cost overrun?" "What are the reasons why we might experience a schedule overrun?" etc.

One effective way to identify risks is to start with a list of generic software project risks. Several such lists are available, including one at www.construx.com (click on Software Resources, then Software Development Checklists, then Complete List of Schedule Risks). Let the team identifying risks select those candidates that warrant further investigation from the generic list. Don't limit risk candidates to what's on the prepared lists. Be willing to consider any factor about the project that could cause difficulty. Generally speaking, project risks relate to the following project characteristics:

- The product itself (complexity, size, the "–ilities," such as reliability, portability, etc.)
- The processes, tools, techniques and technology being used (e.g., CORBA)
- The project team itself (size, abilities, desires, team cohesion)
- Parent organization(s)
- Supplier(s) and subcontractor(s)
- Customer(s)

Since this is probably your first CORBA project, it's likely to be everyone else's as well. Some common CORBA-specific technology risks for first-time projects are:

- Lack of project team familiarity with CORBA and/or the specific ORB product
- Immaturity or instability of the ORB products
- Not meeting performance requirements (e.g., insufficient network bandwidth)

Risk Analysis

In risk analysis the probability and severity of each candidate risk is estimated. The probability estimate is a judgment of how likely we think it is that the risk will turn into a problem. Probabilities can be estimated using any number of scales, from a simple "low-medium-high" scale up to a numeric probability. For example, "If nothing is done to prevent it, there is a 40% probability that our lack of familiarity with CORBA will become a problem for the project."

The severity estimate is a judgment of how much difficulty the project would incur if the risk were to turn into a problem. Estimating the severity for each candidate risk is similar. You can use the simple low-medium-high scale again or you can estimate the dollar-cost for what it would take to deal with the problem. For example, "If our lack of familiarity with CORBA did turn into a problem, we believe it would cost us about $50,000 to recover from it." The recovery costs could include bringing in an outside consultant, delayed deliveries because of missed milestones, and so on.

Risk Prioritization

Practically speaking, the project team can probably handle active control of between a half-dozen and a dozen risks. It's not uncommon for the previous risk identification step to identify two or three dozen candidates. The risk prioritization step is used to figure out which of the candidate risks will be actively controlled and which will be deliberately ignored. The "risk exposure" for each candidate risk is calculated and the risks are then rank-ordered by exposure. Risks that carry a high exposure will be actively controlled, while those with low exposure can usually be safely ignored.

When probability has been estimated in terms of percentage, and the severity has been estimated in terms of dollar cost, the exposure for each risk is simply the probability times the dollar cost. For example, the lack-of-CORBA-familiarity risk, with a probability of 40% and a severity of $50,000, leads to a risk exposure of $20,000. If the immaturity/instability of the ORB product had a probability of 10% and a severity of $100,000, the risk exposure on this risk is $10,000. We should be more concerned about controlling the CORBA familiarity risk than the immaturity/instability risk.

When the probability and/or severity are estimated in terms of a simple low-medium-high scale, the prioritization is less straightforward. Clearly, a high probability/high severity risk carries a higher exposure than a low probability/low severity risk, but judgment will be required to prioritize a low probability/high severity risk relative to a high probability/low severity risk. The important thing is not to have precisely calculated risk exposures. Rather, it's to understand which risks are more important and warrant more effort and attention.

Once the candidate risks have been rank-ordered by exposure, the list is examined from the top down (from the highest exposure to the lowest) to decide which risks should be actively controlled.

Asset Assessments

I've also found it useful to consider software project "assets," which are like inverse risks. They are characteristic of the project that the team should be careful to take advantage of. For example, a very tight set of customer requirements that must be precisely satisfied could be considered a project risk. On the other hand, maybe the customers are much more flexible and willing to give the project team significant control over the detailed requirements. The flexibility offered to the team is an asset that the project team should be aware of and take advantage of.

	PI1	PI2	PI3	PI4	PI5	PI6	PI7	...
Ch1	X						X	
Ch2		X				X		
...								X
Risk1				X				
Risk2		X						
...								X
Asset1			X		X			
Asset2			X				X	
...								X

TABLE 1 Sample traceability matrix

Asset assessment can be performed alongside risk assessment, since it follows the same steps. The generic risks can also be considered candidate assets. For instance, a large, geographically dispersed project team would be a risk whereas a small, colocated project team would be an asset. The probability that the asset will turn out to be useful can be estimated as the risk probabilities are being estimated. Similarly, the asset benefit, that is, the value to the project team if the asset were realized, can be estimated alongside the risk severities. Finally, the assets are rank-ordered in terms of potential benefit exposure (probability times consequence or some similar formula) and a decision is made about which assets will be managed actively.

The Project Plan

The third major step is to develop the project plan. The most important thing to remember while doing the planning is that every activity in the plan, and the level of formality that those activities will be performed at, should be selected carefully to:

- *Help the project satisfy its charter.*
- *Help the project control a high-exposure risk.*
- *Help the project maximize the benefit of some asset.*

In short, every part of the charter, as well as every high-priority risk and asset, must be addressed adequately by one or more planned actions or activities, and every planned action or activity must be there to help satisfy the charter, control risks or maximize assets. Some project teams have taken the extra step of developing a "traceability matrix" to verify the consistency between the project plan and the charter and risk/asset assessment.

The sample traceability matrix in Table 1 shows, for example, that Charter component 2 (whether it's a goal, resource, constraint or asset) is being addressed jointly by Plan components 2 and 6. Similarly, Plan component 3 is in place to address Assets 1 and 2. In this format any empty row means that a charter component, risk or asset isn't being addressed by the current plan. An empty column means that a plan component isn't addressing any charter component, risk or asset.

To the extent that a charter component, risk or asset isn't being addressed by planned activities, the project has a much lower probability of successful completion. Your chances of surviving this project have been reduced. Similarly, to the extent that the planned activities don't address the project's charter, risks or assets, the project team will be wasting its time. Time spent on these activities won't carry any benefit for the project team and may prevent it from doing things that would benefit the project. Given the tight constraints that the typical software project operates under, we should be very sensitive to wasted effort.

To illustrate the variation in process formality based on risk, consider that a large, geographically dispersed project team would probably benefit from a formal approach to configuration and change management, while a small, colocated project team may find an informal approach entirely sufficient.

I start the detailed project planning with the project charter and the list of high-priority risks and assets, together with a project plan template that identifies the basic planning components. Table 2 is a high-level view of this template.

Planning to Control Risk

There are four basic strategies for controlling any given risk:

- *Avoidance*: Act to make the probability go to zero (i.e., make it impossible)
- *Mitigation*: Act to reduce the probability and/or the severity

Force 5

www.force5.com

- *Transference*: Push the risk onto someone else
- *Acceptance*: Develop a contingency plan for handling the problem should it occur

(More information on risk management can be found in References 3 and 4.)

As there are probably some fairly obvious CORBA-related risks on your first project, we can think about specific actions to control them.

- *Lack of development team familiarity with CORBA and/or a specific product:*
  1. General CORBA training (see the OMG Web site at www.omg.org) – free training is provided for OMG member organizations five times per year at OMG Technical Committee meetings.
  2. OMG's "Ask the Experts" forum
  3. Specific ORB and Services training from the vendor
  4. Expert consulting help from the vendor or a knowledgeable professional consultant

- *Immaturity/instability of ORB and/or Services products*
  1. Early prototypes to determine ORB/Services stability
  2. Contact other users of the same vendor product

- *Not meeting performance requirements (e.g., insufficient network bandwidth)*
  1. Early prototypes of performance-critical code sections
  2. Expert consulting with the ORB vendor regarding performance tuning

Tracking and Oversight

Planning the project is one thing. Guiding the project to successful completion requires more. It takes careful monitoring of the project's ongoing performance, comparison of the project's actual performance with the expected status as derived from the plan, and the careful application of corrective actions when significant differences between the actual performance and the plan are discovered. The corrective action could include an entire replan of the project if one or more of the basic planning assumptions turns out to be incorrect.

Ongoing Risk Management

The project manager needs to understand that risks (and assets) have a lifetime. The earliest lifetime phase is where the risk is known but it's too early for it to become a problem. As an example, consider the risk of a lack of ORB knowledge. Once the use of CORBA has been decided, we can identify that risk. But until the ORB product is selected, it's impossible for the risk to materialize. The middle phase is where the risk has a nonzero probability of materializing. The final phase is where the risk has actually materialized as a problem or its probability has gone back to zero. At some point either the project team's lack of ORB knowledge has materialized as a problem or the team has obtained adequate knowledge so it's no longer a problem.

The point is that we can't view the project's risks as being static. It's important for the project manager, and the team as a whole, to keep in mind that some active risks will disappear at certain points in the project and other new, previously unrecognized risks may come into play. Since major changes in the risk landscape are most often associated with major project phase changes (e.g., moving from requirements to design or from design to code), it's recommended that the project team reassess its risks (and assets) at every major project milestone.

As existing risks (and assets) are retired and new ones are discovered, it's vitally important that the project plan be modified to account for those changes to maintain the traceability of the project's charter, risks and assets and the current project plan. It's also possible that the project's charter may undergo change over the course of the project. Again, it's vitally important to the survival of the project that any significant changes to the charter be matched by corresponding changes to the project plan.

A Summary of the Approach

This approach really suggests the adoption of a "project development lifecycle" that's similar to the more familiar software development lifecycle as shown in Table 3.

The project charter establishes the ground rules for the project. The risk and asset assessment is used to discover the important characteristics of the product and the organization(s) working on it. This is the basis for developing the project plan. If the same charter were given to different organizations, different project plans would likely result owing to important risk and asset differences between the organizations. Execution of the plan results in producing a system that satisfies the given charter in light of the known risks and assets. And just like ongoing software product maintenance, the project plans will likely need maintenance due to recognized changes in the project's circumstances.

Results of Using This Approach on My First CORBA Project

In January 1997 I was assigned to my first CORBA project as its manager. The project involved developing a set of CORBA performance benchmark applications to investigate the use of ORBs in high-performance, safety-critical avi-

Technical Activities
Documented requirements/requirements development
Design/UML
Coding standards
Etc.
Quality Activities
Peer reviews
Testing
Requirements trace
Management Activities
Schedule tracking and management
Cost tracking and management
Functionality tracking and management
Quality tracking and management (e.g., defect tracking)
People management (staffing, motivation/morale, environment, teamwork)
Configuration Management
Change Management
Status reporting
Deployment Activities
Product Packaging
System Installation/system conversion
User Training
User Support
Software lifecycle (waterfall, iterative, spiral,…)
Resource allocations (schedule, pert chart, etc.)

TABLE 2 Project plan template

The Theory Center

www.theorycenter.com

SOFTWARE LIFECYCLE	PROJECT LIFECYCLE
Requirements	Project charter
Analysis	Risk and asset assessment
Design and coding	Project planning
Operation of the software, producing the desired results	Execution of the plans, resulting in software product(s) that satisfy the project Charter in light of the risks and assets.
Maintenance due to changing customer needs	Replanning due to changing project circumstances

TABLE 3 Software development lifecycle

onics applications. While I had significant personal experience with CORBA, having been involved with the OMG since 1993, the other project members had none to speak of. As a team, we documented the project's charter, assessed our risks and assets, and built a plan consistent with the particulars of this project. Partway through the project it was decided (outside the project team) that one of the benchmark applications should be subcontracted out to a neighborhood software vendor who also had no significant CORBA experience. Again, we chartered the vendor project and walked the vendor through the risk and asset assessment. We then helped them develop an appropriate project plan (which, by the way, was significantly different than our plan for that part of the project). In the end, the benchmark applications were delivered on time and, in fact, under budget.

Beyond CORBA Project Survival

What's the difference between the way I managed my first CORBA project and how I'd manage any software project? Very little. The only substantial difference is in the candidate risks and assets to be considered. In the CORBA project case we considered risks about the use of a technology we might not have used before. Similar risks would be encountered on your first Java project, your first DCOM project or your first anything project.

This approach to project survival was initially developed while I was obtaining my MS in software engineering at Seattle University. It's been used successfully, starting with my senior capstone project (a six-person/year project demonstrating agent-oriented applications) up to and including the redevelopment of the corporate employee records system at a major manufacturing corporation.

Conclusions

At this level of actions and activities in a project plan, the message should come across that there's no one-size-fits-all software project methodology. Nor could there ever be, despite what some methodology vendors would like us to believe. Only when the specific actions and activities in the project plans are tuned to the peculiarities of the project at hand (i.e., are directly related to the charter, the risks and the assets) will the project be capable of performing in an efficient and effective manner. Only then will you stand the best chance of surviving your first CORBA project – or, in fact, any project.

References

1. McConnell, S. (1996). *Rapid Development*. Microsoft Press.
2. —(1998). *Software Project Survival Guide*. Microsoft Press.
3. Boehm, B.W. (1989). *Tutorial: Software Risk Management*. IEEE Computer Society Press.
4. Fairly, R. (1994). "Risk Management for Software Projects," IEEE Software, May.

Author Bio

Steve Tockey is vice president of consulting at Construx Software and has been employed in the software industry since 1977. Steve has an MS in software engineering from Seattle University and a BA in computer science from the University of California, Berkeley.

stevet@construx.com

Protoview

www.protoview.com

Oracle JServer Scalability and Performance

This new breed of JVM provides what's needed for large-scale enterprise applications

WRITTEN BY
JEREMY LIZT

As Java has evolved from the language of applets and JavaBeans to that of servlets, Enterprise JavaBeans and database stored procedures, a need has developed for a scalable Java platform. No longer are Java applications run only for a single user. Companies are now building enterprise-scale production systems using server-side Java technology, and these systems need to scale to serve tens – often tens of thousands – of concurrent users. Although many efforts have been made to enhance the JDK, Sun Microsystems' reference JVM implementation, Oracle pursued a different strategy as it set out in 1996 to develop an enterprise-scale, server-side Java Virtual Machine from the ground up. This effort manifested itself in the recent release of Oracle JServer, a 100% Java-compatible server environment supportive of Enterprise JavaBeans, CORBA servers and database stored proce-

Challenges of Java Scalability

A platform or application is generally said to be scalable if it provides consistent performance whether serving a handful or an abundance of concurrent users. Ideally, the user of a system will experience delays that are independent of the number of users on the system. If a bank teller must wait 100 msec to perform a transaction during off-peak periods, he or she should wait just about 100 msec during peak periods. Naturally, systems don't scale ad infinitum, but begin to show degraded performance above some critical load. An application may be said to scale to *n* concurrent users when response time begins to bloat with the *n + 1th* user (see Figure 1). Once usage exceeds this threshold, it's preferable for system performance to degrade gracefully and linearly, rather than in an erratic, unpredictable manner.

There are a number of central challenges in building a scalable Java platform. First, the server must provide an effective mechanism to handle multiple concurrent clients. The architecture of most JVMs forces application developers to implement their own scalability by writing multithreaded Java code. A JVM that professes to be a scalable Java server may claim to provide good support for Java language threads. Two fundamental problems impair this approach: the application developer is required to build in his or her own scalability (a difficult, error-prone task), and garbage collection on heavily threaded applications can be extremely inefficient. In a simple model in which each individual user maps to a Java language-level thread, a single garbage collector deals with all garbage from all users.

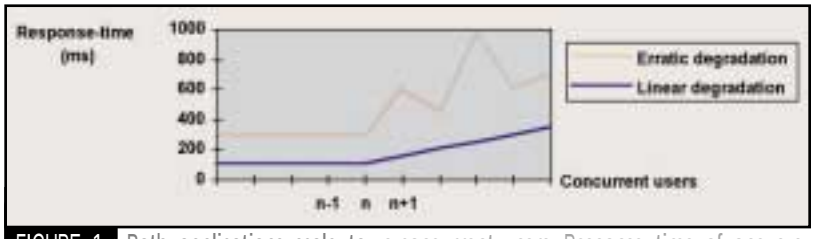


FIGURE 1 Both applications scale to *n* concurrent users. Response time of one system degrades erratically; the other degrades in a graceful, linear manner.

The technique by which a JVM performs storage management, as well as the JVM architecture, determines the efficiency of garbage collection. The second challenge facing the Java server provider is optimizing storage management. One of Java's primary benefits to developers is that they need not expressly allocate and free memory. Automated memory management is the responsibility of the VM, and a poor garbage collection algorithm will diminish the scalability of a Java server.

The third challenge in achieving platform scalability is minimizing the memory footprint consumed by each user of the system. The server will naturally be limited by the physical resources of the configuration, so the JVM must make the most efficient use of its available memory to maximize the number of clients it can serve. Many vendor implementations attempt to achieve scalability by spawning multiple instances of a JDK or similar JVM. This approach incurs substantial redundancy, however, contributing to a burdensome client footprint and restricting scalability.

Finally, a Java server should execute code swiftly as a means to bolster scalability. An inefficient bytecode execution will uneconomically consume precious CPU, resulting in sluggish performance and degraded scalability potential. Common approaches to accelerating execution typically involve some sort of just-in-time (JIT) or dynamic native compilation of Java bytecodes.

Oracle JServer Architecture

The challenges of concurrent garbage collection, memory management and session footprint are directly addressed by the Oracle JServer architecture.

Oracle JServer is tightly integrated with the Oracle8*i* database, and both share the notion of a client session. When a client initiates a session either through the SQL database or directly with JServer, it receives a single session within Oracle8*i* that comprises both a SQL database session and a JServer ses-

Editor's Note: *This article contains benchmark data developed by Oracle. Because we feel this is timely information, we've decided to run the article ahead of confirmation by the* **JDJ** *laboratories. We'll confirm these figures in first quarter 2000 and will publish an update in our Enterprise Application Server issue, scheduled for April 2000.*

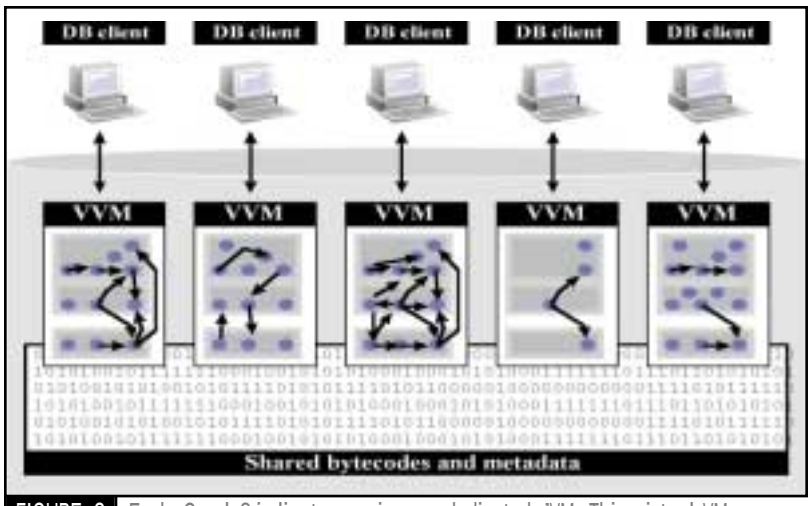


FIGURE 2 Each Oracle8*i* client perceives a dedicated JVM. This virtual VM (or VVM) actually shares much of its implementation and read-only data with other JServer sessions.

sion. Each JServer session maintains its own Java state, which is not shared between sessions. Every client to JServer thereby perceives a dedicated JVM. Although many JVM implementation resources are shared between sessions (read-only bytecode information, for example), the experience of a JServer client is that of a devoted JVM. The client session, appearing to the client as a dedicated VM, can thus be regarded as a *virtual* virtual machine, or VVM (a model similar to that of a UNIX operation system) (see Figure 2).

Oracle JServer was designed as a server-side Java platform, and it is specifically architected to address challenges of Java scalability. A fundamental advantage of the architecture is that it relieves the Java application developer of the need to write multithreaded server code. When developing for the JServer platform, a developer is encouraged to write the application for a single user. Scalability is achieved once the application is loaded into JServer. The virtual machine uses the multithreaded server facilities of the RDBMS to concurrently schedule Java execution, enabling thousands of clients to simultaneously access the application through independent JServer sessions. In addition to freeing the application developer to focus on his or her application code, the JServer architecture provides the related benefit of escaping the garbage collection bottlenecks that can hamper performance in multithreaded Java applications. JServer sessions, or VVMs, maintain independent memory stores and can be garbage-collected independently.

JServer thereby enjoys a performance advantage because the burden and complexity of memory management doesn't increase as the number of users increases. The memory manager always deals with the allocation and collection of objects within a single session. The simplicity of this scenario welcomes the application of sophisticated allocation and collection schemes attuned to the types and lifetimes of objects. For example, new objects are allocated in fast and cheap call memory, designed for quick allocation and access. Objects held in Java static variables are migrated to more precious and expensive session memory. Different garbage collection algorithms are applied in the various memory areas, resulting in high efficiency and low overhead.

Oracle JServer shares common read-only code and other appropriate data between its concurrent sessions. Only state variables unique to a session need to be stored privately for each VVM. Through this model JServer minimizes the memory footprint required to service its clients. Since its architecture scales comfortably to the capacity of its host machine, it never needs to spawn new JVM instances to serve additional users. Its shared memory model can therefore be leveraged over its full user population. Memory footprints for simple programs like hello, world require as little as 35K per concurrent user on JServer. This measurement compares favorably to analogous footprints exceeding 1M on servers that are forced to spawn additional JVMs.

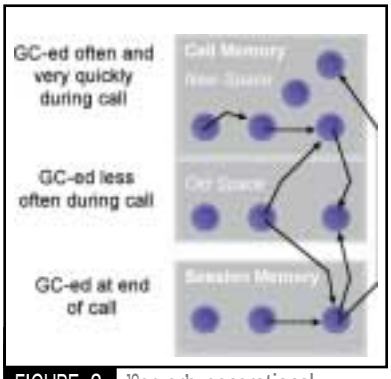


FIGURE 3 JServer's generational garbage collector segregates Java objects into separate memory areas to improve memory management efficiencies. A newly allocated object is placed into New Space, which is "swept" frequently. An object that survives *n* sweeps is migrated to Old Space, swept less often. Objects reachable through static variables at the end of a method call are placed into Session Memory, whose contents are persisted between method calls.

Oracle JServer addresses garbage collection, memory management and footprint challenges by means of its architecture. It confronts the execution performance challenge with its JServer Accelerator native compilation technology. In contrast to the JIT approach employed by many JVM implementations, the JServer Accelerator uses a WAT (way ahead of time) technique. As server code tends to be long-lived on its host machine, it's generally worthwhile to compile it into well-optimized native code. A WAT will invest more time in compilation than a JIT to produce more comprehensive optimizations.

ScaleServer Benchmark

The ScaleServer benchmark was designed to provide a fundamental measurement of Java server scalability. It specifically targets a server's ability to run multiple concurrent users, perform basic computations, and allocate and free Java memory structures. These tasks encompass the four central challenges of Java scalability discussed previously.

Description

The benchmark employs a CORBA infrastructure in which all clients to the server are CORBA clients that connect to a CORBA server running either as a standalone HotSpot Java VM or in the Oracle JServer. Each client creates its own CORBA object on the server side. In the case of Oracle JServer, each client's server object operates in an independent session. Once connected to its server object, each client executes a loop that runs a "work" method on the CORBA server object and sleeps for a period of time. (With the HotSpot Java VM, a set of worker threads is formed to execute the workload on the server side.) The method tests processing power by computing Fibonacci numbers. To test memory management and garbage collection efficiencies, the work

JDJ Consulting Services

www.javadevelopersjournal.com

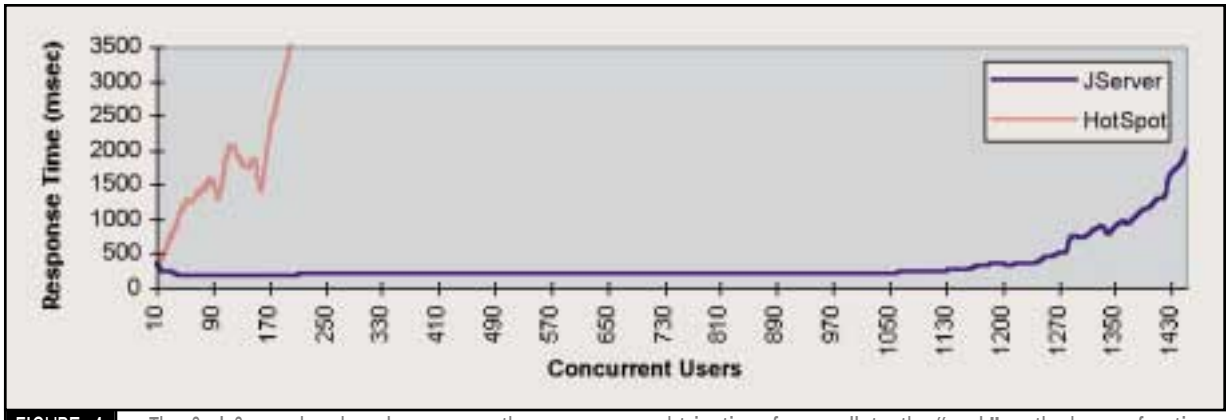


FIGURE 4 The ScaleServer benchmark measures the average round-trip time for a call to the "work" method as a function of the number of concurrent clients. The graph shows the results of the test run on a Sun Enterprise 450 class 4 CPU machine. Oracle JServer produced consistent response times for over a thousand simultaneous users while the HotSpot engine gave bloated and erratic results. Even after surpassing the capacity of its host machine, JServer gave graceful degradation of performance.

method instantiates, holds and drops Java objects. The workload represents a typical conversational load with multiple concurrent, stateful sessions on the server. Clients are added until the load becomes unmanageable to the server. The benchmark clients measure the round-trip time between the issuance of a request and the receipt of a response from the server. (Note that no SQL database access is done in the benchmark.) Although swift SQL access is an inherent benefit of JServer's integration with the Oracle8*i* database, the ScaleServer benchmark is intended to produce a pure measure of a Java server's execution scalability.

Parameters

The Java servers were run on a Sun Workgroup Enterprise 450 machine running four 250MHz Ultrasparc II processors with a 1.6GB/sec UPA interconnect and 1GB/sec PCI I/O subsystem. The E-450 machine was configured with Solaris 2.5.1, 4GB main memory and 100GB of fast hot-swap Ultra SCSI internal storage. Client threads were run on separate machines. Every request by a client caused the server object to compute the twenty-third Fibonacci number, allocate and drop 200 binary object trees of depth 5, and recycle 10 object trees from an additional persistent store of 200 such trees. After instantiating the server object, a client repeatedly issued calls to the server object, sleeping for 3 minutes after receiving each response. The

—continued on page 76

Sprint

gofish@sprint.net

Calling MS Excel via the Java Native Interface

A hybrid application, but the technique works with any COM object

WRITTEN BY
ALLAN K. GREEN

There are many situations in which it's useful or necessary to invoke native functions from Java. One of the more challenging is to invoke Microsoft COM functionality using Java native methods. Developers using Microsoft's J++ platform can sidestep the problem by creating a Java interface directly from the COM (OLB) files, but the resulting Java application may include Microsoft-specific language extensions. However, a platform-independent solution (at least in the Java sense) is available using the Java Native Interface. JNI provides a standard interface by which Java classes interface with native code and vice versa. In this article we'll develop a hybrid application using a Java GUI to invoke MS Excel functionality wrapped in a C++ load library (dll). While the code samples are specific to Excel (and, admittedly, of little practical use), the technique is extensible to any COM object.

Using the Java Native Interface

JNI is a specification for native code interfaces developed by Sun Microsystems and distributed as part of the JDK. A detailed description of the interface can be found in Beth Stearns's article at http://java.sun.com/docs/books/tutorial/native1.1/index.html, but, in summary, using the JNI, you can:

- *Call applications and functions written in C, C++ or Assembler from Java.* In this example we'll use native code to create an instance of an Excel application, open a workbook and worksheet, insert some data and chart it.
- *Call Java functions from native code.* Our example will request chart data values from a Java Option Pane.

While additional JNI features are covered in the Sun tutorial, they won't be used in this example.

Using the JNI requires the following steps:

FIGURE 1 Simple Java GUI interface that invokes the native methods

1. Create the Java class or classes that will act as the native code interface. Our example creates the AXExcel class, which defines the Java methods that will be implemented in native code. The AXExcel class is then compiled to the AXExcel.class file.
2. With AXExcel.class as input, run the javah utility (included in the JDK) with the –jni option to create a C-style header file – in this case, AXExcel.h.
3. Create a dynamic library (dll) implementing the functions in AXExcel.h. The native code in this example will be implemented in AXExcel.cpp. This is by far the most time-consuming step.
4. Build and link the dynamic library as AXExcel.dll. This library must be accessible to the Java AXExcel class.
5. Create a simple Java GUI interface to invoke the native methods (see Figure 1).
6. Test and debug the application.

This example was developed using Symantec Visual Café Professional V3.0 and MS Visual C++ 6.0 running on a WinNT 4.0 SP5 system. Win95 platform developers may need to install DCOM to support the COM features.

Creating the Java Native Class

Listing 1 contains the definition for the AXExcel class. Java native methods can be declared in any class, but the application will be simpler if the native interface is confined to a single class. AXExcel.java defines four native methods:

- ***OpenExcelWB*** creates a new Excel workbook and activates a worksheet.
- ***SetVisibility*** allows the Java interface to hide or show the worksheet.
- ***ExcelGraph*** creates a bar graph from values supplied via getCellValue.
- ***QuitExcel*** saves the worksheet and closes the Excel application.

Note that when the method modifier *native* is used, the method has no implementation. Java expects to find the implementation in a system load library (dll) that is loaded prior to any call to a native method. This can be done during object construction, but the more general practice is to use a static initializer as shown in Listing 1.

The final method in AXExcel is getCellValue, which is called by ExcelGraph to obtain values for the bar graph. getCellValue raises an option pane to allow the user to supply values. Entering a null value signals ExcelGraph to stop requesting values and create the graph.

Next, having compiled AXExcel.java to AXExcel.class and corrected any compilation errors, we can generate the AXExcel.h header file using the javah utility. From the command line, javah –jni AXExcel creates the header file needed for the C++ native implementation.

Creating the C++ Native Implementation (AXExcel.dll)

The starting point for implementation is the AXExcel.h header file shown in Listing 2. C++ programmers will immediately note the unique JNI types, which are defined in the jni.h file usually found in the ..\java\include folder. For a description of the JNI types, extractors and other operators, refer to the JNI section of the Java Tutorial. Our task is to implement each of these in a Win32 dll using the Excel COM interface. To simplify library dependencies, we won't use MFC.

Listings 3, 4 and 5 show the source code for AXExcel.cpp. In Listing 3 the necessary includes and imports are shown along with the DllMain function. The three import files provide the COM functionality for Excel and must be located in directories in the Include paths. The full syntax of import statements is as follows (see comments for Office 2000 in Listing 3):

```
#import <mso97.dll> no_namespace
rename("/DocumentProperties", "Docu-
mentPropertiesXL")
#import <vbeext1.olb> no_namespace
#import excel8.olb> rename(Dialog-
Box", "DialogBoxXL") rename("RGB",
"RBGXL") rename("DocumentProperties",
"DocumentPropertiesXL")
no_dual_interfaces
```

Next, the global IDispatch COM pointers for the Application, Worksheet(s) and Workbook are declared. The DllMain function is required to ensure that:

- The COM Library concurrency model is initialized as Multithreaded on attachment. The default is ApartmentThreaded, which will cause runtime errors.
- The COM library is detached at termination.

OpenExcelWB (Listing 4) demonstrates a number of important concepts. The first task is to extract the path and worksheet names from the jstring arguments using the JNI extractor GetStringUTFChars. Note that the result is a standard C-style string buffer that must be released with the ReleaseStringUTFChars method before the pointer loses scope. Next, we look for the open Excel application or, finding none, create a new Excel task, which sets the application dispatch pointer, pXL. This dispatcher is used to retrieve the Workbooks collection. If the path and wks arguments reference an existing workbook and worksheet, it will be opened in the next try block. Otherwise the resulting error will be caught and a new workbook and worksheet will be created. The method completes by activating the current worksheet, making Excel visible and releasing the buffers used for extraction.

SetVisibility is interesting primarily as a demonstration of variations of the Boolean type. JNI uses jboolean, with values JNI_TRUE and JNI_FALSE. Setting Excel's visible property requires a variant type with the VARIANT_TRUE or VARIANT_FALSE value.

QuitExcel's main functions are to save the workbook and exit the Excel Application, both of which are accomplished in the first try block. However, the last four statements are critical: unless the dispatch pointers are released and detached, the application won't terminate properly.

ExcelGraph method (Listing 5) is the most complex implementation in that it not only creates an Excel chart, it retrieves the data by calling the Java method, getCellValues. After clearing the cells and initializing the range to the cell A1, we locate the Java class with the GetObjectClass method, which returns a jclass object, caller. GetMethodID uses the class to locate getCellValues on the Java side, returning a jmethodID object method. Locating the method requires the class, a C-style string containing the method name, and the *exact* Java method signature. The JNI is unforgiving on this point, so it's a good idea to use javap with the –s option to get the precise syntax. For example, javap –s AXExcel displays (Ljava/lang/String;)Ljava/lang/String; for getCellValues, the necessary third parameter for getMethodID. If a valid method ID is returned, the do-while loop iterates until an empty (0-length) string is returned. First, it retrieves the cell ID, using the Range method GetAddressLocal, and creates a jstring representation of it for use as the getCellValue argument. We then call the Java getCellValue method, which should return a numeric

Author Bio

Allan K. Green owns and operates QualiNet Company in Hillsborough, North Carolina, providing training and consulting services in OO A&D design tools and languages, principally C++ and Java, custom courseware and instruction programs, and systems design and integration services. His previous experience includes information systems management and development for IBM.

value as a jstring. After conversion to a C-string this is inserted into the current cell, the active cell range is moved one cell to the right and the buffer is released. When the do-while loop terminates, the entire UsedRange is used to create a simple bar chart. The method terminates by releasing and detaching the range and chart dispatch pointers.

Compiling, Building and Debugging the dll

The example code was compiled as an MS Visual C++ 6.0 Win32 Dll project. In addition to the default preprocessor options, it's necessary to add _OLE32_ and _WIN32_DCOM to activate the automation options. During code debug it's helpful if the C++ project folder and dll target directory are the same as the Java class directory.

Debugging a hybrid application can be tricky if neither component is known to work. One approach is to design a minimal driver interface like the one shown in Figure 1. In this example the ExFrame constructor creates an instance of AXExcel, and the button ActionEvent handlers use that instance to call the native methods. The Java side can be debugged by building a C++ dll version with "stub" methods, i.e., methods that do nothing except provide valid return values. With a visual debugger like Symantec's Visual Café, the calls to the native methods can be confirmed before you tackle the intricacies of the COM code. On the C++ side, since AXExcel.dll is loaded by the AXExcel class, it must be debugged by executing jave.exe with the appropriate driver class. In this example the Visual C++ debug executable would be:

```
<Javapath>\Java\bin\java.exe ExFrame
```

Application

While this example is admittedly somewhat illogical (e.g., why call back to Java to perform data entry to a visible spreadsheet?), it raises some intriguing potential for reusing user-developed COM objects in addition to the standard COM interfaces like those provided by Microsoft Excel and Word. Theoretically, any COM object could be wrapped in a similarly constructed dll and called from Java via the JNI interface.

akgreen@akgreen.com

Listing 1: AXExcel.java

```
import java.lang.*;
public class AXExcel extends java.lang.Object {
static {
   try {
          System.loadLibrary("AXExcel");
 }
  catch (UnsatisfiedLinkError e)     {
          System.out.println(e.getMessage());
  }
          }
public String getCellValue(String cellName) {
   String inputValue =
       JOptionPane.showInputDialog("Cell" +  cellName+ ":");
       return inputValue;
   }
 public native boolean ExcelGraph();
 public native boolean OpenExcelWB(String path,String wks);
 public native void SetVisibility(boolean flag);
 public native void QuitExcel();
}
```

Listing 2: AXExcel.h

```
#include <jni.h>
#ifdef --cplusplus
extern"C"{
JNIEXPORT jboolean JNICALL Java_AXExcel_OpenExcelWB
  (JNIEnv *, jobject, jstring, jstring);

JNIEXPORT void JNICALL Java_AXExcel_SetVisibility
  (JNIEnv *, jobject, jboolean);

JNIEXPORT void JNICALL Java_AXExcel_QuitExcel
  (JNIEnv *, jobject);

JNIEXPORT jboolean JNICALL Java_AXExcel_ExcelGraph
  (JNIEnv *, jobject);
#ifdef --cplusplus
}
```

Listing 3: Necessary includes, imports and DllMain function

```
#include "AXExcel.h"
#include <comdef.h>
#include <stdio.h>
#include <objbase.h>
#include <windows.h>
#import <mso97.dll>     //mso9.dll for MS Office 2000
#import <vbeext1.olb>   //vbe6ezt.olb for VB6
#import <excel8.olb>    //excel9.olb for MS Office 2000
#pragma warning (disable:4192)
using namespace Excel;
_WorksheetPtr pSheet;
SheetsPtr pSheets;
_WorkbookPtr pBook;
_ApplicationPtr pXL;
BOOL WINAPI DllMain(HINSTANCE hinstDLL,
       DWORD fdwReason,
       LPVOID lpReserved )
{   switch( fdwReason )
{   case DLL_PROCESS_ATTACH:
       CoInitializeEx(NULL,COINIT_MULTITHREADED);
       break;
    case DLL_PROCESS_DETACH:
       CoUninitialize();
       break;
      }    return TRUE;
}
```

Listing 4: OpenExcelWB

```
JNIEXPORT jboolean JNICALL Java_AXExcel_OpenExcelWB
  (JNIEnv *jThis, jobject jObj, jstring path, jstring wks)
{
 const char* jbuf = jThis->GetStringUTFChars(path,0);
 const char* wkname = jThis->GetStringUTFChars(wks,0);
 jboolean rc = JNI_FALSE;
 try {
  if (pXL.GetActiveObject("Excel.Application.8")) {
   pXL.CreateInstance("Excel.Application.8");
  }
  WorkbooksPtr pBooks = pXL->Workbooks;
  try {
   pBook = pBooks->Open(jbuf);
   pSheets = pBook->GetSheets();
   pSheet = pSheets->GetItem(wkname);
   rc = JNI_TRUE;
  }
  catch (_com_error &) {
   pBook = pBooks->Add();
   pSheet = pBook->Sheets->Add();
   pSheet->Name = wkname;
   rc = JNI_TRUE;
  }
  pSheet->Activate();
 }
 catch (_com_error &e) {
  printf("Error: %d, Msg: %s",e.Error(),e.ErrorMessage());
 }
 Java_AXExcel_SetVisibility (jThis, jObj, true);
 jThis->ReleaseStringUTFChars(path,jbuf);
 jThis->ReleaseStringUTFChars(wks,wkname);
 return rc;
}
JNIEXPORT void JNICALL Java_AXExcel_SetVisibility
```

```
  (JNIEnv * jThis, jobject jObj,jboolean flag){
 try{
  if (pXL != NULL){
       if ( flag == JNI_TRUE)  pXL->Visible = VARIANT_TRUE;
   else pXL->Visible = VARIANT_FALSE;
  }
 }
 catch(_com_error &e) {
  printf("Error: %d, Msg: %s",e.Error(),e.ErrorMessage());
    }
}

JNIEXPORT void JNICALL Java_AXExcel_QuitExcel
  (JNIEnv *, jobject) {
 try {
  if (pXL != NULL) {
   pBook->Saved = VARIANT_TRUE;
   pXL->Quit();
  }
 }
 catch(_com_error &e) {
  printf("Error: %d, Msg: %s",e.Error(),e.ErrorMessage());
    }
 if (!pSheet->Release()) pSheet.Detach();
 if (!pSheets->Release()) pSheets.Detach();
 if (!pBook->Release()) pBook.Detach();
 if (!pXL->Release()) pXL.Detach();
}
```

Listing 5: ExcelGraph method

```
JNIEXPORT jboolean JNICALL Java_AXExcel_ExcelGraph
   (JNIEnv *jThis, jobject jObj){
 try{
  pSheet->UsedRange->Clear();
  RangePtr pRange = pXL->ActiveCell;
  int textlen;
  jclass caller = jThis->GetObjectClass(jObj);
jmethodID meth = jThis->GetMethodID(caller,
 "getCellValue",
    "(Ljava/lang/String;)Ljava/lang/String;");
 if (meth == 0) { return JNI_FALSE;}
 do {
  _bstr_t cellId =
                 pRange-
>GetAddressLocal(false,false,xlA1,false);
   jstring range = jThis->NewStringUTF(cellId);
   jstring value =
         (jstring)jThis->CallObjectMethod(jObj, meth, range);
   const char* jbuf =
                        jThis->GetStringUTFChars(value,0);
   textlen = jThis->GetStringUTFLength(value);
   pRange->Value = jbuf;
   jThis->ReleaseStringUTFChars(value,jbuf);
   pRange = pRange->Next;
  } while (textlen != 0);
  pRange  = pSheet->GetUsedRange();
  _ChartPtr  pChart  = pBook->Charts->Add();
  pChart->SetSourceData
   (pRange,&variant_t((long)xlColumns));
  pChart->Visible;
  if (!pRange->Release()) pRange.Detach();
  if (!pChart->Release()) pChart.Detach();
 }
 catch(_com_error &e) {
  printf("Error: %d, Msg: %s",e.Error(),e.ErrorMessage());
    }
 return JNI_TRUE;
}
```

JSERVER —continued from page 70

benchmark continually inserted additional clients and measured server perfomance as a function of the growing load. The benchmark was run for the Oracle JServer and the Sun HotSpot performance engine. The Visigenics 3.4 ORB was used in testing the HotSpot VM. Oracle JServer CORBA support is provided by an embedded portion of the Visigenics 3.2 ORB that performs IIOP marshaling functions. The Java application program was compiled in both VMs: in JServer, using the JServer Accelerator compiler; with the JDK, using HotSpot.

Results

To compare the relative scalability of the two servers, we compare the average round-trip time for a call to the "work" method as a function of the number of clients connected simultaneously to the server. On the Enterprise E-450 class machine, Oracle JServer was able to support over a thousand concurrent clients, providing flat and highly predictable response times until the full processing power of the machine was utilized. As additional clients were added, JServer performance degraded gracefully and predictably. In contrast, Sun's HotSpot Java VM provided highly unpredictable and nonlinear response times at about 65 to 70 clients. HotSpot was unable to leverage the full power of the machine, using less than 20% of the CPUs.

To scale the test even further and evaluate Oracle JServer's ability to exploit even more sophisticated hardware architectures, the same tests were run with Oracle JServer on a Sun Enterprise E-6500 class machine with 12 Ultrasparc CPUs, 4GB of memory and 100GB of fast hot-swap Ultra SCSI internal storage. Results continued as predicted – JServer scaled to support over 5,000 concurrent users while providing very consistent and predictable response times.

Conclusions

The ScaleServer benchmark clearly demonstrates the superior scalability of Oracle JServer over that of the Sun HotSpot Java VM. While HotSpot handles about 65 to 70 concurrent users before fundamentally failing (yielding poor and unpredictable response times), JServer provides fast and highly predictable response times as loads on the system increase to a thousand and even more than five thousand concurrent users.

A grave limitation of the HotSpot VM's performance was that it was able to leverage only about 15 to 20% of the E-450's CPUs. Thus, since CPU wasn't a constraining factor, HotSpot couldn't support additional clients by using additional CPUs. In contrast, JServer consistently scaled linearly up to the capacity of the hardware, exploiting the additional CPUs that were added as the test was scaled from a four-way to a 12-processor system.

Naturally, the more processing a system does, the more it will benefit from running efficient code. While the HotSpot dynamic compiler may be effective for single-user applications, much of its benefit is unrealized in a multiuser environment. The Oracle JServer Accelerator compilation technology on the other hand produces excellent results in highly concurrent scenarios. JServer's compilation technology works very efficiently with the server's shared memory architecture to provide excellent performance under multiuser configurations.

Oracle JServer Qualifications

Oracle JServer was designed and implemented to serve large numbers of concurrent users, and it naturally incurs overhead costs for its scalable infrastructure. Its advantages become more and more apparent as the number of clients increases. Consider an analogy to a delivery company that routinely transports lots of packages, using a truck. If it needs to transport only one or two packages, a car would probably be more appropriate. Although the truck accelerates less quickly than the car, its superior capacity soon becomes critical as loads increase. The car will have to make multiple trips once the third, fourth or fifth package doesn't fit in its trunk. For heavier loads the truck is the faster and more economic solution – especially a truck with a powerful engine!

Summary and Benefits

The component-oriented, memory-safe Java language is becoming the choice of many companies looking to develop and deploy large-scale enterprise applications. As these companies begin to leverage productive server-side Java technologies such as servlets and Enterprise JavaBeans, a new breed of JVM is necessary to provide sufficient scalability. Oracle JServer is currently the only Java platform in the industry that meets all the requirements of an enterprise-class Java server. Specifically, it:

- Supports thousands of concurrent, stateful, conversational clients
- Leverages state-of-the-art memory management and garbage collection algorithms
- Minimizes memory footprint required per session to between 35 and 50K as compared to 1 to 3MB for typical Java VMs such as the JDK
- Exploits advanced hardware architectures, such as SMP and MPP hardware clusters, scaling linearly as new processors are added to the cluster
- Provides superior performance via JServer Accelerator technology

Oracle JServer's scalability was confirmed by a ScaleServer benchmark in which JServer comfortably outperformed its competition by giving consistent performance on multiple platforms and by providing hundreds of times better scalability than rival VMs.

In addition, its scalable architecture offers three fundamental benefits concerning the development, deployment and management of Java applications:

1. ***Ease of programming***: The JServer architecture relieves the Java application developer of the responsibility of writing multithreaded Java code to achieve scalability. Using Java threads is error-prone, poses potential security issues in a server environment, and incurs fundamental scaling limitations. With JServer, application developers don't need to write multithreaded Java code. Since multithreading facilities and scheduling are provided by the JServer platform, the application developer can write the application as if it were to run for a single user. JServer will scale the application to automatically serve many multiples of concurrent users.
2. ***Ease of manageability***: Oracle JServer provides the industry's most manageable Java server platform. Some non-JServer implementations spawn new VM instances to support large volumes of users, an approach that wastes memory by introducing redundancies and simultaneously creates a need to manage multiple JVM instances. In contrast, a single JServer instance scales to the capacity of its host hardware, making efficient use of memory and requiring administration of only a single server instance.
3. ***Smooth upgrade path***: Since JServer is able to exploit the capacity of simple or sophisticated hardware systems, it provides an efficient upgrade path as user loads increase. It works efficiently with both 32- and 64-bit operating systems across all major hardware platforms. Oracle JServer was designed as an enterprise-caliber platform to run server-side Java applications. Its architecture and design facilitates application development, maintenance and upgrade, making it the industry's most scalable and most highly productive Java platform.

AUTHOR BIO

Jeremy Lizt is a senior product manager in the Java Platform Group at Oracle Corporation.

jlizt@us.oracle.com

ADVERTISING INDEX

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

Serving Business Applications

Application servers occupy a prominent place in multitier computing as well as in the world of Java-based e-commerce

WRITTEN BY
AJIT SAGAR

In the world of distributed computing, the industry has latched on to another snazzy, buzzword-compliant, omnipotent entity, the Application Server, also known affectionately as the App Server. Here's the sales pitch. You want a robust system? Fault tolerance? Load balancing? Multithreaded transaction support? Well, don't reinvent the wheel. If you're developing an application that solves a particular business problem, concentrate on solving that problem and on developing that application. Don't waste precious resources trying to focus on solving a problem that's outside your area of expertise. After all, if you were in the business of writing compilers, that should have been the highlight of your job description. Similarly, if creating a framework for load balancing, fault tolerance and the like is not your job, let someone else do it.

When I look at the stages of my career thus far, I fondly remember the days when I designed and deployed my first commercial product, an electronic private branch exchange (EPABX) with the logic written entirely in – dare I say it? – Z80 assembly. If I mention assembly now, people say, "You must be older than you look." I don't program in assembly anymore, of course; we Java programmers frown on such low-level programming, don't we? Why would you want to deal with those tedious, low-level, limited instruction sets when Java compilers are more than willing to do all that work for you? If your application doesn't need low-level programming, the only reason you'd indulge in such activity is because you think you can do a better job of writing compiler-type code than companies that specialize in it.

This month in e-Java we'll take a look at the role played by application servers in multitier computing as well as the impact they'll have in the world of Java-based e-commerce. We'll also examine the categories of application servers that exist today and the facilities offered by these servers. Finally, we'll look at the build versus buy issue vis-à-vis application servers and discuss how one should plan to migrate from the former to the latter.

Evolution of Multitiered Architectures

Application servers are a part of the evolution of computer topologies from the mainframe to the *n*-tier distributed computer model. The earliest computing model was the single computer mainframe. This evolved into the client/server model. Client/server refers to a topology that emerged in the 1980s in which the computer processing tasks were separated into two types of computers connected to each other over a network. Typically, this involved the client, which was the PC, and a server, which was a mainframe. The client was responsible for requesting services from the server and displaying it to the user after applying the appropriate business rules and presentation logic. The services requested were typically served up via a database management system.

The next stage in evolution was the three-tier model. This model split the responsibilities of the client into two tiers, one responsible for the presentation, the other for applying the business rules used to serve up the data to the user. The new tier of computing was given the name middle tier. The decoupling of business rules from the presentation logic abstracted the user interface from changes in the mechanism of interpreting data served up by the database servers.

The middle tier thus acted as a broker between the data source and the data presentation. It was responsible for several functions – transaction processing, business rule validation, data transport between the presentation tier and the database server, security across the tiers, and so forth. The middle tier achieved

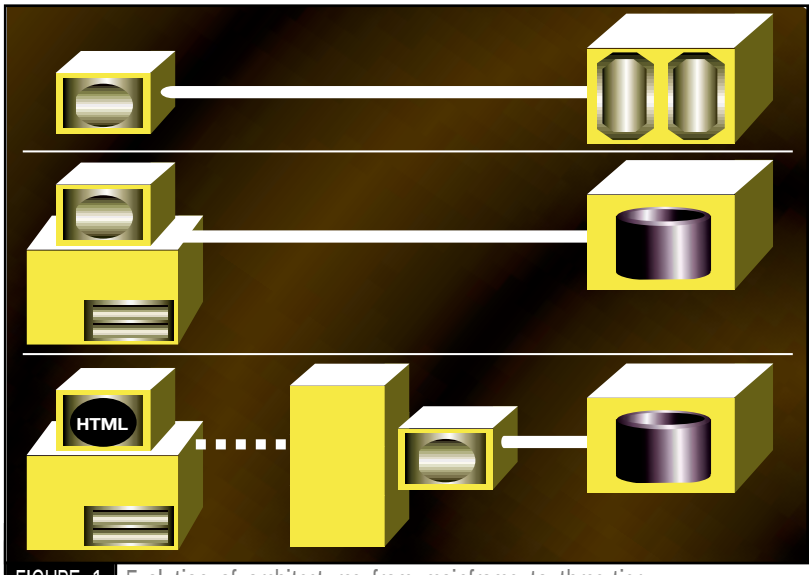


FIGURE 1 Evolution of architectures from mainframe to three-tier

these functions through various mechanisms such as transaction processing monitors, messaging systems, message servers and ORB (CORBA/DCOM) architectures. As the three-tier model became popular, it became obvious that it wasn't feasible for one organization to manage all the complexities of the middle tier and focus simultaneously on the business problems it was striving to solve. As a result, the industry saw the emergence of companies that started providing the middle-tier services packaged as reusable third-party tools that could be applied across various industries. These packaged services were offered via a new class of computer servers called application servers.

Figure 1 illustrates the topologies discussed above.

Application Server in the Middle Tier

Application servers represent typical middleware. An application server may be defined as a server program residing in a middle-tier machine that provides the business logic and services for an application program. The middle-tier machine is part of a distributed network topology, and the application program resides on a client machine that presents the data to the user. In today's world application servers are closely tied to Internet technologies and the client is typically an Internet client. Enterprise-level Java application servers are used to offer Java and browser-based applications, thus supporting the thin-client paradigm. In the *n*-tiered distributed topology, application servers enable modularity and componentization of critical applications by spreading the functionality across various hardware tiers.

The data served by application servers is typically served over the Web. Application servers today are almost always used in conjunction with a Web server, forming the combination called a *Web Application Server*. The Web server is responsible for forwarding requests from the client to the application server and returning HTML content back to the Web client. The Web server uses several alternative approaches to achieve this, including CGI, ASP (Active Server Pages), Java Servlets and JSPs (Java Server Pages). The app servers may also use distributed service protocols such as CORBA, RMI and MTS to achieve object-based communication. If we look at the distributed *n*-tier computing model from the perspective of application servers, the Web application server defines the following divisions of a distributed application:

- *A front-end Web browser-based GUI,* which may be at a PC or network computer or in end-tier devices such as the Palm Pilot, Windows CE or Webtop machines in the emerging consumer device market
- *A business logic application suite that resides in the middle tier,* usually housed in an intranet or a middle-tier server farm
- *A back-end database tier and transaction server,* which is also responsible for interacting with legacy systems

Web Application Server Services

What services are typically encapsulated by an application server? I've mentioned business logic several times. Implementing and executing business logic is certainly the main reason for the existence of application servers. In addition, the application servers' main function is to make sure that developed and deployed applications are scalable, reliable and accessible. Application servers ensure this by offering some or all of the following:

- Improvement in performance and reliability of Web-enabled state and session management, including support for connection pooling
- Database transaction management, ensuring database integrity
- Framework for supporting component development programming models
- Multiplatform and multilanguage support
- Powerful management tools
- Support for security across the multiple tiers of a distributed application
- High performance and scalability support, including resource pooling, load balancing and connection pooling
- Fault tolerance in order to guarantee the availability of application services, including support for clustering for failover recovery
- Directory services for user roles and access permissions
- Workflow tools for defining business process workflows

The current breed of application servers offers these functions as a set of services. Some are offered as core functions; others are add-on services.

Splitting the Middle Tier in E-Commerce Applications

Application servers may be classified into different categories based on the services they offer and the role they play in the application architecture. In an enterprise-level application sometimes more than one application server will coexist and interact to provide the services needed by the application. One emerging trend in such distributed applications is splitting the responsibility into a front-end Web application server and a middle-tier business logic application server. The front-end application server – an example is ColdFusion from Allaire – is responsible for managing the C2B (customer-to-business) interaction. Such servers offer the functionality typically needed for creating virtual storefronts. In addition to the core services, other modules supported by these servers are payment modules, credit authorizations and user profiles. The middle-tier server is responsible for interacting with back-end systems and for executing business logic. Examples are BEA's WebLogic application server and Netscape Application Server. These servers aren't typical-

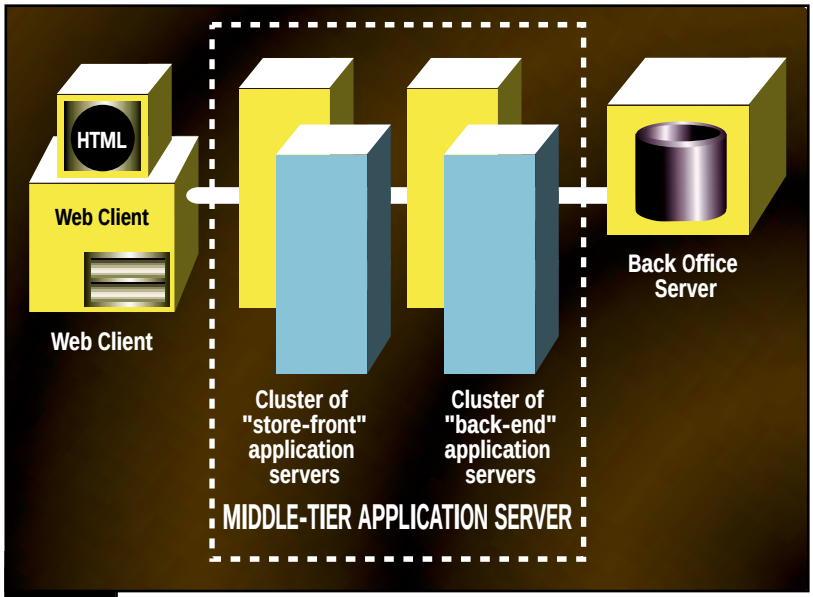


FIGURE 2 Splitting the middle-tier application server

Author Bio

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

ly concerned with the end customer, but rather with B2B (business-to-business) applications. Figure 2 shows an architecture that uses this division of labor between the two levels of application servers.

Java and Application Servers

Java middleware has gained a lot of ground in the past year. Since the announcement of J2EE, the definition of what the API standards for Java-based middleware are going to look like is much clearer. This has helped solidify the ground for Java application servers. Typically, application servers implement a messaging layer, transaction services, business logic and security. The APIs for this are defined by JMS, JTS, EJBs and Java's security model. The good news is that Java standards will help bring ubiquity to the world of application servers in the marketplace. The bad news is that sometimes it takes more time for the standards to solidify and the app server vendors are forced to provide their own implementations to meet the needs of the market.

Build vs Buy

Several companies specialize in building app servers that provide generic services for distributed business applications. It really doesn't make sense for firms to waste developer resources by trying to create these services in-house, chiefly because it's not their area of expertise. However, cost and time to market are considerations that force companies to build their own services. In addition, if the requirements for the applications under development aren't satisfied by third-party application servers, developers have no choice but to build the functionality in-house. Still, the modules and services should be developed with the understanding that they'll be replaced by a stable third-party server whenever it becomes available.

ajit@sys-con.com

JDJ Consulting p/u

JDJ
www.jdjs

Store
store.com

Employment Ad

BEA WebLogic Server 4.5 Released

(*San Jose, CA*) – BEA Systems, Inc., has released BEA WebLogic Server 4.5, an implementation of Sun Microsystems' J2EE platform standard. Through support for J2EE and other new features, BEA WebLogic Server 4.5 makes it easier for companies to develop and deploy transaction-based e-commerce applications. www.beasys.com/weblogic.html

KL Group Releases JProbe 2.5 ServerSide Edition

(*San Jose, CA*) – KL Group Inc., a leading provider of Java components and advanced development tools, has begun shipping JProbe 2.5 ServerSide Edition. The award-winning JProbe Suite now offers enhanced capabilities for performance tuning, memory debugging, code coverage and thread analysis of Enterprise JavaBeans and servlets running under popular application servers and Web servers. www.klgroup.com

New Licensees for Insignia Solutions' Jeode Platform

(*Fremont, CA*) – Insignia Solutions has announced that four additional companies have licensed the Jeode platform, Insignia's faster implementation of Sun's EmbeddedJava and PersonalJava specifications, designed for embedded devices. It's also the first independently developed implementation to pass Sun's EmbeddedJava technology and PersonalJava platform compatibility tests to earn the distinction of "Sun Authorized Virtual Machine." www.insignia.com

Sun Makes New Java Claims Against MS

(*San Jose, CA*) – Sun Microsystems Inc. attorneys say that Microsoft Corp. found new ways to circumvent a preliminary injunction requiring Microsoft products to comply with Sun's Java even before the injunction was vacated by the Ninth Circuit Court of Appeals in August. At a recent hearing Sun asked U.S. District Judge Ronald Whyte to reinstate the injunction based on what Sun attorney Rusty Day called "Microsoft's most recent acts of unfair competition." The Appeals Court vacated the injunction because it said Whyte did not adequately show that Microsoft violated Sun's copyright, although the Court did find that Microsoft likely breached its contract.

SYS-CON Named America's Fastest Growing Publishing Company by Inc. Magazine

Pictured in SYS-CON offices June 1999 (l to r): Amanda Moskowitz, Ignacio Arellano, Christine Russell, Jim Morgan, Fuat Kircaali, Robert Diamond, Nicholas De Jesus, Chad Sitler, Robin Groves, Mary Ann McBride, Jaclyn Redmond, Megan Ring, Alex Botero, M'lou Pinkham, Cheryl Van Sise, Robyn Forma, Carmen Gonzalez (not shown: Sian O'Gorman, Bruno Decaudin, Sean Rhody, Ajit Sagar, Alan Williamson, Ann Marie Milillo, Scott Davison, Bahadir Karuv, Nancy Valentine, Aarathi Venkataraman, Digant Dave, Eli Horowitz)

(*Pearl River, NY*) – SYS-CON Publications, Inc., publisher of ***Java Developer's Journal***, was ranked 194th on *Inc.*'s list of the 500 fastest-growing privately held companies in the U.S.

SYS-CON Publications was the only company in the publishing industry to make it to this year's list, which, not surprisingly, was dominated by computer-related companies (46% of the 500). www.inc.com/500

Theory Center and Navidec Partner

(*Boston, MA*) – The Theory Center, Inc., a leading provider of EJB component-based solutions, has announced a partnership with Navidec, Inc., a leading provider of innovative e-business solutions and services. Under the terms of the agreement, Navidec will resell Theory Center's JumpStart product, a pure EJB component-based software product designed to help companies build flexible, Web-enabled applications in 30 days. www.theorycenter.com www.navidec.com

Free JDataStore Development License Available from Inprise

(*Scotts Valley, CA*) – Inprise Corporation has announced JDataStore, a pure Java database management system that delivers platform independence and portability in a small-footprint package. JDataStore is available free for download for a limited time from Inprise's Web site. www.borland.com/jdatastore

SilverStream Software Creates Partner Advisory Council

(*Burlington, MA*) – SilverStream Software, Inc., has created a Partner Advisory Council to provide its VAR, consulting and training partners with an interactive forum. The council will meet quarterly to share ideas with SilverStream on planning for future product releases and customer support programs for its award-winning application server. www.silverstream.com

Together/J Version 3 Released

(*Raleigh, NC*) – TogetherSoft LLC, formerly Object International, released Together/J version 3, delivering a host of new features for visual UML development across the enterprise. Analysts, architects, designers and developers can download the free TJ Whiteboard edition from www.togethersoft.com.

Y2K Liability Limits Pass House, Head for Senate

(*Washington, DC*) –The House of Representatives recently passed HR 775 (236-190), the first bill providing liability limits for Y2K-related litigation. It now heads for the Senate. National Association of Computer Consultant Businesses board member and president of Commercial Programming Systems Al Strong says, "The NACCB's position is that liability protection is necessary, but that damages awarded should be limited to the value of the work completed." www.cpsinc.com

SYS-CON Radio/JDJ Media Cosponsor of December Java Conference

(*Pearl River, NY*) – SYS-CON Radio and ***Java Developer's Journal*** have been named media cosponsors of the third annual Java Business Conference that will take place at the Jacob K. Javits Convention Center in New York City December 7–9. According to publisher Fuat Kircaali, "***JDJ*** has been participating at this conference since its inception, and we are delighted to be media cosponsors this year." This past June the publishing firm and its radio adjunct were also media cosponsors of JavaOne, the largest developer conference of the year.