SILVERSTREAM SPECIAL FOCUS ISSUE

# JAVA™ DEVELOPER'S JOURNAL

JavaDevelopersJournal.com

Volume: 4 Issue: 3, 1999



SYS-CON PUBLICATIONS

JAVA DEVELOPER'S JOURNAL

## EDITORIAL ADVISORY BOARD

Ted Coombs, Bill Dunlap, David Gee, Michel Gerin, Arthur van Hoff, Brian Maso, John Olson, George Paolini, Kim Polese, Sean Rhody, Rick Ross, Ajit Sagar, Richard Soley, Alan Williamson

*Editor-in-Chief:* Sean Rhody
*Art Director:* Jim Morgan
*Executive Editor:* M'lou Pinkham
*Managing Editor:* Brian Christensen
*Production Editor:* Hollis Osher
*Editorial Consultant:* Scott Davison
*Technical Editor:* Bahadir Karuv
*Product Review Editor:* Ed Zebrowski
*Industry News Editor.* Alan Williamson
*E-commerce Editor.* Ajit Sagar

## WRITERS IN THIS ISSUE

Steve Benfield, Brad Cooley, Bruce H. Cottman, Claude Duguay, Rachel Gollub, David Jung, George Kassabgi, Doug Porter, Jim Redman, Sean Rhody, Chad Ruff, P.G. Sarang, David Skok, Robert Tiffany, Alan Williamson

## SUBSCRIPTIONS

For subscriptions and requests for bulk orders, please send your letters to Subscription Department

**Subscription Hotline: 800 513-7111**

*Cover Price:* $4.99/issue
*Domestic:* $49/yr. (12 issues) *Canada/Mexico:* $69/yr.
*Overseas:* Basic subscription price plus airmail postage (U.S. Banks or Money Orders). *Back Issues:* $12 each

*Publisher, President and CEO:* Fuat A. Kircaali
*Vice President, Production:* Jim Morgan
*Vice President, Marketing:* Carmen Gonzalez
*Advertising Account Manager* Robyn Forma
*Advertising Assistant* Megan Ring
*Accounting:* Ignacio Arellano
*Graphic Designers:* Robin Groves, Alex Botero
*SYS-CON Radio Editor* Chad Sitler
*Webmaster:* Robert Diamond
*Customer Service:* Sian O'Gorman, Paula Horowitz, Ann Marie Milillo
*Online Customer Service:* Mitchell Low

## EDITORIAL OFFICES

SYS-CON Publications, Inc.
39 E. Central Ave., Pearl River, NY 10965
Telephone: 914 735-7300 Fax: 914 735-6547
Subscribe@SYS-CON.com

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



*Worldwide Distribution by*
Curtis Circulation Company
739 River Road, New Milford NJ 07646-3048 Phone: 201 634-7400



SYS-CON PUBLICATIONS

FROM THE EDITOR

*Sean Rhody, Editor-in-Chief*

# Stuck in the Middle with You

About a year ago, in a magazine not too far away, I wrote an article called "Middle-Tier Madness." A year and several languages later, we're back at the middle-tier stage again. Distributed computing is one of my main areas of interest, so my concern with the middle tier shouldn't surprise anyone.

I've done work with all three of the major standards for distributed work – CORBA, COM and EJB. I much prefer EJB. For those of you that don't recognize the acronym, it stands for Enterprise JavaBeans, a specification from Sun Microsystems that describes how to construct server components for a Java-based server.

We can do Java server components for any of these standards (although COM is more trouble than it's worth), and can likewise do Java clients that can speak to servers of this type. The selection of the standard has a great deal to do with the overall environment in the company that's going to build a distributed system. If the company is a Microsoft shop, COM is almost a no-brainer. If there is a mix of computers – UNIX, Windows and Macintosh – COM is probably less of a choice, so it comes down to CORBA or EJB. The EJB specification was released a year ago (March 1998) and the first servers began to appear toward the end of the third quarter, so until recently the only choice was CORBA. And it may still need to be the choice, depending on the environment.

> ***"Developing distributed systems will make a perfectionist out of you. It has to."***

One of the greatest strengths of EJB is that it's an all-Java environment. The strength of that approach lies in the fact that by using a single language and programming paradigm, EJB allows for a much richer approach to data transfer and marshaling. No abstract IDL is required to create components; you simply write them in Java. To pass complex objects from one machine to another, you implement Serializable, which is trivial, since that interface has no methods.

That strength is also the biggest weakness of EJB. One of the biggest strengths of CORBA is the ability to integrate multiple languages. In many large companies Java is only one of several languages that will be used for development. Typically, a company uses C/C++, PowerBuilder, Visual Basic and sometimes COBOL. For companies that need to continue to use these languages, EJB offers only a limited amount of usefulness. If a company can commit itself to moving to Java as its main language, EJB is a good choice, as you can work around the other languages to a certain extent and rewrite as time permits. Otherwise CORBA is your best bet.

Whatever you choose, you're in for some interesting times. I guarantee you that developing distributed systems will make a perfectionist out of you. It has to. In earlier times, errant code could crash only one system at a time. Now, bad code in the middle tier has the capability of affecting thousands of people. It's also much more difficult to debug, as the number of possible problems increases dramatically due to the additional layers of communications and the ambiguities present in some of the specifications. If you want to make it work, you have to tighten up your overall code process. Nothing less than zero defects will work.

I hope you'll find information in this issue to aid you in the pursuit of that goal. Let me know – I have to start planning next year's version of this column.

**About the Author**

*Sean Rhody is the editor-in-chief of* **Java Developer's Journal**. *He is also a senior consultant with Computer Sciences Corporation, where he specializes in application architecture – particularly distributed systems. He can be reached by e-mail at sean@sys-con.com.*

GUEST EDITORIAL

*Jim Redman*

# Java Makes a Move Back – into Embedded Systems

The modern manufacturing facility or laboratory often appears as thousands of points of information, scattered in and among hundreds of pieces of machinery and other equipment. Good integration of these information sources provides for an ongoing challenge.

The goal of systems integration is to get these machines networked and accessible in some uniform manner throughout the manufacturing facility, and to provide information in human-readable form by uniform, nonproprietary and platform-independent means. Java and Internet technologies can help to bring this goal within reach – leaving it as an implementation project and not as the research project it appears to be today.

The first step in this process is to blend Java and embedded Web servers with traditional factory automation methods. This may mean replacing outdated equipment (or the controllers on older equipment) with a new Ethernet and Web-enabled controller or – and this is often more cost effective – augmenting existing systems with Java-based monitoring and control systems. These retrofitted solutions add to the existing functionality of production lines without significantly impacting the current operation. These small controllers are called *embedded systems* and almost always run with no more than a minimal non-graphical user interface. The challenge with these systems is information access.

Embedded systems are everywhere, but they mostly go unnoticed in our daily lives. The most obvious ones are kitchen devices such as microwaves, bread makers, toaster ovens and so forth, but this also includes the systems controlling the under-the-hood operations in most modern vehicles. Call me a cynic, but I don't expect toasters or bread machines to have an Ethernet connection or Web access anytime soon. But embedded systems within factories present a completely different picture.

When most people think of Web servers serving up Java applets, they think of the large, fairly powerful computer systems on the Internet. However, the simplicity of the Web server and the architecture of Java make it an obvious match for the world of embedded systems. Many of these systems are using modern versions of the old 8- or 16-bit processors, such as the old Zilog Z80 and Intel 8088 (the chip in the original IBM PC). The cost of adding a full, traditional user interface, complete with hardware, to these small systems is prohibitive, yet information is often available that would merit more display and access.

In factory automation these embedded systems often control complex processes and equipment. They're used in almost every step of every manufacturing process. The robots that build cars use embedded computers; almost every chemical process from refining oil to filling toothpaste tubes uses embedded real-time controllers. As you might imagine, these processes are not simple and to optimize and monitor these systems requires some way to get the information out of the controller. The first challenge is simply to get a user interface to these systems. Java applets and an embedded Web server make this process relatively simple. The existence of Java-based toolkits specifically designed for this type of automation display can make building these screens fairly straightforward. Now the information for the display becomes a part of the equipment, owned by and embedded within the controller. Logically, this is how it should be – the information and access to the information centralized in one integrated package and accessed through Java applets from anywhere.

Many of these embedded systems are controlled by real-time operating systems, and the vendors of these systems – almost without exception – provide support for embedded Web servers. There's also a whole class of industrial controllers, called Programmable Logic Controllers or PLCs, that run with very small amounts of memory (often less than 64 K bytes) that also support embedded Web servers. The growing acceptance of Java in these applications shouldn't surprise anyone who can remember the ancient history of Java (almost nine years ago). This is the application it was originally designed for: providing a means for all sorts of consumer devices and computers to communicate with each other even though they're based on all kinds of different microprocessor chips, each with very little memory space.

The future is bright for Java in these applications. While the efforts to create a specification for real-time Java are ongoing, perhaps more slowly and with more difficulty than may have been hoped for, the use of Java as a near-real-time monitoring tool is growing. There's an impressive trend toward equipment retrofits based on Java and embedded servers. Organizations exist to produce embedded Web servers and user interfaces for existing equipment. There's also a move toward server-side Java for embedded systems that will allow some logic to be provided in Java on the embedded computer. This includes such things as protocol stacks that are specific to industries such as SECS/GEM in the semiconductor industry, and LECIS in the pharmaceutical laboratory and drug discovery industries.

So when someone mentions an applet, don't just think of the big guys. Think of the applets running from 8-bit microprocessors with memory space of 128 K (or even smaller) that could happily be providing information access in the factories making those big 64-bit systems as well as the many other things we use and enjoy in our daily lives.

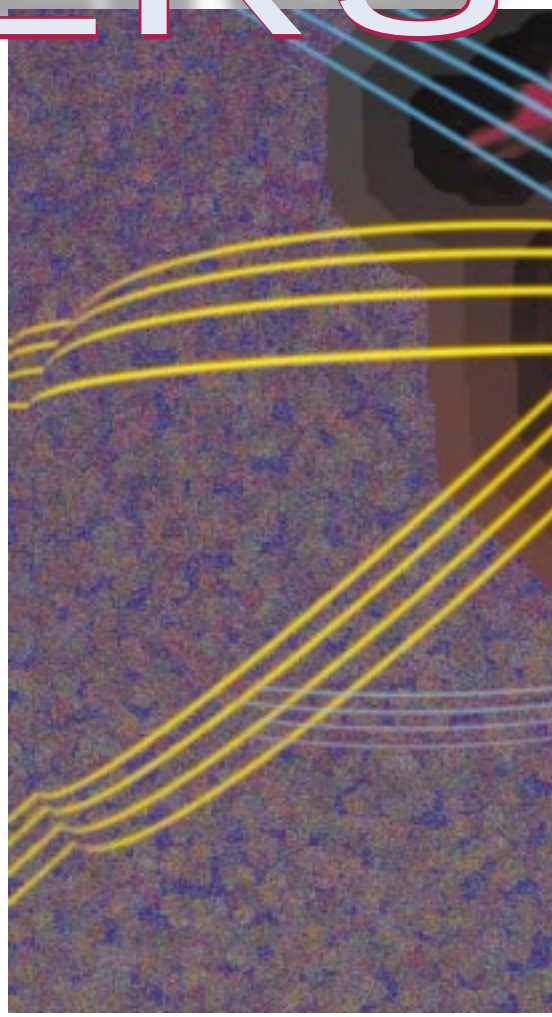
**About the Author**

*Jim Redman is the president of ErgoTech Systems, Inc., a company focused on developing Java applications and toolkits for plant-floor automation. This includes links to low-level systems and hardware, and also network links – including CORBA support – for enterprise distribution of factory automation information. He may be reached at JRedman@ergotech.com.*

# SERVICE BROKERS

## *Transforming legacy assets into Java services*

*by* **Bruce H. Cottman, Ph.D.**



### The Challenge of Java Integration with Legacy Assets

Organizations are developing a new class of electronic business applications, using Java to capture new business logic. In some cases these e-business applications ,have gone beyond just successful deployment to redefining the competitive landscape. Examples are package tracking, home banking, travel services, automobile purchasing, home shopping, customer support, billing, global securities trading and publishing. The driving forces behind these applications are increased competition in growing markets and the need to reduce costs in services and support. Because e-business is a significant part of the business image in competitive markets, demand for higher performance and reliability is continually increasing.

Successful e-business applications must meet demanding performance, scalability and reliability requirements that previously have been found only on the mainframe. A short list of required Java e-business server capabilities includes multithreading, load balancing, resource management, state persistence, connection pooling and fault-failover. Achieving any one of these capabilities is a large development cost.

JavaSoft, along with leading partners like IBM, are meeting the challenge of Java e-business requirements with Enterprise JavaBeans (EJB). The key capabilities of the EJB framework are:

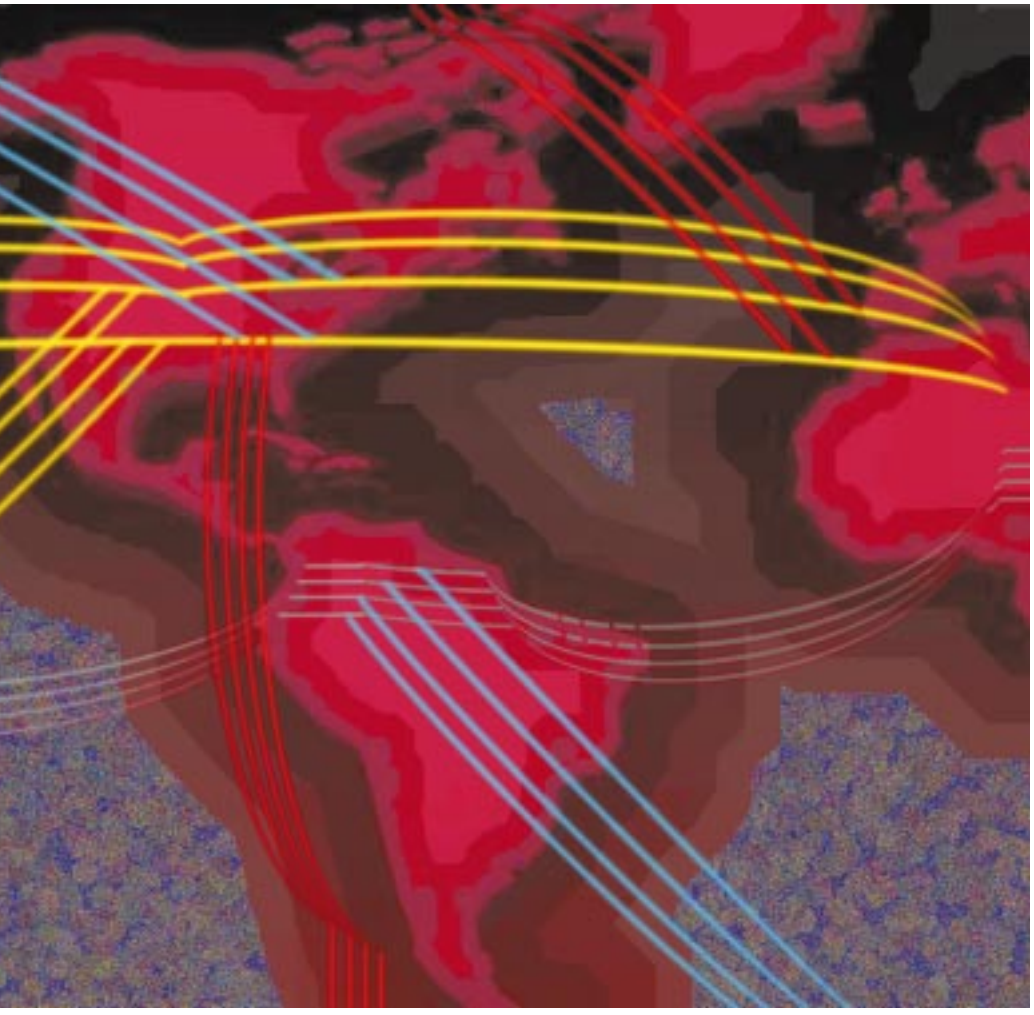
- E-business logic is packaged as a JavaBean component that executes within an EJB server. The EJB server platform supplies thread management, state persistence and other enterprise capabilities to the JavaBean. The e-business application developer gains performance and scalability from the EJB server platform facilities.
- The EJB server platform supplies data access (JDBC), transactions (JTS), messaging (JMS), directory (JNDI) and other fundamental services as standard Java interfaces. The e-business application developer gains Plug and Play component software development productivity by using tools that support EJB service specifications. Component software developer productivity, previously available only to the visual JavaBean developer, can now be achieved by the server-side Java developer.

A complete, well-defined set of interfaces is a major strength of Enterprise JavaBeans. The service interface limits the complexity that's visible to the application implementation while enabling the application to exploit the full power of the distributed object system. Server-side component services are fundamentally new because in the past, the integration of service-based capabilities always had to be custom crafted. Without a component framework such as EJB, there was no way of predicting – other than empirically – how the capabilities of services from different vendors could be integrated. The EJB server standardizes how the behavior of one Java service interface will change when

another Java service interface is also being used. Business logic and different service interfaces become Plug and Play application components (beans!).

Enterprise JavaBeans is a complete platform for Java e-business applications. But what about the world's existing business logic? Can we afford to build new e-business applications that ignore existing production systems? The answer, of course, is "NO!"

The most critical challenge facing Java developers today is with integrating legacy data and applications. The world's information technology (IT) investment (sometimes referred to as *legacy*) is realized in mission-critical business logic that controls such critical systems as fund transfers, stock exchanges, transportation scheduling, payroll and the worldwide distribution of goods and services. It's estimated that there's over $5 trillion in IT assets that conduct the day-to-day business of the world's economy. In 1997 the U.S., Europe and Japan spent over $600 billion on the maintenance and upgrade of these IT assets.



The roadmap for legacy integration is inherent in the specification of the EJB platform. Access to legacy assets should be accomplished using Java platform services such as Java Database Connectivity for legacy data and Java Transaction Service for legacy transactional logic. By integrating through Java platform services, legacy business logic and data appear as native JavaBeans to the e-business application developer.

Let's explore the technical challenge of transforming legacy IT assets into Java services.

## The Challenge of Transforming IT Assets into Java Services

E-business applications are inherently distributed and multitier. In a multitier distributed application, the logical architecture is typically three-tier architecture, with business logic separate from presentation and data access logic. Mid-tier Java business logic requires access to back-tier IT assets that:

- Provide high performance throughout for both small (one to hundreds of rows) and large (thousands or millions of rows) amounts of data
- Sustain high performance throughout under peak concurrent user loads as well as trivial user loads
- Maintain high availability greater than 99.9%
- Support centralized management and administration using standard tools
- Use standard interfaces to reduce the cost of development and maintenance

Java developers are discovering that existing middleware is not designed to support hundreds or thousands of connections from distributed business logic servers to databases. For example, client/server middleware, such as ODBC, was designed to connect a single resource-rich desktop client to a database server. An entirely new kind of middleware is required that can manage large concurrent sessions loads of an e-business application server.

A new breed of middleware must meet the performance requirements of e-business and it must also meet the rigid integration limitations of legacy IT assets. By introducing a new tier, a middleware tier between the e-business logic server and the legacy asset tier, we have a platform to add all the facilities that we need. A middleware tier can enhance the legacy asset by adding such facilities as load balancing, connection pooling, multithreading and distributed failover. I-Kinetics calls this middleware architectural approach the Service Broker.

A good Service Broker isn't just a pass through proxy for the Java service interface. What looks like a simple implementation of a Java service interface may require the Service Broker to coordinate the implementation of several features within the limitations of a legacy interface. For example, obtaining a connection to a legacy application may appear straightforward. However, if that connection is to be pooled, load-balanced and encrypted, and to support a distributed two-phase commit, then the Service Broker will have to implement most of the distributed connectivity management facilities for the legacy asset.

Adding new capabilities is a significant advantage of a Service Broker architecture. For example, I-Kinetics DataBroker implements the JDBC service with additional capabilities such as multithreaded data streaming, load balancing and connection pooling. You can expect Service Broker architectures to be able to offer the best JDBC 2.0 extension implementations for transaction resource management, batch updates and backward and forward scrolling result sets. Additionally, SNMP management, query caching, messaging and object persistence are advanced capabilities a Service Broker can offer across JDBC, JTS and JMS services.

The impact of using a Service Broker on

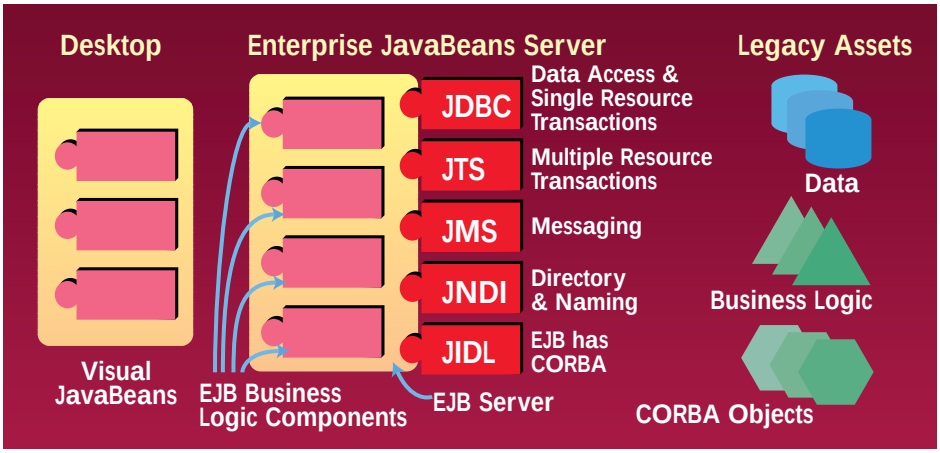



*Figure 1: The Enterprise JavaBean framework*

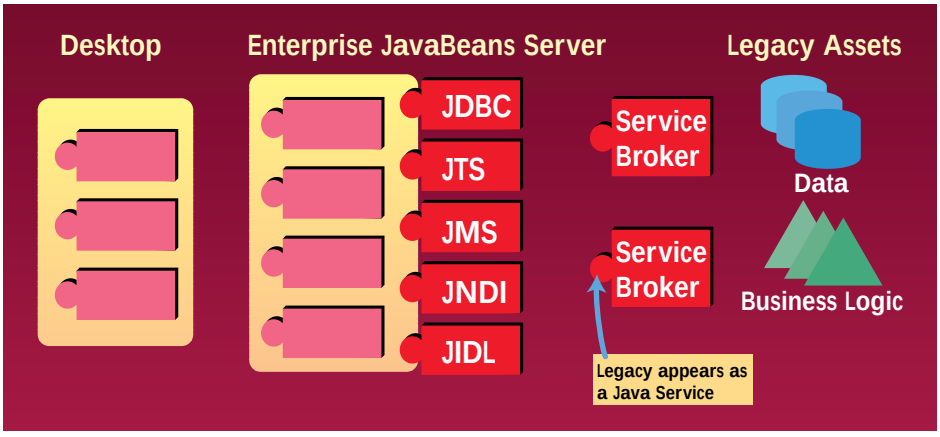



*Figure 2: The Service Broker transforms legacy assets into standard Java services.*

the full life-cycle cost of the e-business application can be enormous. For legacy system integration projects, 40 to 70% or more of the development cost may be eliminated because:

- A Service Broker transforms the legacy data structures into standard Java data formats.
- A Service Broker offers standard application service interfaces that are widely supported by tool vendors. A Java development tool can be used for the complete development life-cycle -- from prototyping to full deployment of the multi-tier e-business application.
- Complex capabilities such as multithreading, opportunistic concurrency, load balancing, connection pooling, partitioning, recoverability, failure isolation and administration are introduced by the Service Broker without any implmentation cost or complexity in the business logic.

Let's explore a practical example of accessing mainframe data from Java with several different approaches, including the Service Broker approach.

## Comparison of Different Mainframe Data Access Solutions

One of the most significant data access challenges facing e-business application developers today is accessing mainframe data. Access to mainframe data can create significant business opportunities, but only if a sufficient business case can be made.

- **Lower life-cycle costs** – You need to retain mainframe business logic while moving new e-business application load off the mainframe to meet growing demand.
- **Lower operational costs** – Equipment support and staff training costs of terminal user interfaces (i.e., green screens) isn't competitive with Web-based browsers.
- **Increase revenue** – You can deploy e-business services that directly increase your market share at the expense of your competitors.

There seems to be a befuddling array of choices when it comes to which solution to use to access mainframe assets. However, your requirements are timeless as they exist for any system deployed for a business revenue-critical enterprise environment. These requirements are reliability, performance, scalability, security, legacy support, multiplatform support, management and low development and maintenance costs. There are a multitude of solutions for integrating mainframe assets, most of which can be identified as being in one of three general categories:

- Terminal emulation middleware
- Indirect access gateway middleware
- Direct access Service Broker middleware

IBM's Information Management System (IMS) provides On-Line Transaction Processing (OLTP) for much of the world's critical business and commercial service applications. In existence for over 25 years, only CICS surpasses IMS in managing the world's business-critical transactional data. IMS is an ideal testbed for comparing the capabilities of terminal emulation, gateway middleware and Service Broker middleware.

The business logic for the IMS testbed consists of a common Java application that invokes each different middleware solution to extract 100 rows of IMS data. The Java application and middleware is hosted on a Solaris 2.6 Ultrasparc platform (e-business server). The IMS application and data are hosted on an IBM OS/390 mainframe. The Java code differs only in the call syntax required by the specific middleware interface. JDK 1.1.6 with the JIT enabled was used for all configurations.

The performance data shown in Table 1 consists of the average of a sample size of five measurement runs. Column 2 (Process Launch), is the amount of time needed to launch and initialize the business logic process. Column 3 (Connect) is the amount of time needed to establish a connection to IMS. Column 4 (Execute) is the amount of time needed to execute the transaction through the middleware and Column 5 (Extract) is the amount of time needed to extract the IMS data from the data block returned by the transaction query. Column 6 (Total Time) is the total clock time needed to execute all aspects of the Java application and middleware on the e-business server. The individual times of Process Launch, Connect, Execute and Extract account for over 90% of the Total Time. Column 7 is the total number of bytes transferred from the OS/390 mainframe tier to the e-business server. Column 8 is the amount of OS/390 processor CPU consumed to execute the transaction.

If your mainframe applications are mature and you just want to overhaul the interface, not add any new business logic, then you may want to consider using terminal emulation middleware. The terminal emulation (or *screen scraper*) approach doesn't require any installation of new software on the mainframe. However, terminal emulation has distinct disabilities when used as e-business middleware. The mainframe host must manage a virtual terminal session as well as add terminal management control code data in the data stream for each transaction. All this terminal emu-

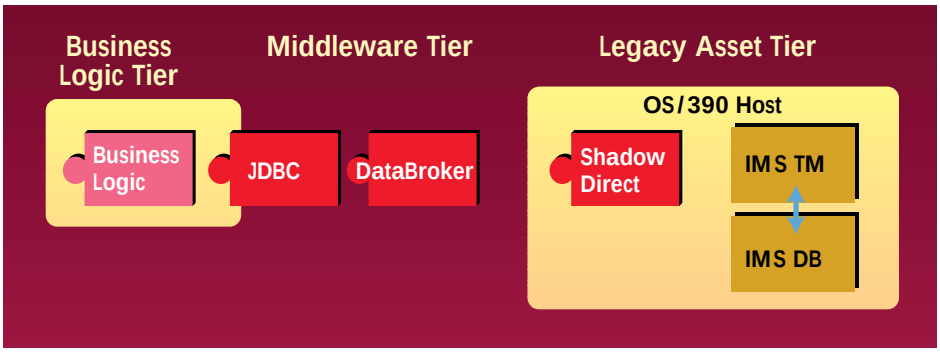



*Figure 3: The Service Broker three-tier server architecture for IMS.*

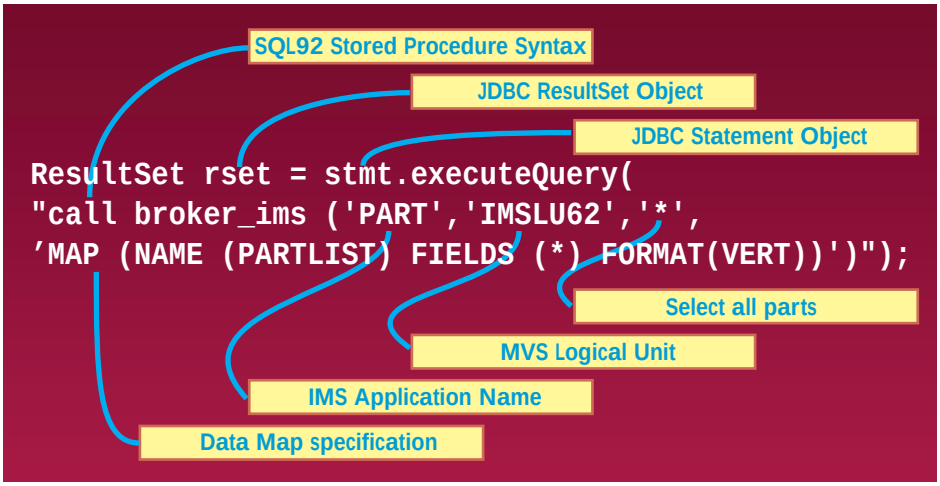



*Figure 4: Service Broker JDBC syntax for accessing IMS data using DataBroker and Shadow Direct for IMS*

lation processing and control is wasteful as it goes unused by the mid-tier business logic.

For the IMS testbed results given in Table 1, the terminal emulation middleware consumed 85 msec – 280% more than the Gateway and 400% more than the Service Broker solution. In terms of capacity planning, terminal emulation will need 300% or more equivalent MIPS to service the same user load as a Service Broker solution. Additionally, any 3270 screen format change will require code maintenance of each affected screen-scraper script. You can expect high maintenance costs, low performance and increased load on the mainframe with a terminal emulation solution. At a current cost of $10,000 per mainframe MIP, terminal emulation is impractical for the high concurrent user loads sustained by e-business systems.

The Gateway middleware for accessing IMS was the IBM DB2 Connect ODBC driver on the Java platform connecting to DB2 on the mainframe. DB2 acts as a gateway by enabling a DB2 procedure to access IMS. Two different Java platform configurations were used:

1. The Java application invoked a CGI/Perl script which in turned invoked the ODBC driver. The Perl script "extracted" the returned IMS data from the ODBC driver as an array of strings. The array wasn't passed back to the Java application and if it had, it would have resulted in a larger Extract time.
2. The Java application used the Javasoft JDBC/ODBC bridge. The JDBC/ODBC bridge was invoked directly and data returned directly to the Java application using the same extraction logic as the Terminal Emulation and Service Broker middleware configuration.

The performance of the CGI/Gateway configuration is at a large disadvantage because of the process launch time of 200 msec. However, most of the process launch time can be almost completely eliminated by using FastCGI, which keeps the CGI process in memory.

The JDBC/ODBC Connect and Extract performance was better than CGI/Perl for Gateway middleware. In this case, Java string manipulation and foreign function call facilities matched Perl. Perl's string manipulation facilities are famous for being able to match or sometimes exceed C/C++ code performance. Both were equal in their use of mainframe resources as they both shared the ODBC/DB2 gateway to IMS. The developer productivity advantages of Common Gateway Interface (CGI) using Perl and other scripting languages has been matched and exceeded by the developer productivity found with Java servlets. JDK 1.2 performance improvements in string manipulation and JIT technology will further increase Java development and runtime advantages. Given the performance advantages of Java servlets over CGI, it becomes easy to understand why server-side Java is so quickly replacing CGI as a business logic platform.

The Service Broker configuration shown in Figure 3 consists of the I-Kinetics DataBroker and the Neon Systems Shadow Direct for IMS. DataBroker features a complete JDBC driver. The DataBroker is the Service Broker that directly interfaces to Shadow Direct – an installed service running under OS/390 that integrates directly with the IMS Transaction Manager.

| Middleware Solution | Process Launch (msec) | Connect (msec) | Execute (msec) | Extract (msec) | Total Time (msec) | Total Data (Bytes) | OS/390 CPU (msec) |
|---|---|---|---|---|---|---|---|
| Terminal Emulation | 21 | 500 | – | 410 | 940 | 11,700 | 85 |
| CGI/Gateway | 200 | 350 | 190 | 120 | 870 | 7,600 | 30 |
| JDBC/ODBC/Gateway | – | 310 | 200 | 75 | 600 | 7,600 | 30 |
| Service Broker | – | 12 | 23 | 39 | 81 | 3,500 | 21 |

*Table 1: Performance results for different middleware access to 100 rows of IMS data*


The first immediate performance advantage realized with the Service Broker approach is the Connect time of 12 ms for the IMS testbed. The DataBroker automatically organizes all database connections into a connection pool. When a requested database connection is already available in a connection pool, it will be reused.

For e-business transactions with small Execute and Extract costs, eliminating fixed overhead Process Launch and Connect costs are essential to achieving high performance. For Insert/Update/Delete transactions with an Execute time of 10 ms or less, a Connect time of 300 ms will heavily tax the peak transactional throughput. Reducing Connect time with connection pooling can achieve significant increases in performance and scalability.

Other performance advantages realized with the Service Broker were with Execute and Extract performance. Because DataBroker is directly integrated with Shadow Direct and Shadow Direct directly accesses the IMS transaction manager data queues, the resulting Execute performance is very high. Additionally, with Shadow Direct compressing the data stream the amount of data transmitted (3,500 bytes) is dramatically smaller than with Terminal Emulation (11,700 bytes) and Gateway (7,600 bytes) middleware configurations.

The Java code for accessing IMS using the Service Broker is shown in Figure 4. IMS applications are accessed using a SQL-92-compliant stored procedure call, broker_ims, invoked using the JDBC statement interface method executeQuery. The broker_ims stored procedure call includes parameters that specify the IMS application to execute and result set data formatting. From this example, it is easy to grasp the power of using Java service interfaces, such as JDBC, to access legacy assets, such as IMS.

## The Right Way: Transforming Legacy Assets into Java Services

The Service Broker architecture provides a path for the enterprise to scale up existing client/server applications to high-performance, high-capacity multitier e-business applications. The enterprise is protected against being locked into a single platform or vendor middleware solution. The enterprise IT infrastructure decision maker can depend on Java services, such as JDBC and JTS, to be essential standard elements of the Java and Enterprise JavaBean platform. The Service Broker provides a complete migration from current IT systems to future IT systems, overcoming the drawbacks of other integration strategies.

An important part of the Java service legacy integration strategy is for the software industry to create Java Service Brokers for the world's $5 trillion of IT assets. You can expect the Java software vendor community to supply Service Brokers for a wide range of legacy assets. Currently, you can access almost any mainframe asset, such as ADABAS, CICS, DB2, IMS, IDMS and VSAM, as a JDBC data source with the I-Kinetics DataBroker. IBM has announced the access of their popular messaging middleware, MQSeries, as a JMS service. Of course, the leading transaction processing vendors, such as BEA and IBM, will be offering JTS for their transaction monitors.

A good Service Broker will offer high-performance and scalability features to complement a high-capacity Enterprise Java-Bean platform. Building your own Service Broker is a difficult and expensive experience. Seek out and research existing Service Brokers for your current legacy integration needs. With Plug and Play Service Brokers delivering JDBC, JTS and JMS services, you can focus on the critical tasks of developing new e-business applications.

## References and Resources:


- "Solving the Data Inter-Operability Problem Using a Universal Data Access Broker," Mike Higgs & Bruce Cottman. IEEE Data Engineering Bulletin, September 1998.
- "Introducing DataBroker: Standards Based Data Access for the Enterprise," Mike Higgs, Netscape ViewSource 1997.
- "DataBroker Version 6: A Technical White Paper," I-Kinetics, September 1998.
- "Universal Data Access," Bruce Cottman. ***Java Developer's Journal***, April, 1997.
- *The Essential Distributed Objects Survival Guide*, Robert Orfali, Dan Harkey and Jeri Edwards. John Wiley & Sons, Inc., 1996.
- *Client/Server Programming with JAVA and CORBA*, 2nd ed. Robert Orfali and Dan Harkey. John Wiley & Sons, Inc., 1998.



**About the Author**

*Bruce H. Cottman is the president of I-Kinetics, Inc. Since founding I-Kinetics in 1991, Bruce has focused on solutions for transforming legacy into high-performance component software for large distributed systems. He holds S.B. and S.M. degrees in physics from the Massachusetts Institute of Technology and a Ph.D. in physics from Rensselaer Polytechnic Institute. Bruce can be reached at bruce.cottman@i-kinetics.com, or at his Web site, www.i-kinetics.com.*

bruce.cottman@i-kinetics.com

# CORBA vs Servlets: What to Use Where

*Getting started in distributed object computing*

*by* **Rachel Gollub**

Distributed object computing in Java has become increasingly popular as more complex products are written using a multitier architecture. A number of products and protocols are available for facilitating communication, and many developers have trouble deciding which ones to use in a given situation. Many of the communication methods work well together, and each has its strengths and weaknesses. In this article I'll discuss two of the most popular methods, CORBA and servlets. Both are useful for distributed computing, and they complement each other well.

## What Is Distributed Object Computing?

Distributed object computing is development involving communication between two or more independent modules of an application. Usually, it involves communication between a client and a server – for example, a Java time-tracking application. This application would have a client side written as a Java applet and a server side that stores and retrieves information from a database. Users could enter the hours they spent on a specific project on the client side, and that information would be transferred to the server side and stored in the database. Likewise, if they wanted to retrieve previously stored hours, their request would be passed to the database and the information would then be passed back to the client side, generating a report dynamically.

This application is a good example of a multitier architecture. The client communicates with the server, and the server communicates with the database, creating three different layers to the application as well as a need for a communication protocol between each set. Since they have only the two modules, client/server applications are a special case of multitier applications. Communication between the server and the database can be done directly (using JDBC or a similar API), since the database is usually on the same intranet as the server module. However, the client and server modules not only could be on different intranets, they could also be separated by a firewall or another divider. Therefore, communication between them can be difficult. In order to communicate, the application must assemble information on one side, encapsulate it and send it to the other side, which must retrieve and decipher it. This is the problem distributed computing protocols and products are designed to solve.

## What Is CORBA?

CORBA stands for Common Object Request Broker Architecture. The Object Management Group (OMG) is the body responsible for creating and maintaining the CORBA specifications. CORBA includes specifications for a language to define objects, a protocol for exchanging information and a protocol for passing objects over the Internet. Several companies have taken these specifications and developed implementations of CORBA for Java. Inprise VisiBroker is the leading commercial implementation, and Sun recently released Java IDL as a core API as part of the Java 2 platform release. There are also implementations available for different languages and platforms.

The language CORBA uses to define objects is the Interface Definition Language, or IDL. Once an object is defined using IDL, it can be translated to a specific language such as C++ or Java. This object can then be used by both the client and server modules as a native object. The object is passed between the client and the server using the Internet Inter-ORB Protocol (IIOP), a protocol for translating the General Inter-ORB Protocol (GIOP) for use with TCP/IP. In other words, IIOP is the protocol used for transferring these CORBA objects from one side to the other over the Internet.

The Object Request Broker (ORB) is the active part of a CORBA implementation. It receives a request for information from one module of the application, finds an implementation of the object associated with that information and handles the transfer between modules. For example, in the time-tracking application above, the applet code might call a method that is actually implemented in the server. When the method is called, the ORB is given the request, passes any parameters of the method to the server side, executes the method and sends the return value back to the client. This is all transparent to the client code, so that writing a client application using CORBA is not significantly different from writing a normal Java application. The CORBA specification includes ORB interoperability, guaranteeing that an object passed to one vendor's ORB can be understood by any ORB written by any other vendor. A simple example of the client server and IDL code for an application can be seen in Listing 1.

CORBA is an excellent protocol to use when the goal is to pass information transparently between the client and server. Many of the implementations exhibit high performance and are easy to use and incorporate into existing applications. The guarantee of ORB interoperability makes it easy to switch implementations and add code or JavaBeans from other sources to existing CORBA-compliant code.

## What Are Servlets?

Servlets are Java modules that can be executed by a Web server. They are similar to CGI scripts, but have several significant advantages. First of all, they are run within the same process as the Web server, while CGI scripts are executed in separate processes. This makes servlets faster and more efficient than scripts. Second, servlets are written in Java, so they are immediately portable to multiple platforms and have the range and flexibility of the full Java language. This means, for example, that a servlet could also use the JDBC API to access a database, or even use CORBA to access a different server. CGI scripts are generally written in platform-specific languages or compilations of languages, making them hard to support and less flexible. Finally, the Java Servlet API allows easy access to a full range of information about the request, even allowing objects to pass

between the client and server side. The most common uses of servlets are to drive dynamic Web sites and to produce dynamic HTML within a Web-based application.

Servlet objects are usually passed using the Hypertext Transfer Protocol (HTTP). HTTP, the most common Internet transfer protocol, is supported by all Web browsers and servers. It's specifically designed for transferring Hypertext content over the Internet, and is specialized for doing so. This means that while servlets are extremely efficient for producing and transferring dynamic HTML, they are less efficient for transferring objects between an applet and a server. For comparison, they are faster for object transfer than some implementations of CORBA, but slower than most.

Servlet implementation depends mainly on the Web server chosen to incorporate them. Sun has released a product, the Java Web Server, that natively supports servlets. A number of products, including one from Sun, add servlet support to existing Web servers, so most common ones can now support servlets. A complete list of supported servers is available from the JavaSoft home page. Once you've written a servlet and compiled it using the JDK and the Java Servlet API (or the Java Servlet Development Kit), you put the class file into a specific servlet directory in the Web server hierarchy and restart the server. At that point the servlet is ready for use. A simple example of a servlet can be found in Listing 2.

## Which Should I Use?

Servlets and CORBA both have their strengths and weaknesses, and which is appropriate depends on the situation. CORBA is optimized for transferring objects transparently, while servlets are designed more for server-side processing resulting in dynamic HTML.

In the time-tracking application listed above, there are two functions that happen with reasonable frequency: storing information to the database and generating reports. In the first case, the application should pass data from the client to the server and store it in the database. This is simple object passing from an applet to a server application, which CORBA does very efficiently. The same process could be done with servlets, but it would require more coding and maintenance to work efficiently, and would generally have poorer performance. The second case is more complicated – are the reports in HTML or in Java? If HTML, then the application is passing a request to the Web server and expecting dynamic HTML in return, something that servlets do most effectively. Again, the same process could be done using CORBA (pass the request to the back end using CORBA and use that to write an HTML file that can then be displayed or downloaded by the client), but in this case CORBA requires more coding and more maintenance, and has poorer performance. However, if the report front end is in Java, CORBA again becomes a better answer.

In the case of a Java front end and HTML reporting, new questions come up. Each method is useful in its area, but there's a lot of overhead involved in implementing two different communication methods in one application, particularly a small one. Some other considerations are budget and memory requirements. Running a client-side applet, an ORB, a server–side application, a Web server and possibly a servlet plug-in can be expensive and memory intensive enough to demand a decision between the two methods. Until recently, servlets tend-

> ***"CORBA is an excellent protocol to use when the goal is to pass information transparently between the client and the server."***

ed to be the cheaper alternative, since kits for running servlets with the major Web servers can be downloaded free. The introduction of Java IDL, however, has generated a CORBA option that is also distributed free, so either one is now a reasonable choice on a small budget. In this case, the best way to make a decision, if you can use only one method, is to plan to optimize the application for one method or the other.

One possibility is to change the front end for the application entirely to HTML. Most time-tracking applications have reasonably simple front ends and could be easily converted to HTML. This would eliminate the need for CORBA, since all pages are now dynamic HTML and will be most efficiently served by servlets. Conversely, you might decide to use a different reporting tool that uses a Java front end, eliminating the need for any HTML, which then allows you to use CORBA most effectively. Finally, you might decide to leave both the way they are and do some performance testing to see whether it is more efficient to use CORBA for your HTML or servlets for your applets, since these performance measurements are very much dependent on CORBA implementation and code specifics. Since each method can be used to duplicate the other one, once you have chosen the one with the minimum loss of performance, you can standardize on that method.

Larger applications, on the other hand, benefit greatly from using more than one method of communication. Performance and scalability are critical issues for large applications, and using more than one method can assure that the application is optimized for maximum performance. CORBA and servlets don't interfere with each other at all in an application, and can be used side by side in the same code without any problems. For example, a Web site management application may let the user save dynamic HTML to a database and then view it. CORBA could be used to send the information to the database and a servlet could then be invoked to retrieve the information, do any dynamic substitution necessary and display the final dynamic Web page. Using both can be a distinct advantage over having to optimize an application for only one method.

## Where Can I Get More Information?

More information on CORBA, including CORBA, IDL and IIOP specifications, is available at www.omg.org/, the OMG home page. There is also a CORBA home page at www.corba.org/ and JavaSoft's Java IDL is available for download from the Products and APIs section of the Java home page, java.sun.com/. Information on servlets, the Java Web Server and the Java Servlet API is available from the servlet home page at jeeves.javasoft.com/. This page includes a list of products available from other vendors to support servlets on a number of servers.

**CODE LISTING**

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


**About the Author**

*Rachel Gollub is a senior engineer/architect at Imparto Software Corporation. She started programming in Java (then Oak) in 1995, and was hired by JavaSoft, where she worked until 1997. She contributed some of the demo applets shipped with the JDK, and made contributions to the Java Web Server and the security classes. Rachel can be reached at rachel@imparto.com.*

rachel@imparto.com

# 'Is the End Nigh?'

## *It better be – we've paid good money for it*

*by* **Alan Williamson**

Welcome to your monthly dose of controversy – the part of the magazine where I ask you to push back the keyboard, stop debugging that Java class that has been bugging you for the past couple of hours and get your shot of caffeine as I invite you to take a look at this crazy Java universe through the eyes of a British company.

It's been an interesting month for us here in Scotland, so I have a lot to tell you. As usual, we'll contrast our findings to a personality trait. This month I think we'll look at *gullibility*. That's one that all of us at some point in our lives have been subjected to. Whether we *admit* it or not is a trait for another month, so let's not get ahead of ourselves.

What I'm about to say may come as a bit of a shock to some of you. It appears that something rather exciting is about to happen. We're not too sure what it is, but I am reliably informed by the BBC and CNN that it will be major. What am I talking about? The end of the world, of course. The end of civilization as we know it. The millennium is fast approaching and we're down to days as opposed to years.

Since every other journalist under the sun has written about it, I thought it was about time I threw my two cents in about the whole thing. As with any good Clinton or Michael Jackson story, the millennium bug is full of complete misinformation. The only consistent fact is the name of the problem: Y2K. The computing industry, true to form, has assigned yet another acronym to the list of thousands we already manage. No wonder public perception is of doom and gloom if we shroud the problem in our own tech-speak!

### But There's Always Hope

I'd seen so many contrasting and scaremongering reports that I'd given up hope on the world as a whole. Had the whole place gone mad? Reports of planes dropping from the sky, buildings falling over, governments crashing and wars taking place were all going to be attributed to a simple date problem. Fortunately, my sanity was restored when I read a very informative and realistic article in the *New Scientist Journal* that profiled what the world is really going to be like when we wake on January 1, 2000. And here's where I'll let you all into a wee secret: the sun will still be shining, the earth will still be revolving...and Jerry Springer will still be broadcasting!

Don't get me wrong – I know there will be some problems we'll have to deal with, but on the whole I believe that the majority of the world's population will indeed be very disappointed with the turn of the year 2000. It's not going to be the big bang we're all expecting. I think the biggest shock will be to the companies that – for reasons best known to themselves – hired the cowboy consultants who are charging £20k a day for the 31/Dec/1999 to 1/Jan/2000 period, only to discover that the problems still exist. What's confusing me is just what these people expect to achieve.

For example, let's assume that they're Y2K experts. Let's further assume a problem does occur. Do they expect to fix it then and there? Because if they haven't been able to fix it beforehand, what are the chances of their achieving success on the given night? Personally speaking, I feel that this sort of behavior is taking advantage of computer-illiterate companies. Some of you will say they deserve it, or even So what? Well, there's that attitude, but that really doesn't do the industry a whole lot of good.

We all know the basic foundations for the Y2K problem: the saving of disk space when dealing with dates, and the year precision. Back in the 1960s it was assumed that their software would not still be running in the year 2000. Which, I suppose, you can't blame them for. If I were to ask if you thought the class you just wrote would still be running in 30 or 40 years' time, how many of you would say yes?

Not that many, I suspect. Why? Well, we probably think development languages will have moved on so much by then that there will be no place for a lonely Java class to exist, let alone still provide a useful function. This is another great debating point, and one that I welcome all e-mail on.

(Speaking of debating points, let me thank you all for the response to my article [***JDJ*** Vol. 3, Issue 12] on company loyalty – I think I touched a nerve there. I had put forward the notion that employees aren't as loyal to companies as they were once. This sparked a flurry of e-mail from readers accusing me of not seeing the "two-way street" of loyalty. Of course I see the two-way street, where the company also shows loyalty to the employee. There are many stories of people who have given the majority of their working lives to a company only to be dumped when the going got tough. I appreciate this, and as I said in my article, it's the bigger companies that have made it sour for us smaller companies. I enjoyed debating the issues with you. So keep the e-mails coming!)

Back to our Y2K issue. Assuming our Java classes will see it to the next millennium and even past it to the next one, in some museum of computing through the ages will they cope with the change? Well, Sun has assured us that Java is indeed Y2K compliant – assuming you've been a good developer and stuck with the standard date classes within the JDK. If you haven't, you're on your own, which is fair enough.

### Just for Fun...

Let's assume that there was a problem with the fundamental date classes that meant a wee problem on Hogmanay (New Year's Eve) 1999. How serious would this be to fix, compared with the problems facing the Y2K developer now? Well, assuming a problem existed in the java.util.Date class, for example, all that would be required would be for Sun to fix it and reissue the class file or virtual machine to the world at large. All we'd have to do is copy over the new class file and restart. That would be it. No recompiling required on our behalf. This would be the miracle cure that would require nothing more than a simple copy. That's the power of Java.

But this ease of upgrading or bug fixing can be applied to anything in the Java universe. That's one of the advantages of having a good class design and a good solid object structure. The ability to swap a bad module for a good one without disrupting the rest of the system is indeed a powerful one. For this reason, I believe that the Y2K problem is indeed a special time.

It'll probably be the last time we see a major problem on this sort of scale. If anything like this pops up in the future, we can take refuge in the fact that for us to fix a


small fundamental component in the system won't require a lifestyle change.

Of course, this is assuming that we're still around to enjoy the turn of the century. While I was researching the material for this article, I stumbled across the origins of dates and calendars. I also came across some of the more doom and gloom predictions about this time of the century and I'll share some of my findings with you.

For a start, we can attribute the basic calendar to the early Egyptians. At one time they were managing up to three calendars at once, since each one was failing at given times of the year. The basic numbering of our years came from the biblical event of the birth of Jesus Christ. Hence the "BC" notation for describing years before the year 0. But I discovered a number of sources that point to the fact that Jesus may have been born closer to 75 AD, and around October rather than December, which is when we traditionally celebrate it. They can ascertain this by trying to figure out when the bright star – which supposedly guided Mary and Joseph to the stable – would be that high and that bright.

It's always amazed me that there was once a time when someone said; "Okay, this is now year 0, agreed?" Can you imagine being that decision maker? Talk about responsibility! So the fact that we're coming up to the year 2000 is more of a numerical fluke than some religious destiny.

Another man's name that's popping up frequently at this time of the year is that of our old French friend Nostradamus. If you've never heard the name, or even seen the movie, Nostradamus was a sixteenth-century prophet who seemingly predicted the world wars and a number of other notable events throughout history. The problem is that Nostradamus wrote his prophecies with much of the same vagueness as the Bible – and look at how many interpretations THAT document has had! But one of his most shocking predictions is coming up, apparently this year.

Nostradamus has predicted that the King of Terror will be coming in the seventh month of 1999 and will either destroy or significantly damage the world as we know it. Again, this is all so vague. Some believe it will be a meteor, while others maintain it's a nuclear war. The translation of the original documents is also open to debate, as the seventh month can mean either July or September. To join in or listen to the debates, join the alt.prophecies.nostradamus newsgroup at www.liquidinformation under a Nostradamus search.

What can we learn from this? Well, I wouldn't go signing that huge loan agreement just yet on the basis that the world might not be available for you to start making the repayments. But I wouldn't completely dismiss it. Let me tell you a wee story that scared the living daylights out of us. We've been experiencing some seriously bad storms here in Scotland. I was taking some time out from the keyboard, and we started chattering about Nostradamus and his predictions. I looked out, noticed the trees bending at an unbelievable angle and made the comment, "You know with all these storms, maybe this is the beginning of Armageddon." As soon as I had the last word out, the complete power for the whole village went off. We were standing in darkness, and we all just looked at each other in disbelief at the potential timing of those fateful words. Spooky!

On that note, I think we'd better wrap this article up before I encourage any further bad karma. Before I close, let me give you this month's recommended reading.

## Recommended Reading

The books I love reading are the ones that tell stories of the rise – and sometimes fall – of large corporations. I was given a book, *Insanely Great*, penned by Steven Levy, that charts the rise of Apple computers and how Apple relied on the very fact of being Apple in the early days. He talks about how great it is to own an Apple, and there's a bit in the book where Levy admits that being an Apple devotee isn't without its costs. We all know Apple devotees – people who, for reasons non-Mac users can't fathom, believe that the only real computer is an Apple. According to Levy, it's because there's safety in numbers. In the early days the marketing of Apple sold little more than the reality. Therefore, for the buyer not to feel isolated he had to maintain the momentum of the Apple marketing people and gloss over the problems that hindered many of the early Apple machines to his colleagues, friends and family. It was like being a member of an elite club, where the golden rule was never to mention the failings. You loved your Apple, worms and all. It's an entertaining book, and you'll chortle at some of the information.

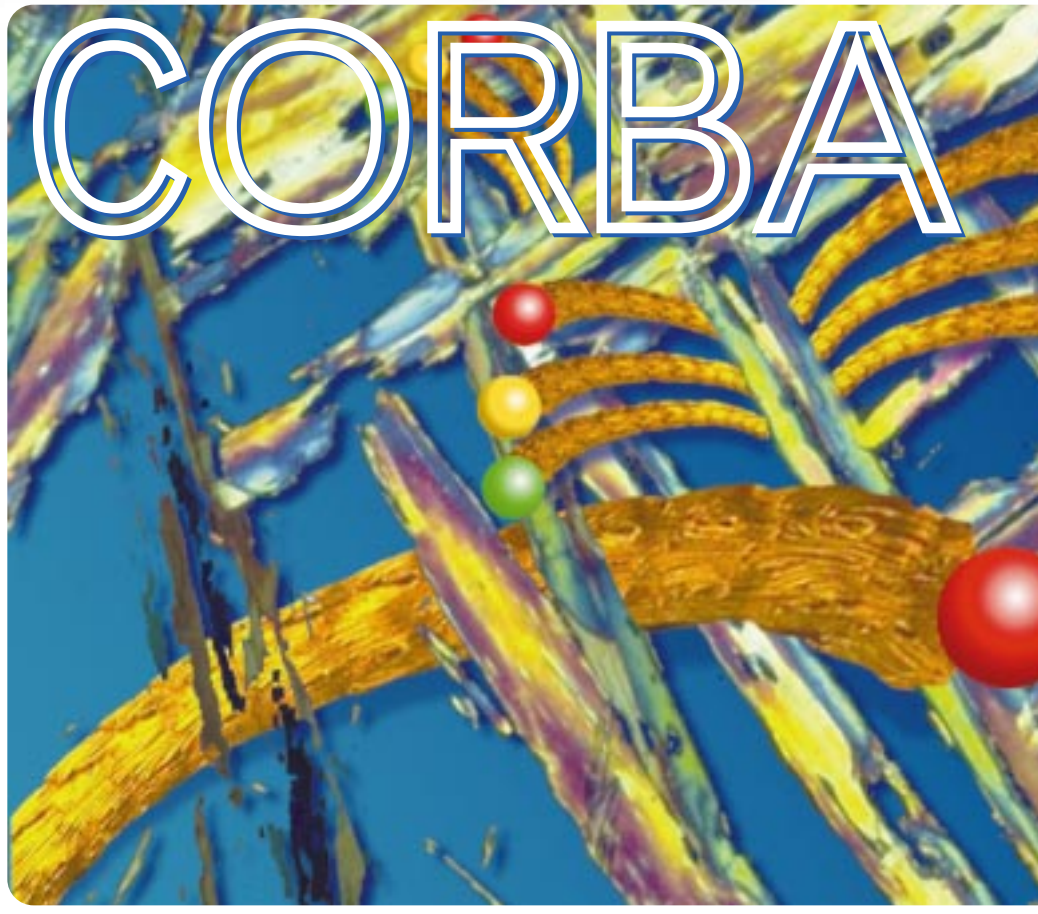
Now I'm going to prepare my nuclear bunker in preparation for the end of the world. If it hasn't happened by the time next month comes round, I'll be back!

**About the Author**

*Alan Williamson is CEO of n-ary Ltd., a Java consulting firm 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 has contributed to the 2.1 Servlet API. He can be reached at alan@n-ary.com (www.n-ary.com).*

alan@n-ary.com

# Callbacks IN CORBA



## *Using a traffic light controller can help guide your callbacks*

*by* **P. G. Sarang**

In recent days, CORBA has fast become a standard for the development of distributed applications. A CORBA application may consist of one or more CORBA server objects and many clients who connect to these servers. A CORBA server object makes itself available to the client by registering with the CORBA Naming Service or a CORBA Trader Service. A client locates the desired server object on the network by using this Naming or Trader service. Once the server object is located, the client receives a reference to it. Using this reference, the client can invoke methods on the server object and carry out its desired work. Thus it's the client that usually makes use of server facilities; the server simply returns the results of method invocations to the client. However, in some situations, it may be necessary for the server to invoke a method on the client object. For example, the server may like to notify the client of the occurrence of a certain event on the server or the completion of a processing job requested by the client. This method of invocation on the client is called Callback. CORBA specifications allow Callbacks on clients. This article discusses the implementation of CORBA callbacks using Java.

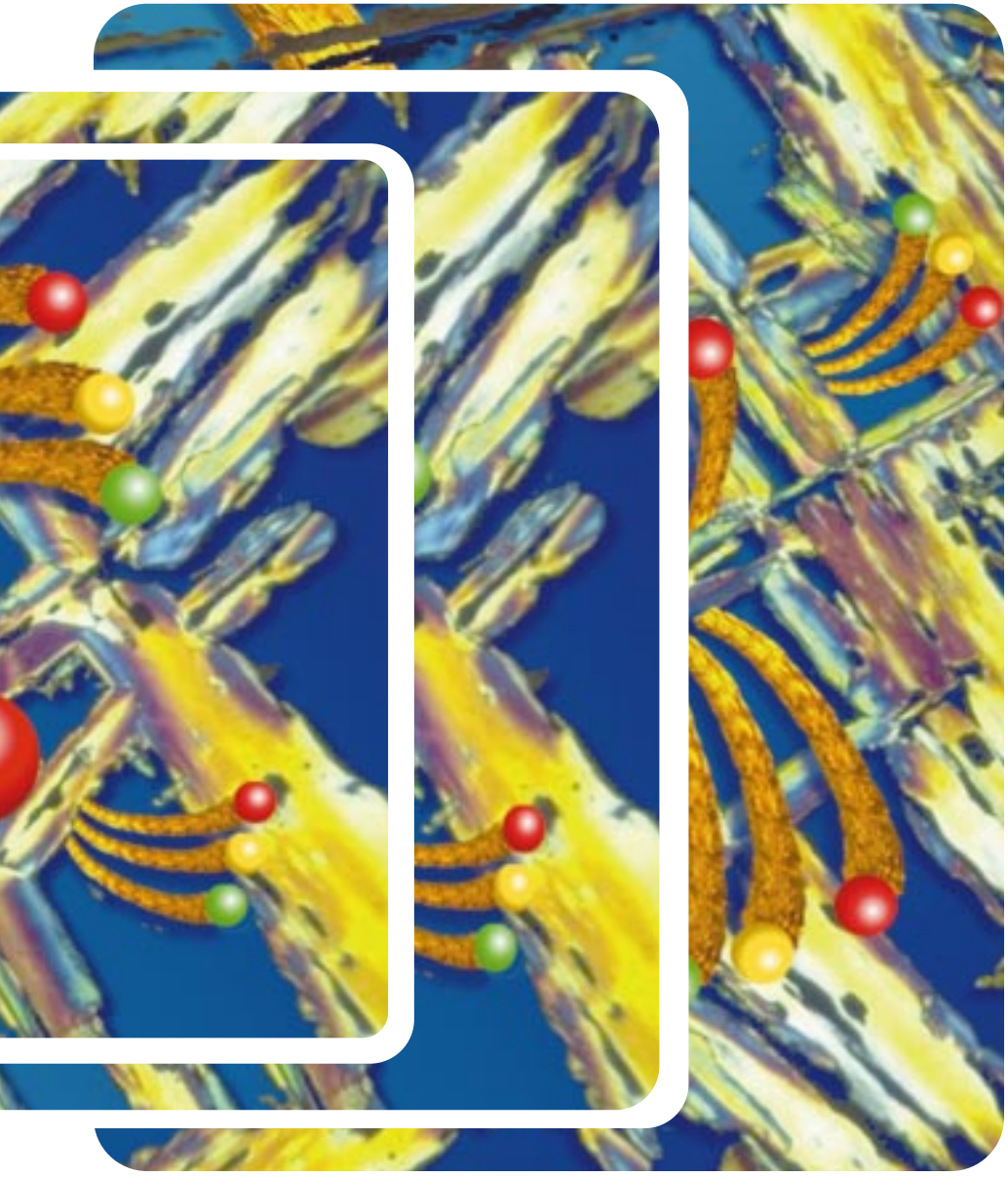
CORBA specification is language-neutral and thus both client and server programs can be implemented in different languages as selected by the programmer. To implement Callbacks, a client must pass a reference of itself to the server. The server is responsible for storing such references to different client objects and calling methods on the appropriate client as and when required. The server must store these references in a language-neutral format. Thus a proper CORBA data type should be used for storing such references. This article uses the example of a traffic light controller to discuss the implementation of CORBA Callbacks using Java. Java has been used as the implementation language for both client and server applications; however, any other language of choice may be used for the implementation.

The traffic light controller application design consists of a traffic light controller object and several traffic light objects (see Figure 1).

Each traffic light object, when created, registers itself with the controller. The controller stores the reference to each registered object. The controller object acts as a CORBA server that registers itself with the CORBA Naming Service. A traffic cop object that wishes to control the lights at a particular junction locates the controller object

by looking up the CORBA Naming Service for the desired name. Once the controller is located, it'll retrieve the number of lights that the controller is currently managing. The user interface of the cop application shown in Figure 2 displays the panel for each registered traffic light.

Using these panels, the cop sets the states (colors) of the various lights and requests appropriate changes to the controller. The controller, in turn, scans through the list of registered lights and requests the lights to set the desired state (color) by calling a method on each of the light objects.



## CORBA IDL

The development of a CORBA application begins by writing Interface Definition Language (IDL) code. The CORBA IDL for the traffic lights application is shown in Listing 1.

The module traffic defines two interfaces – TrafficLight and TrafficCoordinator. The TrafficLight interface defines CORBA traffic light objects while the TrafficCoordinator interface defines the CORBA coordinator interface.

The TrafficLight interface contains a short data type that stores the light number. It also provides a method called SetColor(), which is called by the coordinator to set the state of the traffic light. This is a Callback method, which is invoked by a CORBA server object.

The coordinator interface declares a CORBA sequence variable to hold references to registered TrafficLight objects. The sequence is implemented in Java using a Java array or a Java Vector class. The coordinator declares a method called Register(), which is called by the TrafficLight object to register itself with the coordinator. The coordinator declares another method called SetColor(), which is called by the traffic cop application. The SetColor() method receives two parameters – the traffic light number and the color for the light. The coordinator uses this traffic light number to locate the desired traffic light object and sets the color to the desired one by calling SetColor() method on the object.

## Mapping IDL to Java

I used Visigenic VBroker for Java for the development of the application. The first step in developing the application is to map the IDL code to Java. The following command line will do the mapping. Note that Visigenic provides the idl2java utility.

```
idl2java -no_tie -no_comments traffic.idl
```

The –no_tie option tells the compiler not to generate the tie classes and the –no_comments option disables the comments generation. The compiler creates a Java package with the same name as the CORBA module – traffic. The two CORBA interfaces are mapped to two Java interfaces – TrafficLight and TrafficCoordinator. The idl2java compiler also creates the implementation classes and the example classes for the two interfaces. The files _example_TrafficLight.java and _example_TrafficCoordinator.java can be used to provide the implementation of the two CORBA interfaces. Maintain a backup of the original files to ensure that your implementation code isn't overwritten the next time you run idl2java compiler. Copy the _example_TrafficLight.java to TrafficLightImpl.java and _example_TrafficCoordinator.java to the TrafficCoordinator.java file. You'll write your implementation code in the newly copied files.

## Developing the Application

The source code for TrafficLightImpl.java is given in Listing 2. The class declares a variable number of short data types for holding the light number. It declares one more variable of Java class type as TrafficLightServer. The source for the TrafficLightServer class is given in Listing 3 and is discussed later. The class TrafficLightServer creates the object of TrafficLightImpl class. The SetColor() method of the implementation class (TrafficLightImpl) uses this reference to call the SetColor() method on the TrafficLightServer object. The constructor for the TrafficLightImpl class receives two parameters – the reference to TrafficLightServer object, which called this constructor, and the light number. The two parameters are saved into the two class variables discussed above. The constructor prints an appropriate message to the user upon successful creation of the object. The accessor and modifier methods for the Light-

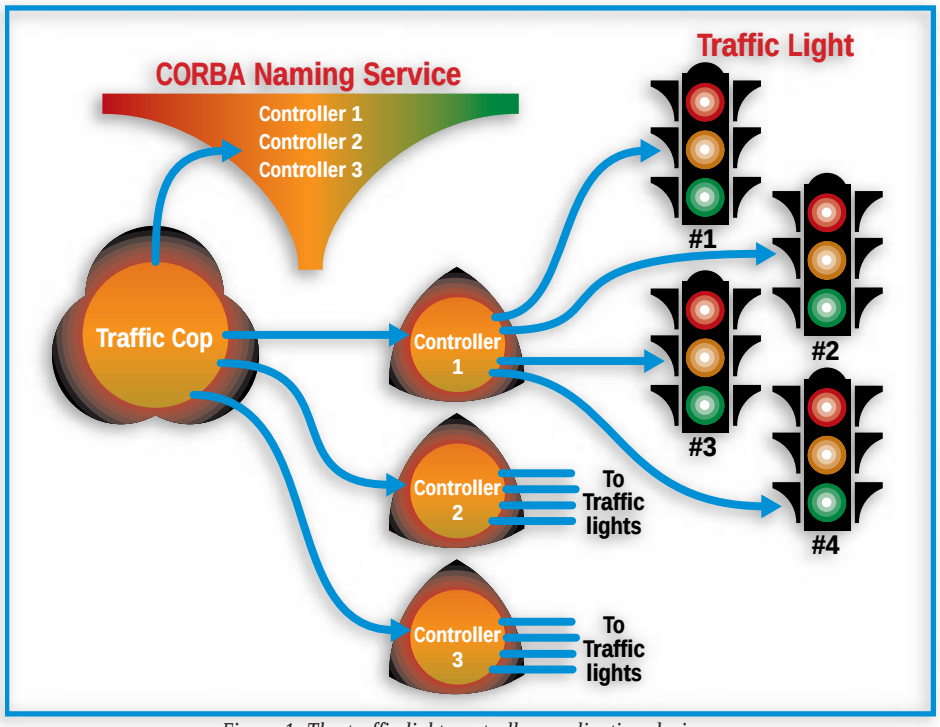



*Figure 1: The traffic light controller application design*

Number attribute, respectively, retrieve and save the value to the local class variable – Number. The SetColor() method of TrafficLightImpl class simply calls the SetColor() method of the TrafficLightServer class.

Now look at the implementation of the TrafficLightServer class given in Listing 3. This class is derived from the Frame class and is a Java command line program. The program receives one command line argument that specifies the number for the traffic light to be created. The init() method sets the location of the frame, depending on the light number. For simplicity, only four traffic light objects are considered. The init() method then builds the user interface for the traffic light object by creating three objects of the TLightPanel class. The object hierarchy for the user interface of TrafficLightServer is shown in Figure 3.

The TLightPanel class is derived from Panel class and creates a traffic light in the panel painted in a given color.

After creating the three TLightPanel objects and adding them to the container, the init() method of the TrafficLightServer class sets the listener for window events, and sets the default light color to red by calling the SetRedLight() method. Once the user interface is constructed, the main() method of the TrafficLightServer class resolves the reference to the CORBA Naming Service and locates the coordinator object. It registers the newly created traffic light object with the coordinator by calling its Register() method. The program then waits for invocations to be made by the controller.

The TrafficCoordinatorImpl class provides the implementation for the CORBA TrafficCoordinator interface. The source for the TrafficCoordinatorImpl class is given in Listing 4. The class declares a vari-

able called LightsList from the Java Vector class. This Vector variable is used for implementing the CORBA sequence declared in IDL. The class constructor prints an appropriate message to the user upon successful creation of the object. The accessor method for the lights attribute copies each element of the vector into an array of TrafficLight objects and returns the array to the caller. The modifier method for the lights attribute isn't implemented as this isn't required for the current application. The Register() method receives the reference to the traffic light object and copies it into the vector. The method then prints an appropriate message to the user upon successful registration of the light. The SetColor() method is called by the traffic cop application. The method receives two parameters – the light number and the color to be set. Note that the color parameter is passed as a Java String rather than as an object of the Java Color class. CORBA doesn't provide a Color data type. Thus, if you use the Color class in Java, you'll need to implement both the traffic cop application and the coordinator application. The method SetColor() iterates through the list of registered light objects, and for each object the internal light number is verified against the number received as a parameter. If the match is found, the SetColor() method on the matched light object is called. The TrafficCoordinatorImpl class is instantiated by the TrafficCoordinatorServer class.

The TrafficCoordinatorServer class source is given in Listing 5. The main() method of the class initializes the ORB, creates the coordinator object and exports it to ORB. The method then obtains a reference to the Naming Service and binds the newly created object to the Naming Service with the name Coordinator. The TrafficLight and TrafficCop objects locate the coordinator object using this name. The coordinator then simply waits for invocations to occur.

Finally, the TrafficCop class creates a traffic cop application. The cop application locates the desired controller object and receives a list of registered objects from the controller. The user interface then shows all the lights with the initial color for each light set to red. The object hierarchy for the user interface is shown in Figure 4.

The user interface allows the user to click on each of the traffic light objects to select the desired color. After the selection of state for each light object, the user presses the Set button to set the various traffic lights to the desired states.

The main() method of the TrafficCop class initializes the ORB and obtains a reference to the Naming Service. It then

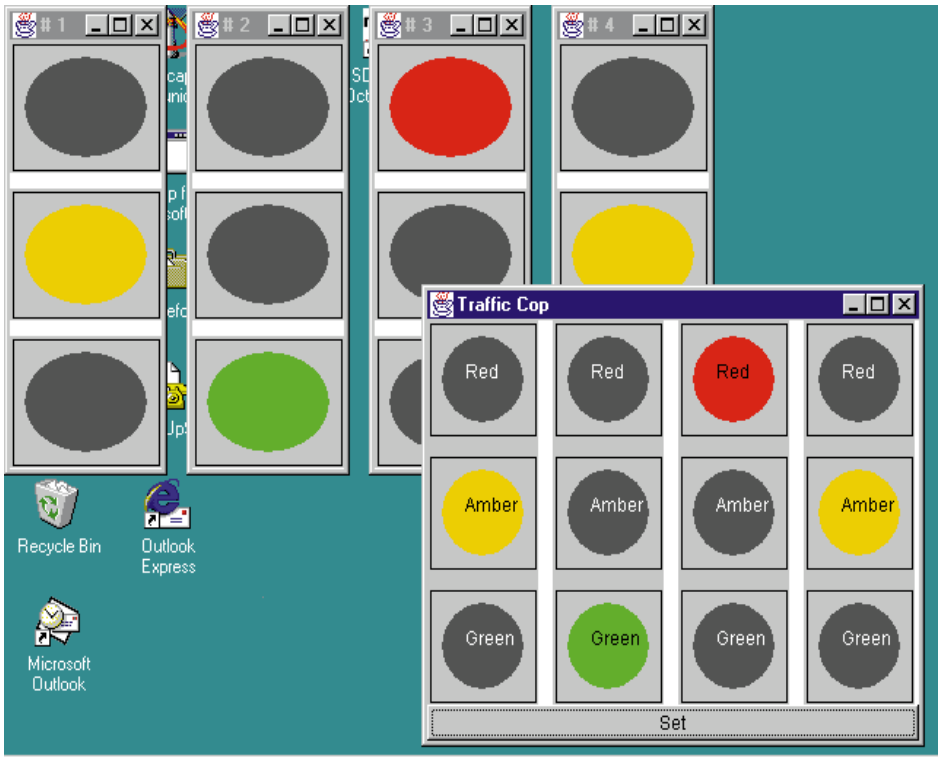



*Figure 2: The user interface of the traffic light controller application*

locates the Coordinator object and obtains a reference to it. The program retrieves the array of light objects from the coordinator by calling the accessor method – lights() – on the coordinator object. The main() method then creates an instance of the TrafficCop class. The constructor of TrafficCop calls the init() method, of the class to construct the user interface of the application. The class constructor calls the init() method, which constructs the object of the TrafficBasePanel class and a button object and adds the two components to the container using BorderLayout manager. The program then sets the window event listener and displays the frame window to the user.

The TrafficBasePanel class is derived from the Panel class. The class constructor looks up the number of light objects received by the cop object and constructs the number of TrafficLightsPanel objects that's equal to this number.

```
MAXPANELS = Cop.Lights.length;
LightsPanel = new TrafficLightsPanel[MAX-
PANELS];
```

Each TrafficLightsPanel object displays a traffic light consisting of three light objects. The class constructor calls the init() method, which sets up a layout manager to display the four light panels and add them to the base panel. Once again, for simplicity, only four lights are considered.

The Update() method of the TrafficBasePanel class is called by the TrafficCop object whenever the user presses the Set button. This causes a refresh of all the panels displayed on the Cop user interface to reflect the changes made by the user. The Update() method iterates through all the displayed light panels, retrieves the color setting for each panel and calls the SetColor() method on each of the light objects.

```
for (int i=0; i<MAXPANELS; i++)
{
 Cop.Lights[i].SetColor
(LightsPanel[i].clr);
}
```

The TrafficBasePanel class may hold up to four TrafficLightsPanel objects. The TrafficLightsPanel class constructor calls the init() method to do the user interface. The

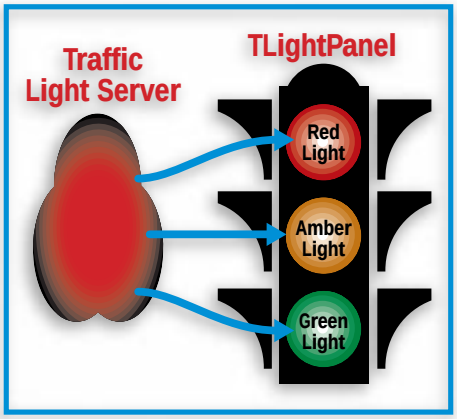



*Figure 3: The object hierarchy for the TrafficLightServer user interface*

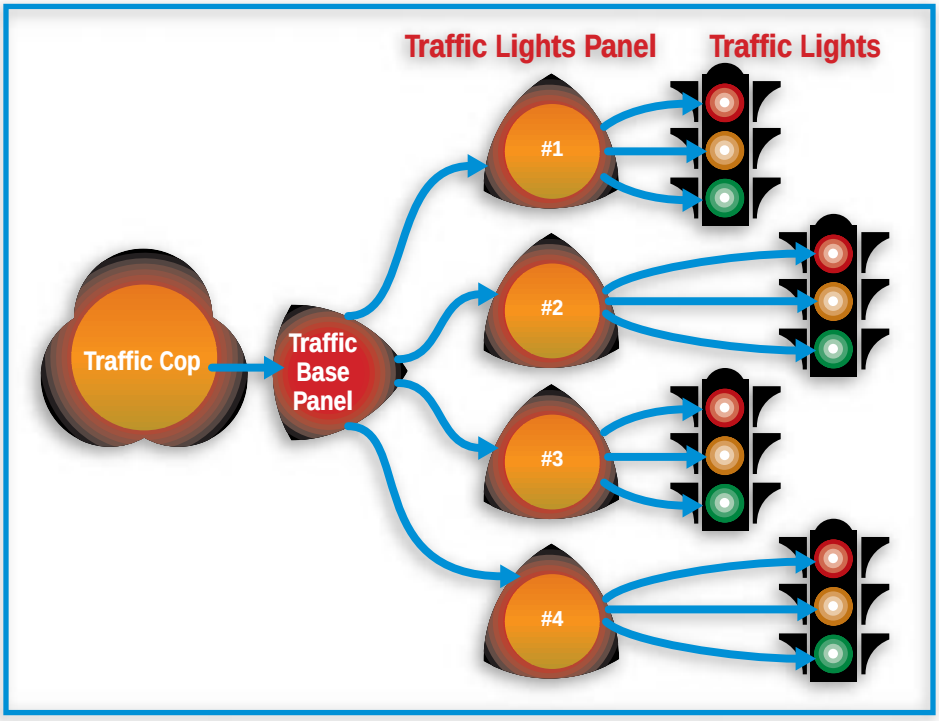



*Figure 4: The object hierarchy for the TrafficCop user interface*

init() method creates three LightPanel objects for red, amber and green lights and adds them to the container. It then calls the SetRedLight() method of the class to set the default light selection to Red. The SetRedLight() method sets the color of the red light to red and the colors for the other two lights to gray. Similarly, the other two methods – SetAmberLight() and SetGreenLight() – set amber and green lights, respectively, and set the rest of the lights to gray. The TrafficBasePanel class implements the MouseListener interface. In the mousePressed() event, the clr string variable of the class is set to the appropriate color value, depending on the LightPanel object being clicked. The event handler also calls the appropriate set() method for setting the color of the traffic light to the desired one. For example, a click on the red light highlights the red light and the other two lights are turned gray.

Last, the LightPanel class is used for drawing individual lights – red, amber and green. The class constructor receives a reference to the TrafficLightsPanel object so the mouse events can be passed on to it. The second parameter specifies the color of the light. Depending on this parameter, the appropriate color string – red, amber or green – is printed on the light. The paint() method constructs the visual appearance of the traffic light. The SetColor() method of the class is called whenever the user updates the display of the cop application by pressing the Set button. The method copies the received color into the local class variable and repaints itself to show the changes.

## Compiling the Application

As mentioned earlier, Visigenic VBroker for Java was used for developing and testing the application. The first step in building the application is mapping the IDL code to Java using the idl2java utility supplied by Visigenic. The following command line is used for mapping IDL code to Java:

```
idl2java –no_tie –no_comments traffic.idl
```

To compile the several Java classes discussed above and shown in Listings 2 through 6, use the make.bat file shown in Listing 7. This creates all relevant files for the application. The next step is to run the application.

## Running the Application

To run the application, start the osagent service and the CORBA Naming Service. The following two command lines start these services:

```
start osagent ñC

vbj -DORBservices=CosNaming -
DSVCnameroot=TRAFFIC -DJDKrenameBug
com.visigenic.vbroker.services.CosNaming.Ext
Factory TRAFFIC namingLog
```

Next, start the coordinator service using the following command line:

```
start vbj -DORBservices=CosNaming -DSVCname-
root=TRAFFIC -DOAid=TSession TrafficCoordi-
natorServer
```

Once the coordinator is started, create the traffic lights using the following command:

```
vbj -DORBservices=CosNaming -
DSVCnameroot=TRAFFIC -DOAid=TSession Traffi-
cLightServer 1
```

This creates Traffic Light #1 and registers itself with the coordinator. The appropriate message is printed in the coordinator window. Likewise, create three more traffic lights, replacing the command line parameter with the appropriate number for each light.

At this stage, the screen displays four traffic lights (see Figure 2). Next, start the traffic cop application by using the following command line:

```
vbj -DORBservices=CosNaming -
DSVCnameroot=TRAFFIC -DOAid=TSession Traf-
ficCop
```

The user interface for the traffic cop is shown in Figure 2. Click on the individual lights to set the desired colors and press the Set button. The corresponding lights will be set on the four light objects.

I didn't provide an unregister method for the traffic light. Thus, if you close a traffic light and re-create it, two copies will be registered with the coordinator. If you restart the cop application, both copies will be shown on the cop panel. To avoid this situation, close the coordinator and the cop and rerun the application if any of the lights need to be closed.

## Conclusion

Callbacks in CORBA allow the CORBA server to call a method on the client. The callbacks are achieved by saving the reference to the client in the server class. The server object is responsible for tracking such references and invoking any of the public methods on the desired client by using these references. Callbacks are very useful in real-life situations to notify the client of the occurrence of a certain event or of the completion of processing on the server end. This article has discussed one such real-life example and explained how CORBA callbacks are implemented using Java.

**CODE LISTING**

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

**About the Author**

*P. G. Sarang, Ph.D., is president and CEO of ABCOM Information Systems Pvt. Ltd., a consulting firm specializing in Internet, Java, CORBA, Visual C++ and VB programming and training. He can be reached at sarang@abcom.com.*

sarang@abcom.com

# JCalendar

## *This calendar widget is right on time*

*by* **Claude Duguay**

It seems ironic that the JCalendar widget was one of the first that came to mind when this column was being conceived. At the time, it seemed likely that Sun would include a calendar component with Swing – there were hints in the beta releases, and the preview directories contained minor evidence that this was one of the areas to be developed.

Swing 1.0 was released, later Swing 1.1, then JDK 1.2 hit the streets, and still there was no calendar component. Here you have it – JCalendar – with all the bells and whistles.

### Overall Design

As always, our design places considerable importance on flexibility. In keeping with the spirit of Swing, we support custom renderers, use the list selection model and provide a JComboBox-style popup implementation. Figure 1 shows a calendar view using the default renderer. The JCalendar widget lets you specify the number of horizontal and vertical months to display. The arrow buttons let you move to the previous or next month, as do the page up and page down keys. You can select multiple days, using either the mouse or the keyboard, and move around with the arrow keys. The home and end keys also let you jump to the first or last day.

To build JCalendar we'll need several supporting classes. Each day in this grid display is handled by the CalendarMonth class. The month title is provided by a CalendarTitle class, contained in a CalendarHeader with optional ArrowButton instances. Together these are managed by a CalendarView class. Because JCalendar needs to handle multiple views in a single framework, we use an additional CalendarGroup class that is aware of each of the other views and consolidates many of the common operations. Finally, we'll provide a JCalendarField class that works like a JComboBox.

### Rendering Days

The CalendarMonth class is responsible for much of the logic required by JCalendar. It handles keyboard events, renders each of the days, interfaces with the list selection model, and more. In this section we'll walk through the listings that relate directly to the CalendarMonth class. When you get the source code, run the CalendarMonthTest harness and you'll see something similar to Figure 2, displaying each of the days for the current month.

Listing 1 shows the code for our CalendarRenderer interface. We declare two methods to support both the rendering and the background color selection for the unused cells. The getBackdrop method returns a Color object. The method that does the actual rendering is called getCalendarRendererComponent. It requires that we provide a reference to the calling component, the value of the day being rendered (which is a string in this case) and two flags to indicate whether the rendering cell is selected and/or has the focus.

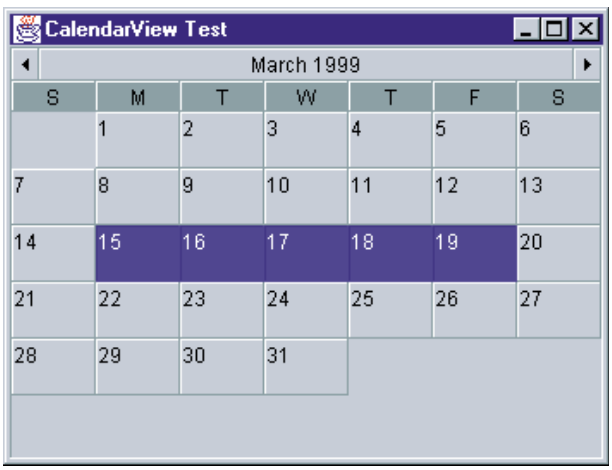



*Figure 1: JCMaskField classes*

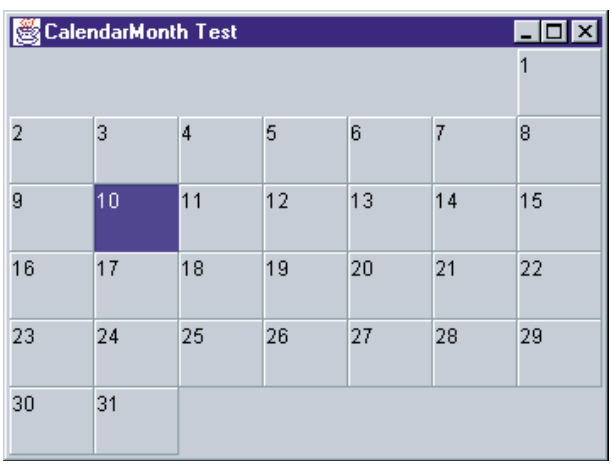



*Figure 2: CalendarMonth display*

Listing 2 is the DefaultCalendarRenderer implementation. As with most of the Swing default renderers, we extend the JLabel component. Our view will use a raised and lowered border to indicate selection, with a blue background to indicate focus. Our implementation is used to render both the days and the headers in the CalendarView. Nothing forces you to use the same renderer for these, but they implement the same CalendarRenderer interface. We make the distinction between headers and cell rendering in the constructor and save the value for later use.

The getBackdrop method simply returns lightGray for the background. Our getCalendarRendererComponent method does the real work. It sets the label text based on the value in the second argument, then sets the border, foreground and background colors depending on whether the cell is selected and/or has the focus. Listing 3 shows an alternate CalendarRenderer, called a SimpleCalendarRenderer, that provides a simplified, flattened view. The background is normally white and no borders are used. Selected cells are highlighted with a blue background. The focus and header underline are drawn by overriding the drawComponent method.

Listing 4 has the logic for the CalendarMonth itself. You'll notice that this class also requires a CalendarGroup instance. CalendarMonth is responsible for rendering and provides selection and notification along with keyboard and mouse event handling. It also exposes a few accessors and navigational methods. Dates are stored using Java Calendar objects that provide us with information about each year, month and day.

Some of the logic in CalendarMonth could have been handled by new Calendar methods from JDK 1.2, but you may need to get this running under Java 1.1.

The accessor methods are self-explanatory. The setDay method uses the setSelectionInterval from the ListSelectionModel interface to keep the model in sync. We use the Swing DefaultListSelectionModel in all our examples. Some of the support methods include nextMonth, prevMonth, setFirstDay and setLastDay. We also use a setActive method to set the active flag, which lets us determine if the current month has the virtual focus.

The rendering is handled in the paintComponent method and is delegated to the drawCell method, which actually calls our registered renderer and uses the Swing CellRendererPane to paint the component. The CellRendererPane is critical to this equation. It gets added to the CalendarMonth panel in the constructor as the Center component in a BorderLayout, so it expands to fill the viewing space. The CellRendererPane has an extended paintComponent method that takes the graphics context, the render component and the requested *x*, *y*, width and height.

The CalendarMonth paintComponent method draws the background based on the renderer getBackdrop color and then walks through a 6 x 7 grid, checking each cell for validity. This is handled by the isValidDay method. In addition, we provide an isSelected method to determine if a cell is currently selected in the ListSelectionModel. The isSelected method always returns false if the CalendarMonth is not considered active. This is important, since we may be sharing the same ListSelectionModel between multiple CalendarMonth objects. Because the renderer is not actually a child component, we also need to provide our own calculations for the getPreferredSize and getMinimumSize methods.

To handle mouse selection, we intercept the mousePressed and mouseDragged events, so we must implement all the methods in both the MouseListener and the MouseMotionListener interfaces. We register these in our constructor. The mousePressed event gets the focus and activates the clicked-on month before calculating the selected day from the mouse position. If the shift or control modifier keys are pressed, we extend the selection; otherwise we select the pointed-to day. In either case, we fire off an ActionEvent to any registered listener. The implementations for addActionListener, removeActionListener and fireActionEvent are listed at the end of the code. The mouseDragged event merely calculates the pointed-to day and extends the selection.

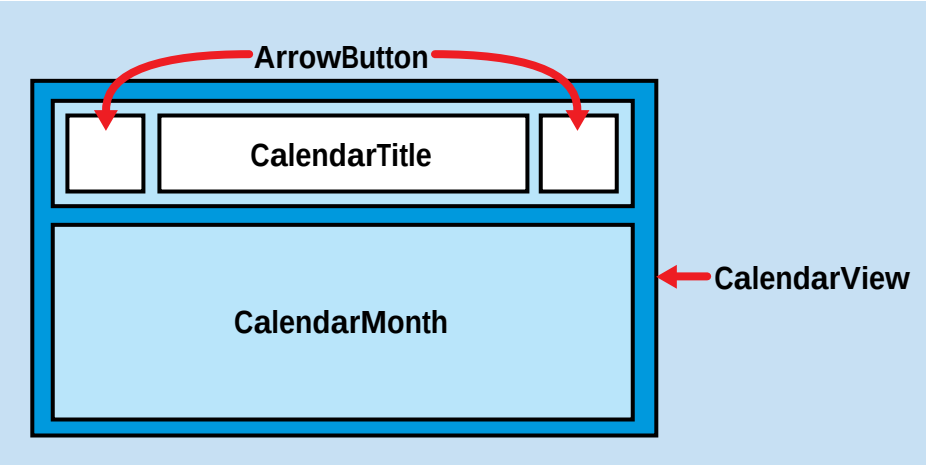



*Figure 3: CalendarView layout*

## Monthly Functions

Navigating between months can be done with the page up and page down keys, by moving beyond the first or last day of the month with the arrow keys or with the arrow buttons provided in the CalendarView. This class is largely a wrapper for several elements, as shown in Figure 3. The ArrowButton objects are optionally placed in the upper left and right corners, with the CalendarTitle between them. Let's take a look at the individual elements and wrap this section up with the CalendarView class itself.

The ArrowButton in Listing 5 is a simple extension of the BasicArrowButton. Our implementation extends BasicArrowButton, and overrides a copy of the paint method to make the border thinner in order to stay consistent with the rest of our elements, which use the ThinBorder class from Listing 6.

Listing 7 shows the CalendarTitle class, which extends JLabel to show the current month and year – a convenient way to set the border and alignment values. The CalendarHeader class in Listing 8 implements a few of the same methods used by CalendarMonth to handle render-


*Figure 4: JCalendar with a 2 x 3 layout*

ing. The CalendarHeader class shows the first letter of each day of the week above the CalendarMonth display, and uses the CalendarRenderer interface to do the drawing. As such, it uses the CellRendererPane and implements the getPreferredSize and getMinimumSize methods explicitly. We also implement the isFocusTraversable method to always return false since this object is not selectable.

We tie all this together in the CalendarView, as presented in Listing 9. Most of the work in our constructor sets up the layout and child components. The west and east arrows are created based on a pair of Boolean arguments passed in by JCalendar. After creating the CalendarMonth, we register as an action listener to respond to basic selection events. When the month changes, we need to update the CalendarTitle text, as formatted by the formatDate method.

## JCalendar Widget

The high-level view and widget control are pulled together in the JCalendar class. Figure 4 shows the JCalendar in action, with a 2 x 3 grid specified. As you can see, the arrow buttons are present only in the top left and right CalendarView. The behavior is complicated by our desire to allow the user to flow from month to month with the arrow keys and to automatically update each month as a sequence whenever a month is changed. Furthermore, we want to avoid flicker and provide seamless repaint events, so we need to implement a CalendarGroup to manage these effectively.

Listing 10 shows the CalendarGroup class. We keep track of the parent object so we can set paint events at the highest level after setting up the children This avoids unnecessary paint events and keeps the elements from repainting sequentially. The group elements are held in a Vector array and we use an active variable to save the current month index. While there are several methods in this class, they're generally unremarkable: setting and getting the current month, adding new members and testing or setting positions. The nextMonth and prevMonth methods walk the list of group members, calling their own respective nextMonth and prevMonth methods.

The main JCalendar class is presented in Listing 11. We provide a number of constructor variants with default selection model and renderers. Each one calls the main constructor, which sets up a GridLayout and populates the grid with CalendarView objects. This is where we decide which arrows are to be activated. Most of the following methods set and get properties for the date, selection model, header and cell renderers. We listen for CalendarMonth events and fire our own action events, implementing the addActionListener, removeActionListener and fireActionEvent methods to support this.

## JCalendarField

One of the best ways to put our calendar to work is in popup menus and ComboBox-style fields. To demonstrate this, we use the SimpleCalendarRenderer and produce a JCalendarField widget to add to your collection. Figure 5 shows how it looks when the user clicks on the arrow button. There are a number of minor tricks at work in this class. The class extends JPopupMenu and uses its ability to handle arbitrary components.


*Figure 5: JCalendarField popup*

There doesn't seem to be a way to extend JComboBox to use a new list popup. Since we need to use a component that needs to receive mouse events, we're forced to manage our own drop-down positioning as well. As with many of the previously mentioned problems, the solution actually involves only a few lines of code. The hard part, usually, is finding the solution. Keep that in mind – especially for this widget. If you implement one from scratch, it's likely you'll be spending much of your time in this part of the code – unless you remember how it's done.

Listing 12 shows the JCalendarField class. We provide two constructors, one of which uses the current date; the other expects you to provide one. Our main constructor sets up a BorderLayout with a JTextField in the center and a normal BasicArrowButton on the right, pointing down. To push the button into the field we get the field border and set it as the border for the whole component, setting the JTextField border to null. This keeps things consistent with the currently selected look-and-feel. We also create a JPopupMenu and add a JCalendar instance as its only child.

If the popup isn't visible when the button is pressed, we position it under the button and set the date to reflect the field value. Notice that we use getPreferredSize rather than getSize because the actual size is not actually established until the popup menu is first displayed. It causes problems with positioning if we don't use this approach. We also register to receive action events from the JCalendar object, closing the popup and retrieving the selected date when the mouse is clicked.

## Summary

Both the JCalendar and JCalendarField, with their respective renderers and other features, provide a great deal of power. The calendar metaphor is ideal for navigating temporal regions. This is useful in browsers for setting date ranges or simply for determining what day of the week a particular date falls on. For users, this is the intuitive choice, consistent with their material experience and easy to understand. You can make your programs more accessible and customize this calendar widget to your heart's content. Make it work for you.

**CODE LISTING**

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

**About the Author**

*Claude Duguay has been programming since 1980. In 1988 he founded LogiCraft Corporation, and he currently leads the development team at Atrieva Corp. You can contact him with questions and comments at claude@atrieva.com.*

claude@atrieva.com

CASE STUDY

# Pathways' SilverStream Solution

by Chad Ruff

SilverStream gets an *A*- for HTML development project

**Sage Software uses SilverStream to help social services agencies help the homeless**

**About the Author**

*President/CEO Chad Ruff founded Sage Software, Atlanta, Georgia, in 1995, and has concentrated his company's efforts on delivering state-of-the-art custom applications using PowerBuilder, MapInfo, Java and other Internet-related technologies. He can be reached at chad.ruff@sagesoft.com.*

chad.ruff@sagesoft.com

## Darwin's Natural Selection

About two and half years ago our company was hired to write an application that would allow our client, Georgia Tech Research Institute (GTRI), to access their financials over the Web. It was our first major push into data access over the Web and we were pretty hyped. GTRI hired us because we were a PowerBuilder shop and all of their applications were written in PowerBuilder. Both Georgia Tech and our company, Sage Software, were excited about using PowerBuilder's new Web.PB. It looked very promising.

We soon realized that none of GTRI's existing code could be reused for this project. The programmers before us were guilty of the same thing that Sage Software, and most PowerBuilder programmers, were guilty of: we wrote our business logic in the user interface and not in business objects. The idea of reusing the code was thrown out and we started from scratch.

Creating Web applications was very tedious at that time. Basically, you wrote and compiled an executable that would sit on the Web server. You started that application like a normal application; however, this application did not consist of a user interface. When a request from the Web came across the wire, it was directed to the CGI-BIN directory where a special application parsed the URL string and decided what to do with it. It first deciphered a directory in the URL string and determined the name of the program. It then looked in an INI file in the Windows System directory to find the IP address and port number.

Next, it instantiated a new user object and called one function inside that user object. If the function's main purpose in life was to display a tabular report, your function connected to the database, created a DataStore object, retrieved and looped through the data. You were responsible for taking each row and each column and inserting a <TR> and <TD> tag. Finally, you disconnected from the database and sent back a massive string to the Web server.

Now let's get complicated and say that the user wanted to scroll through the data. Your function would be required to do the same thing; however, you needed to track what row the user was on and what rows should be displayed next. Since the Web is a stateless environment, you had no way of knowing who the user was, what screen he or she was on, what rows had previously been selected, and so on. If you wanted to track information between each Web hit, you had to store it in the database or in cookies. Thus you'd reconnect to the database, retrieve the data (again), determine what the user did last, skip to that row and begin your <TR> and <TD> tags.

Forget about data entry and updates. You created your own form through concatenating a large string with form elements. Once the data was submitted, you'd construct your SQL statement and submit it to the server. As Archie Bunker would say: "Those were the days."

## The Strongest Shall Survive

Thank goodness for competition and the American way. Without good clean competition, I'd still be writing Web applications in this prehistoric manner. Two and a half years have brought us a long way – to the age of the Internet, e-commerce and application servers.

Like all good paradigm shifts, the industry latches onto a terminology that sticks. The new terminology is *application servers*. Depending on which vendors you talk to, you're likely to get different definitions of an application server. While there are several bullet points that define what an application server should be, for the moment let's forget about the essential – though less sexy – components, including load balancing and failover. That's terminology that the IT juggernauts like to use when ensuring upper management that the system is 100% reliable. I want to talk about the items that help me, as a developer, to be more productive in my work.

## State and Session Management

To me, state and session management is the single most important thing about an application server. Let's say that your application has 40 separate Web pages, all accessible by a left-frame navigation. I want my application server to be able to track what my user has done on each of the 40 pages. If the user goes to the search page and searches for and retrieves 200 rows of data, I want

that page to persist until the user's session times out. In other words, the user shouldn't have to hit the back button 10 times to get back to that search result. The user should only have to hit the search button once.

For shopping cart applications, I don't want to be responsible for tracking what users have selected in their carts by storing this information in a cookie or database field. I want the page to either track it or store the items in session variables.

I also don't want the overhead of connecting to the database for each transaction. I want a database connection pool that my users can share. The longest part of any application is the initial connection to the database.

### *Business Logic Host*

In true three-tier applications, your business logic resides on the middle tier. Application servers, by default, are that middle tier. You write your code in business objects and separate this business logic from the user interface. You also want this middle tier to be open so that it can talk to other transaction servers to get your suppliers.

### *Intelligent HTML Forms*

As a throwback from my PowerBuilder days, I want HTML elements that can be programmed. I want to be able to place an HTML field on the page and bind it to a database field. I also want the server to be able to create the necessary SQL to create, read, update and delete. I want each element to be able to be programmed such that when a user clicks on the submit button, I can reference the elements with more than a simple "value" method. I also want to be able to program these elements using Java.

### *Easy Distribution*

The most obvious advantage of deploying on the Web is the ease of distribution. We're no longer responsible for installing our executables and OCXs on our users' machines. Even though we think this deployment has simplified the use of Web technology, it can still be complicated. One project I worked on had static HTML forms, images, Java applets and classes, and Oracle Web procedures. To migrate from a development environment to a production environment wasn't an easy task. I want my application server to be able to publish everything with the click of one button.

### *And On and On...*

The list goes on for application servers, but these are my hot buttons. Don't forget about security, load balancing, thread pooling and access to different data sources such as Notes, PeopleSoft, ODBC and JDBC. We all have our hot buttons from the experiences we've suffered through over the years.

## The Project

Enough commentary. At Sage Software many of our clients turn to us for Web-based solutions to improve all sorts of business processes. One such project, for an organization called Pathways, began in October 1998. Based on the needs of Pathways, we decided an application server was the best approach and chose SilverStream Software Inc.'s application server for the project.

Pathways, Inc., is a collaborative effort of more than 25 local social services agencies, three local governments and United Way of Metropolitan Atlanta. Pathways and the Web application's goals are to:

1. Develop an innovative computerized system to help social services agencies work effectively together with individual consumers to help them recover from homelessness. The application will allow staff and volunteers at participating agencies to access the database simultaneously over the Internet.
2. Generate accurate demographic data on the size and characteristics of Atlanta's homeless population and chart the effectiveness of local programs that work with the homeless. Since Pathways' agencies supply most of Atlanta's homeless and homelessness prevention services, the database they share will quickly yield meaningful qualitative and quantitative data on one of the community's most pressing social problems. The data will be used to craft innovative local and national solutions.

During the bidding process, we asked the normal question: "Java or HTML?" The natural response was a rich user interface – thus Java. After a quick study of the infrastructure of Pathways, it was determined that the bandwidth was too small to support a large-scale Java application. Most agencies would be sharing a 28K modem to access the Internet. So HTML was chosen.

The next question: "What development environment should be used?" Pathways allowed Sage Software to select the best application server. In the past we used Active Server Pages (ASP) for HTML and SilverStream for Java. Since this was going to be an HTML project, we initially intended to use ASP; however, one of Pathways' requests was that they wanted an easy migration from the HTML environment to the Java environment. They were forward thinking and knew that one day the bandwidth would catch up.

This requirement paused our efforts and we reexamined SilverStream's HTML capabilities. After a quick evaluation, we realized that SilverStream was stronger in HTML generation than anything we'd ever seen. In addition, we could write all of our business objects in Java. This meant that when we decided to build our forms using Java, we could reuse all our business objects. Not a new concept, but SilverStream is practicing what the industry was preaching.

## The SilverStream Solution

We were awarded the project in mid-November 1998; development began on December 1. The project had massive amounts of data that we were to track and the project timeline was short. We were faced with approximately 20 pages that

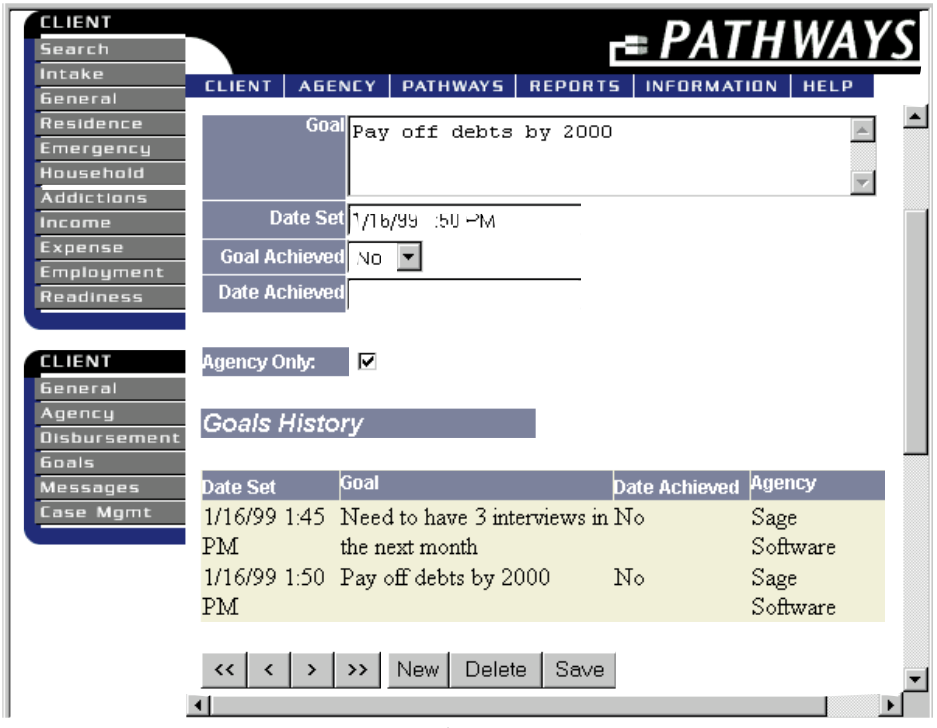



*Figure 1*

CASE STUDY

worked against 25 main tables and 30 code tables.

The application required each user to have job-specific security roles. These roles were needed to allow the user to see different pages and have different read/write access on each page. This was easily accomplished in the database by creating three tables – a user table, a role table and an objects table. The user table, of course, tracked all of the users. The role table created different groups, much like roles in Windows NT security or Oracle security. Finally, objects were created that mirrored the page names of the application. Read/write attributes were assigned to each role for each object and each user was assigned one or more roles.

In addition to the normal security defined above, some pages allowed the user to read and write data only to the agency to which they were assigned. In other words, if one user went into the case management page and entered notes for Agency A, another user assigned to Agency B could not see those case notes.

## The Beginning...

How was this application created in two months with a minimum amount of resources? Please keep in mind that this application dealt only with pages. Pages, by SilverStream definition, are HTML pages with a little JavaScript. For this application we didn't dabble with the Java Forms or Views. Even though there's a lot of power in the Forms and Views, we stuck with the Pages for this project. See Figure 1 for a basic understanding of the interface.

A simple login screen was required to allow the user to have access to the application. Using the Page designer, I simply created an HTML page with tables and a form. The form contained two single-line edits and a Login Button. Each object is no simple form element; rather, it is a Java object that has many properties and functions.

The Login Button has an event called pageActionPerformed. Listing 1 shows the Java code for this event. You'll notice that this is Java code, not JavaScript. This is a common misunderstanding.

Let's start with line 14. Here I access a field (HTML single-line edit) with a method called getValue. This simply gets the value of the text field object. I then check to ensure that the user has entered at least one value. On line 17 I call the agScriptHelper.alert() method. This produced JavaScript that is called on the body load event in HTML.

If the page passes both validations, I then populate a string with a query that I want to pass to my business object. (For security and privacy reasons I've altered the query statement, and I can't show you the business object.)

On lines 35 and 36 I populate a hashtable with two parameters; then, on line 41, I invoke a business object that performs the SQL against the database. The page gets passed back a result set if the query was successful. On line 55 I go to the first row of the result set (there should only be one row), and on lines 66 through 68 I retrieve the values of the first row. On lines 71 through 75 I set the session objects that will be used throughout the application. These session objects are the only ones required to run the application. Finally, I issue the showPage method, which will open up the frame set for the main interface.


If the user were to view the source, they would see Listing 2. As you can see, this looks nothing like Listing 1. The Java code runs on the server, and the user never sees anything except the simple HTML elements.

## CRUD

Create, Read, Update and Delete: these are the primary things you want to do when accessing your database. Clearly, SilverStream was designed from the ground up to be concerned about database access. All of the primary screens in the application require CRUD in one way or another.

Creating a CRUD page in HTML is very simple.

1. Select New Page from the page designer.
2. Select a data-aware page format.
3. Select the primary table.
4. Select the fields of the primary table.
5. Select the Style for the Page. The Style sets the default colors of the page and the text on the page. It also gives you the basic CRUD navigation buttons – First, Last, New, Save, Delete, etc.
6. Give the page a name.

SilverStream then takes over. It creates your HTML page and your data access object, then binds the fields to the form elements and provides the navigation bars.

If you're only trying to update the entire table, then you're finished. However, most of us want to update a subselect of the table. In our case we have a client record and we want to update all of the case management files for that client. Thus we only want to retrieve the client records in the case management table. There are several ways to do this. This was our approach:

We have a search screen that allows the user to search the database based on one or more fields. When the user locates the record that he or she wishes to modify, we set a session value. On the left frame various images were created for navigating to different sections of the client. The images are programmable, so we grabbed the session value and constructed the where criteria for the page. The where criteria consisted of "CLIENT_KEY={session value}". Then, using the above predefined method, we passed the page name, frame name, the where criteria and the sort criteria as the parameters. When the page loads, it reads in the values that were passed and constructs the query for the database. The sample code appears in Listing 3.

For each of the pages, Pathways wanted to see common information about the client. Thus we created a page that contained these elements – name, address and age. We created our own methods for retrieving the client information and saved the page. In certain pages, such as the Case Management page, we dragged and dropped the Header page, making it a subpage of the Case Management page. From the Case Management page we can treat the subpage as an object and call its methods and access its properties. How about that? Object-oriented HTML development.

Now let's add security to this screen. The security method is a common element for each page. It has a rather complicated SQL statement that needs the user ID and the page name as the parameters. For security reasons I won't display the code for this. I will tell you, however, that it was easy to accomplish this for all the pages. We added the security methods to the Header page. On the pageRequestBegin event we called the method and passed in the session object's user ID and the screen name. The security object would pass back a 0, 1

or 2. If it was a 0, the user didn't have access to the page and had to be directed to an error page. If the return was a 1, the user had read access only, and we simply set the property for the Save, Delete and New buttons such that the HTML would not be generated for the objects.

## Data Views

Data views are powerful objects within SilverStream. On each of the pages we allowed the users to update one record at a time. They could scroll through the records, save and delete, and then scroll to other records. To give the user an idea of how many records were in their select, we added a Data View just below the navigation bars. The Data View was linked to the page's cursor so that we could see all of the records for that select. The records in a Data View are displayed in a table with neat scroll bars that allow the user to scroll up and down on the records.

## Stored Procedures

I'm adding this section for the SilverStream enthusiasts who haven't coded their first stored procedure call. Putting your business logic in stored procedures is still a good idea for heavy transaction-based functions that hit many tables. Listing 4 contains a very simple call to a stored procedure in SilverStream. You'll notice, however, that this is a purely JDBC call, which is why it took me some time. I was looking for SilverStream classes, but there's no need to extend the JDBC calls.

Line 3 shows the connection string to the database. If you're unsure of the connection string, look in the SilverStream management console under the database that you're using. On Line 16 is the stored procedure that you'll be calling. You'll use the "?" for the parameters in this string. Don't attempt to concatenate a string that includes the actual values. The "{" is required too. On Line 19 you're connecting to the database. You will, of course, use your own user ID and password. On Line 21 you're creating your Callable Statement and on Lines 23 and 24 you're putting in your two parameter values. Finally, on line 25, you're calling the procedure. This stored procedure doesn't return a result set, which is why it took me a few hours to figure out. There are a ton of examples on how to get a result set back. As you can see, this is pure Java and JDBC, so it can be applied outside of SilverStream.

## Final Analysis

SilverStream is an excellent tool for creating fully functional enterprise Web applications. At the time of this writing, the development of the project is complete two weeks ahead of what we considered a tight schedule. We are now beginning the alpha test phase; however, Pathways has been able to see the work as it progressed. They provided valuable feedback along the way, saving us time in the long run.

The SilverStream newsgroup has been an excellent source for how-to's and problem solving. We only needed to call SilverStream once for assistance.

The entire project went quite smoothly, which is generally hard to say about any application server. It was a large project that we completed in just two months. The resources involved were two SilverStream developers, an Oracle DBA and a Visual Basic programmer. Your next question is: "Why a Visual Basic programmer?" We found that a majority of the work was creating the presentation of the screens and the layouts. Our Visual Basic programmer picked up SilverStream and created all of the forms without one day of training. The SilverStream coders came through and added the code once the forms had been created.

SilverStream gets an A- by my grading system. I don't give A+'s, and it must be flawless to get an A. I'll use it for all future HTML development projects.

**CODE LISTING**

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

# The Application Server Market

*by* **David Skok**

**Some 40 companies claim to have application servers, each offering widely different functionality. Where is the market headed? Where do the different types of players fit?**

The application server category is one of the more confusing markets to understand. In addition, the market changed rapidly during 1998, with a number of companies being acquired. This article aims to clarify the situation.

## Software: A History of Consolidating Services on New Platforms

Looking back at the major shifts that have taken place in the software industry, a trend emerges that is a helpful indicator of what is likely to happen in the evolution of the application server market. As new platforms emerge, there are generally a number of players providing point products that address a specific niche area. As the market matures, leaders emerge who tend to incorporate most of the point products' functionality into their primary product. An example of this is the relational database area where data warehousing, object databases, full text retrieval, transaction processing and bitmapped indexing have been consolidated into market-leading products such as Oracle8. In another example, we see SAP take a leadership position by integrating functionality that was previously available only in multiple point products. On the client side we see Microsoft integrating all of the standard office products into a single suite, and in the process eliminating players that have led a single product category. I predict that a similar consolidation of services is about to take place in the middle tier.

## The Next Consolidation: The Middle Tier Becomes Essential, Driven by Thin Client

The driving force for the shift to the next computing platform is clearly thin client and Web-based computing. Organizations frequently need to deploy e-commerce applications that are both consumer and business-to-business focused. Furthermore, there is strong pressure to provide ubiquitous access to applications at a low cost.

Supporting Web and thin client applications automatically implies a shift toward placing most of the processing in the middle tier.

## The Middle Tier Before the Web

Prior to Web-based and thin client computing, we saw a number of point product areas in the middle tier:

- Distributed object services (ORBs, OTMs, etc.)
- TP monitors, transaction managers
- Connectivity products providing application access to, and integration between, legacy data, ERP applications, RDBMSs, etc.

These products didn't provide facilities to support the generation of a user interface or what we refer to as "Presentation Services."

## Web Application Servers

Starting around 1994, the major drive to create dynamic interactive Web applications spawned a new kind of middle-tier product: the Web application server. The first generation of this kind of product was simply a Web server running CGI scripts. The second generation focused on making it easier to develop and deploy dynamically generated HTML pages, taking over where CGI scripts left off. In this generation the primary purpose of these products was to provide presentation services (see Figure 1). A major limitation of this architecture was that business logic was being embedded in scripts inside Web pages.

- ***Presentation services:*** Dynamic HTML generation, state and session management
- ***Integrated tools:*** Servers that come with integrated development tools to help build the HTML-based applications

### *The Next Step*

As Web application servers evolved, they added important facilities to increase their usefulness (see Figure 2).

- ***Limited object services:*** As developers realized the limitations of placing business logic in scripts on Web pages, facilities were added to partition this logic into reusable middle-tier objects. Most vendors added RMI and CORBA support to allow distributed access to these objects. Some vendors also allowed access to COM objects.
- ***Connectivity services:*** In addition to robust, server-grade drivers for the main

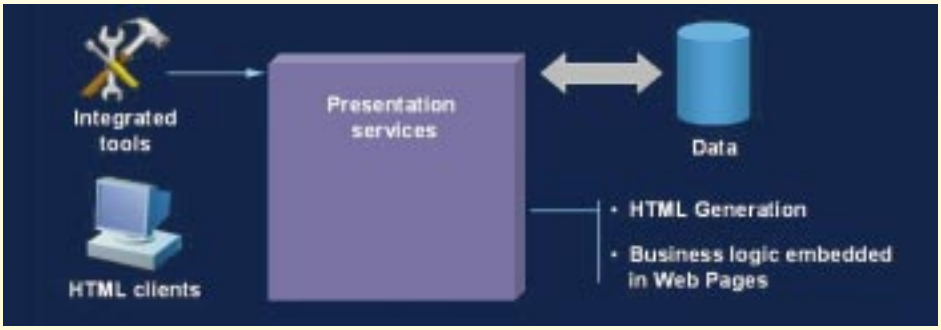



*Figure 1: Second-generation Web application servers*

## Enterprise JavaBeans (EJB)

EJB is an important specification in a number of ways:

- It defines a standard, vendor-independent way of writing middle-tier objects and provides standard interfaces for bean writers to access services for naming, security, transactions, etc.
- It makes the bean writers' job much easier as they do not need to be aware of distributed object/remote protocols, transactions, threads, security and state.
- It clearly separates the roles of the bean writer, application developer, container/server developer and the bean deployer/administrator. Standard interfaces govern the interaction between the different roles.

It's worth noting that EJB is a long-term vision. In version 1.0 the specification is immature and leaves many key areas undefined. Most players expect that the standard won't reach the level of maturity that will make it widely usable until version 2.0.

relational databases, connectivity is provided to access data in the major ERP applications and legacy systems.

- ***Enterprise deployment services:*** As scalability and reliability became paramount to supporting the large numbers of users accessing e-commerce Web sites with 24x7 availability, features such as load balancing and failover became important. Security and application manageability are the other key deployment services.
- ***Java clients:*** While HTML Web-based clients are satisfactory for many applications, heads-down production users working at a system eight hours a day need a richer UI. Java at the client meets the need while still retaining thin-client deployment advantages. Well-designed servers support Java clients over secure HTTP and provide a three-tier data architecture (queries executed at the server, with result sets remotely accessible via objects at the client). Facilities should be provided for automatic distribution and updating of the client-side application for applications deployed outside the browser.

## Web Application Servers Meet Distributed Objects

So far we've described two separate markets, the first consisting of object, transaction and connectivity services with no presentation services; the second, a market driven primarily by the need to deliver thin-client presentation support.

In mid-1998 these two worlds collided, resulting in major market confusion (see Figure 3). Leading up to the collision was increasing industry recognition that middle-tier distributed objects provided the right architecture for the creation of larger-scale sophisticated applications. Sun's Enterprise JavaBeans (EJB) specification was the catalyst for the collision. It provided a standard specification for how these middle-tier objects should be written, and made them far easier for the developer to write (the hard work would be done by the EJB container and server vendors).

### *A Market Thrown into Confusion*

Both the application server vendors and the vendors of object and transaction services realized that they needed to support the EJB standard; at this time a variety of different products changed their name to *application server*. All of a sudden there were 40 application server vendors, each offering widely different types of functionality. Some were focused on Web applications, offering strong presentation services; many others were pure object servers with no presentation services at all.

### *What Does It Take to Be an Application Server?*

As a result of the two markets converging, it is clear that we need a new definition of what is required to be an application server. Are presentation services important? Are connectivity services a requirement? Should the server come with integrated development tools?

The following section outlines where I believe the market is headed and explains the different component parts.

## Where the Market Is Headed

I believe that customers are looking for an application server that combines all the services shown in Figure 4.

The rationale for each of the components follows:

- ***Presentation services:*** Since it is the Web and thin client applications that are driving application logic into the middle tier, it is clear that application servers require presentation services to generate dynamic HTML pages; provide HTTP services or interface to an existing Web server, and state and session management; and support thin Java clients, including providing application destribution and updating for applications that are deployed outside of a browser.
- ***Distributed object services:*** These are clearly required in the form of EJB support to allow the easy wrapping of business logic and other components in a reusable, remotely accessible, secure object. This will include naming services and an ORB to allow IIOP-based communications.
- ***Distributed transaction services:*** These are required to coordinate transactions across EJB objects. For example, if you create an order form that is using an order object and an order detail object, you will need to use a transaction that wraps both objects when saving a new order. This service should also support transactions across heterogeneous databases (e.g., simultaneously update tables in both Oracle and DB/2).
- ***Application services:*** These should provide additional services to make creation of rich Web applications far easier. Examples of what should be provided here include e-mail send and receive with triggers for incoming e-mail, full text retrieval, content management and dynamic publishing for Web pages and applications such as product catalogs and online research, workflow and push capabilities.
- ***Connectivity services:*** These are clearly required to allow the newer Web or thin client applications to leverage existing data and applications. They should provide an open architecture for adding user-defined data connectors as well as off-the-shelf connectors to SAP, PeopleSoft, Lotus Notes, CICS, MQ Series, Tuxedo, etc.
- ***Enterprise deployment services:*** These are required for scalability, reliability, manageability and security. Features are load balancing, both server- and session-level failover, a management console and APIs, an SNMP agent to allow inclusion in third-party management systems and a complete security system including authentication against existing directories (LDAP, NT, certificates, etc.), encryption and access control.
- ***Integrated development tools:*** At a minimum, these are required to help developers create HTML applications that fully utilize the runtime features of the presentation services, and to deploy EJBs. However, customers are looking to simplify their development of these more sophisticated, multitier distributed applications. That requires a toolset that covers all of the services and that can be used by both expert Java programmers and 4GL programmers who know products such as PowerBuilder, Delphi and Visual Basic.

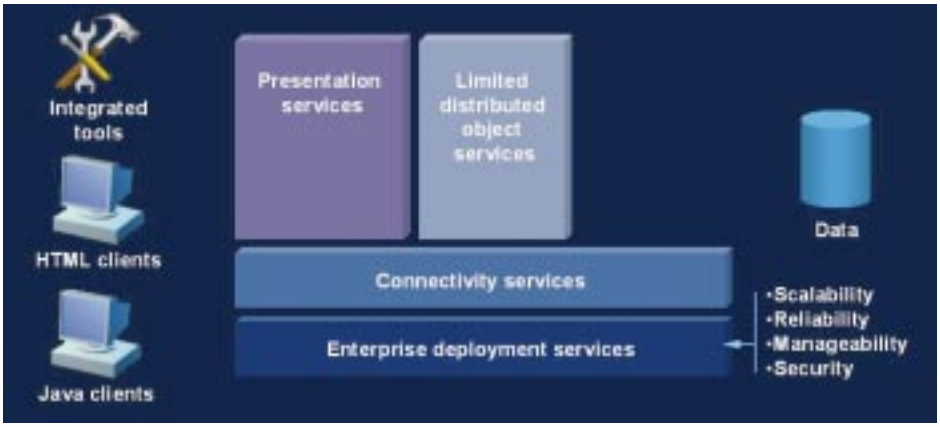



*Figure 2: Third-generation Web application servers*

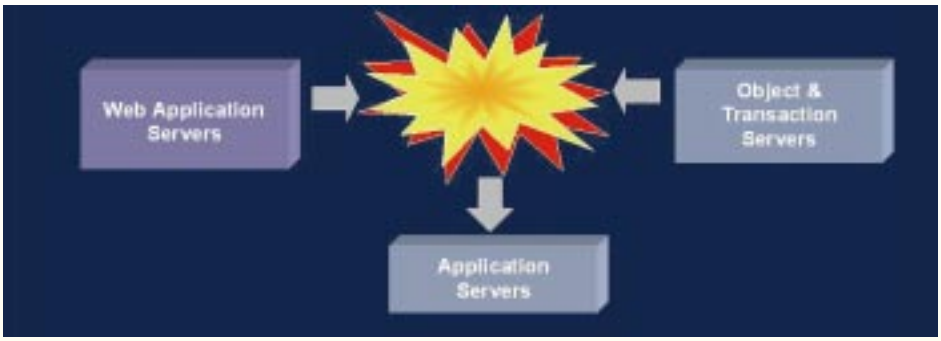



*Figure 3: Two markets collide*

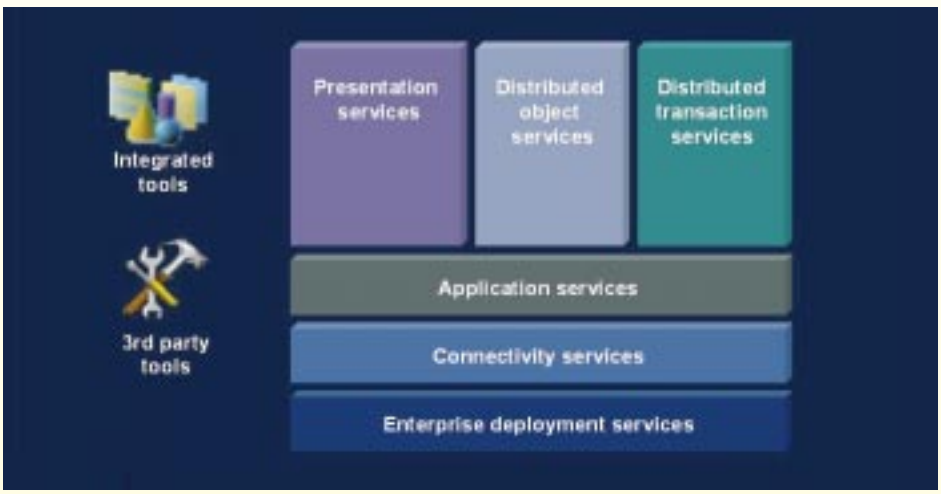



*Figure 4: The next generation of application servers*


*Figure 5: Where the players are today*

## Where the Players Are Today

Figure 5 shows three different types of players.

Examples of pure presentation servers are Allaire's ColdFusion and Microsoft's Active Server Pages (ASP). Examples of pure object servers are products like Iona, Oracle and WebLogic. Few products combine both presentation and object services in an integrated package. SilverStream and NetDynamics are examples.

It should be pointed out that no vendor has all the attributes required today, particularly as it will most likely require version 2.0 of the EJB spec to achieve this, and that is not yet available. Clearly, the right-hand group shown in Figure 5 is nearest to where the market is headed.

## The Importance of Neutrality

Application servers are expected to integrate with the wide variety of existing architectures and standards that exist in today's typical customer environment. These include:

- Multiple platforms: UNIX and Windows NT
- Multiple relational databases: Oracle, DB/2, Microsoft SQL Server, Sybase, Informix
- Multiple distributed object standards: CORBA, DCOM, EJB, RMI
- Multiple applications and legacy environments: SAP, PeopleSoft, Lotus Notes, CICS, MQ Series, Tuxedo, etc.

Based on the foregoing, customers are looking for application server vendors that are neutral and don't have a particular bias toward an operating system, data source, etc. Several of the application server vendors don't meet this requirement and consequently don't provide the necessary level of support for other competing products. This is generally considered a major drawback.

## Summary

This market has undergone substantial change in the last 12 months. It is still maturing and will probably take another 12 to 18 months to sort itself out. Most companies will be evaluating and deciding on their standards during 1999 and 2000 as this is a time-critical technology that must be used now to remain competitive. Although a single winner has not yet emerged, the leaders are clearly separating from the pack based on vision, technology, ability to execute, support services and strength of installed base. A clear requirement is that the product must integrate all of the above services, in particular offering strong presentation services combined with strong distributed object and transaction services.

**About the Author**

*David Skok, the chairman and founder of SilverStream Software, Inc., a company he formed in June of 1996, holds a bachelor of science honors degree from the University of Sussex, England. He can be reached for questions and comments at dskok@silverstream.com.*

dskok@silverstream.com

# SilverStream 2.0

## by SilverStream Software

*SilverStream attacks complex Web application development and deployment in version 2.0*

by **Steve Benfield & Brad Cooley**

**SilverStream 2.0**
**SilverStream Software**
One Burlington Woods, Suite 200
Burlington, MA 01803
www.silverstream.com
Phone: 781 238-5400 Fax: 781 238-5499
info@silverstream.com
Price: $8,500 per processor for NT or UNIX

To put it bluntly, SilverStream 2.0 sets a new standard for large-scale Web development and deployment. We first looked at the product in June 1997 when they were the newest entrant in the application server market. It lacked many enterprise features such as scalability, fault tolerance and CORBA support. In addition, it only offered advantages in the area of Java client development and deployment. With 2.0, things are quite a bit different.

SilverStream 2.0, released in October 1998, not only fulfills the early promise of the 1.0 product but includes innovative approaches for writing thin-client, HTML applications utilizing server-side Java business objects.

### The Application Server Market

The market for enterprise-class application servers is hindered by a confused corporate IS industry. Faced with impending deadlines for completing Y2K initiatives, the majority of the industry has been cautious about making platform decisions for three-tier Web-based applications. To define the product space for application servers, we'll use Forrester Research's simple definition: "a software server product that supports thin clients with an integrated suite of distributed computing capabilities. Application servers manage client sessions, host business logic, and connect to back-end computing resources, including data, transactions, and content." Given this definition, we'll quickly summarize how SilverStream addresses the main application server components.

### Server Architecture

The SilverStream server is written entirely in Java and can be deployed on NT, Solaris and HP platforms. The overall server architecture, shown in Figure 1, can be divided into three parts:

#### *Server Front End*

By default, SilverStream includes a high-performance HTTP 1.1 Web server that supports Secured Socket Layer 3.0 for encryption. However, SilverStream can easily coexist with other industry-standard Web servers, such as those by Microsoft and Netscape, by using the included SilverJunction plug-ins for URL redirection. For user authentication, the server integrates well with LDAP, JNDI, NT Security, NIS+ or X.509 digital certificates. SilverStream 2.0 also has solid support for building and communicating with CORBA objects via IIOP.

#### *Application Logic Tier*

From a Java application server standpoint, SilverStream includes full support for invoked or triggered business objects, servlets, state and session objects, HTML generation, table version tracking, access control and CORBA objects.

#### *Database Access Tier*

SilverStream uses connection pooling and sophisticated transaction management to relational databases via native JDBC drivers. All industry standard DBMS platforms are supported, including DB2, Oracle, Sybase, Microsoft, Informix, SQL Anywhere, Access and others. For performance purposes, asynchronous fetch-ahead and configurable data buffering is included. In addi-

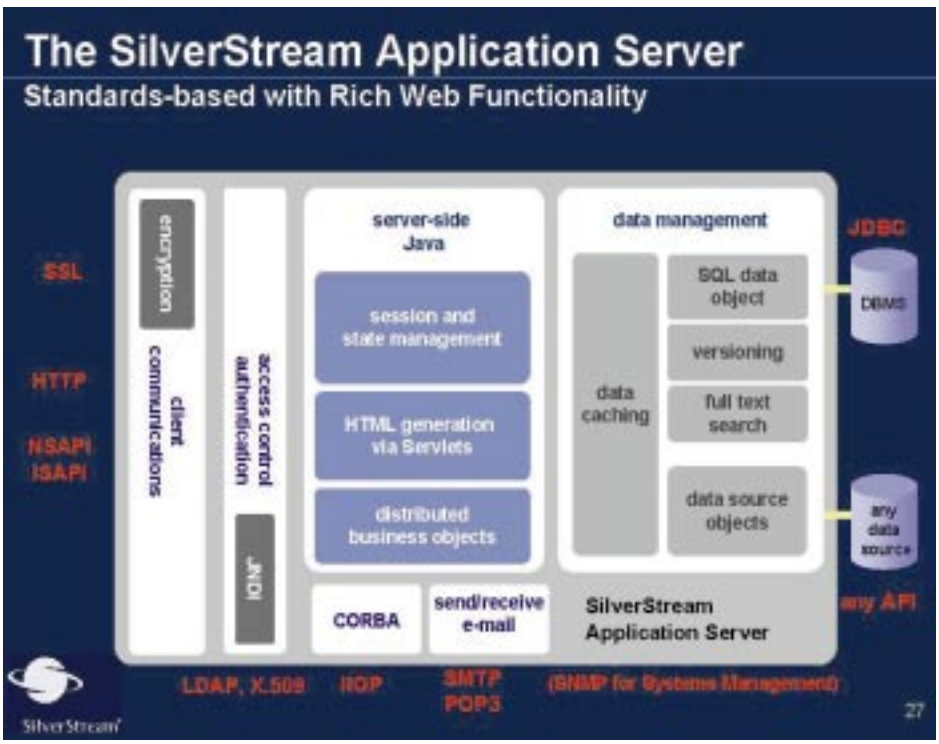



*Figure 1: SilverStream's server architecture*

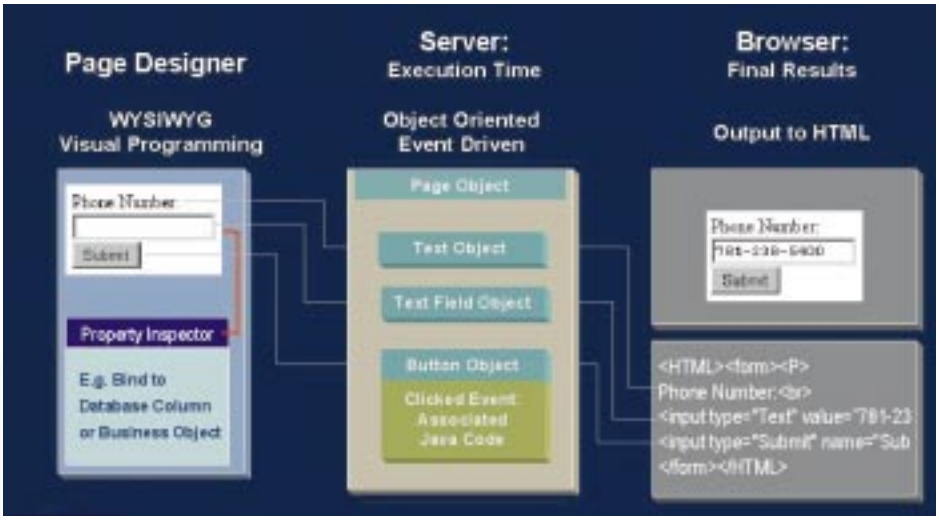



*Figure 2: SilverStream's HTML generation*

tion, SilverStream includes the Fulcrum Search engine with extensive, integrated support for full text queries. Data Versioning is also built-in.

## Server Deployment Options

SilverStream applications can be deployed on a single server (NT or Unix) or clustered. Clustering provides near-linear performance scalability and a separate cache manager coordinates server-cached data for the cluster. SilverStream includes a dispatcher to route incoming client requests to the cluster (third-party dispatchers are also supported). Multiple dispatchers can be used to further eliminate the possibility of a single point of failure. In addition, the SilverStream server includes load balancing and session-level failover mechanisms.

## SilverStream Designers

From a developer's perspective, perhaps the single most compelling reason to choose SilverStream is the integrated development environment, or "designers." Developers familiar with IDEs such as PowerBuilder will feel right at home bouncing around the Page Designer, Form Designer, Object Designer and other designers. The designer consistency is especially surprising in the Page Designer, the mechanism for creating thin-client HTML applications (more on the Page Designer later).

All designers offer the familiar event-driven programming model favored by products such as Visual Basic and PowerBuilder. The developer has full GUI control over a wide range of control options, most inherited from standard AWT classes. Double-clicking on any control takes you to the Java programming editor, where all Java classes are supported along with hundreds of prebuilt SilverStream Java classes and helper objects. SilverStream doesn't package their own version control system, but they do include built-in support for PVCS and SourceSafe. The Form Designer, used to create client-side Java applications, includes a debugger with all the expected features, including breakpoints, step-through and step-over. The Debugger doesn't work on the server; obviously, this is a deficiency. In SilverStream's defense, certain debugging APIs are still incomplete within the current Java specification or fragile enough to prevent SilverStream from making a tremendous effort at round-trip debugging. We expect to see round-trip debugging (Solaris, Microsoft, HPUX) in the 3.0 release of the product.

To the SilverStream server, the designers are simply additional users, making it easy to impose access control and other security restrictions on different sets of developers.

## SilverStream Forms – Client-Side Java

SilverStream builds on the award-winning Form Designer of 1.0 by adding capabilities for dynamic hierarchical views as well as the ability to deploy SilverStream Java client applications outside a browser using the Silver JRunner runtime Windows executable. SilverStream also generates Java 1.1 applets and can be used in any browser that supports it. SilverJRunner deployments actually run much faster than applet deployments because they don't perform the bytecode checking needed for the browser's security sandbox. In addition, SilverJRunner handles caching of applications to the client.

SilverStream's client-side applications consist of forms and views. When a form is created, it's assigned a default data source – most likely a table, although it could be some business object. SilverStream generates all the code needed to automatically retrieve data when the form is run on the client machine, and columns are easily bound to any data source. Two really nice controls that are included are a data-aware choice control for binding columns to lookup tables and a full HTML editor that can be bound to columns in your database. The HTML Edit Control allows you to build applications that allow end users to build their own HTML pages. A good example of this would be building product description pages. The editor is surprisingly rich in functionality and is easy to use.

In addition to forms, SilverStream has views. Think of a view as a data-bound hierarchical grid control. That is, you define column lists as you would for a grid control, but you can have unlimited levels of master-detail hierarchy. In addition, you can have multiple bands per level and multiple row selection, and to top it all off, it's an updatable control. It also allows you to define expressions for any column so you can perform tasks such as multiplying two columns and summing the totals.

## Handling Master-Detail Processing

While SilverStream does have all of these nice controls, what really makes it productive is how it allows you to create master-detail forms. For example, if you want to build a customer form that has a tab folder with customer orders and contacts, the form can be built with the wizards. Since SilverStream is aware of your referential integrity settings (primary and foreign keys), it knows how to relate forms to views. If you drop an order view on the customer form, it builds the WHERE clause for you. Then you'll see only the orders for the particular customer you're working on. If you take that same order view and place it on an employee table, you'd see only the orders that the employee sold. While SilverStream takes care of a lot of this for you, you can change things through the property sheets or through programming.

## Three-Tier Data Caching

A final note on forms and accessing data: as SilverStream downloads a form to the user's machine, it's already executing the query for that form (assuming the developer has chosen an automatic query). This means that as soon as the form instantiates on the user's machine, the data is most likely already streaming from the database to the application server. A thread on the server loops and retrieves 150 rows at a time from the database. Simultaneously, the client requests 100 rows at a time in a background thread. As soon as data is moved to the client, it's cleared from the server. What this means is that the amount of data on the server is always kept to a minimum. There's no need to load 5,000 rows to the server and then send them down to the client. The client gets the first

100 rows of data as fast as possible, while at any one time there may be only 500 rows sitting in the server as this data-caching mechanism does its job. If a certain form really needs all 5,000 rows immediately, a simple method call to the data cache changes the client-side retrieval to 2,000 rows at a time in a foreground thread. This entire caching mechanism is automatic and the developer doesn't need to code anything special to use it. The result is high data throughput as data passes each of the three tiers. It also results in faster response time in Java applications.

## Rich Application Server Services

SilverStream makes building partitioned, distributed business objects a much more intuitive process by providing a designer to create stateless, server-side, triggered business objects that respond to invoked requests, database-triggered events, POP3 mail events, scheduled events and/or standard servlet requests. In addition, SilverStream Data Source Objects (DSO) are easily built to provide a clean middle-tier partition to relational and nonrelational data sources, such as SAP, PeopleSoft, Lotus Notes and CICS. In fact, once created, developers code against DSOs the same way they code against relational data sources – the development environment treats data source objects just like tables and makes developing front-end apps against components pretty easy. Additionally, SilverStream includes support for persistent business objects.

Also new in 2.0 is SilverStream's data connector wizards. These wizards allow for the rapid creation of DSOs that go against SAP, PeopleSoft and Lotus Notes. Developers can also create their own data connectors for various nonrelational data sources of interest.

## Page Designer – True Innovation for Thin Client Applications

The most striking feature of SilverStream 2.0 is the Page Designer, which makes a major breakthrough in creating HTML applications by allowing the developer to use a designer environment much like Visual Basic's. It's complete with object-oriented server-side Java, event-driven programming, WYSIWG design features, control properties and methods. If you've built HTML before using scripting tools or even HTML editors, then you're in for a treat. It's hard to believe that someone didn't come up with this before.

During development, you define data sources (such as SQL or SilverStream's Data Source Objects), place controls on the page and bind them to columns, set properties, write server-side Java, write client-side JavaScript and basically do everything you need to do in one place.

As shown in Figure 2, all page controls render appropriate HTML under the standard Java servlet model. In addition, SilverStream includes built-in support for an extended event model, URL redirection, persistent session objects, access to server-side business objects, JavaScript helper objects, subpages, DataViews and framesets. For anyone trying to build business-oriented HTML applications, the Page Designer alone makes SilverStream worth evaluating. Since all page controls are beans, developers are able to extend them and create their own customized versions. Applications that we've seen include allowing controls to be enabled/disabled, thereby generating pure text or an input field during runtime, DHTML generation and XML generation.

> *"SilverStream makes building partitioned, distributed business objects a much more intuitive process."*

The Page Designer provides amazing level productivity. Because SilverStream handles master-detail transaction processing in pages similarly to forms, building complicated three-tier master-detail pages is a snap. SilverStream also handles full session and state management; passing data between pages is trivial and can easily persist across servers. In the persistent session object demo, we were able to completely shut off the server and reboot it, then hit Next on our page, and the application continued without a hitch – very impressive!

## Other Features

In addition to all the features listed above, SilverStream includes controls for file upload/download, the Visigenics CORBA IDL toolkit, a very powerful server management console, extensive data-binding capabilities, an authenticated version of Sybase SQLAnywhere, the ability to create JavaBeans and JARS, and a powerful mail API. SilverStream's unified security model can be applied to any component that is built, including row-level security against the database.

## Pricing

SilverStream is priced at $8,500 per processor, regardless of platform or number of users. In published performance specifications, SilverStream servers supported anywhere from 400 to 100,000 users, depending on application complexity and frequency of use. Development versions of SilverStream are available at $495 for a single developer, $2,495 for five developers and $4,995 for 10. With a full-priced deployment server, any number of developers can work at one time, making the cost per developer very trivial for larger organizations.

## What We'd Like to See in the Product

This version of SilverStream is so feature-rich, it lacks very little for the database-centric Web application developer. Nevertheless, we'd like to see some improvements in future releases, including a full-featured report writer, explicit XML support, a more powerful database administration tool, a round-trip debugger and better application documentation tools. In addition, SilverStream needs Enterprise JavaBean support. Luckily, this is under development and is slated to go into beta in late spring. With their EJB support, SilverStream will also be opening the server to third-party development tools such as Café and JBuilder. Likewise, SilverStream can be used as a front-end presentation server to existing CORBA and EJB servers.

## Conclusions

SilverStream is definitely going to be making the short list of application server platforms in the years to come. The tremendous talents of the development and management teams combine to make SilverStream a logical choice for developers who need the power to easily develop database applications to any data source, deploy on Unix or NT platforms, remain browser agnostic and desire the power and future promise of the Java language. A truly innovative offering, SilverStream's many strengths far outweigh its shortcomings. In an industry ready to explode, SilverStream raises the bar once again.


**About the Author**

*Brad Cooley and Steve Benfield are principals with Bondi Software, a software consultancy. Both have extensive experience with corporate business software development. Brad is in Greensboro, North Carolina, and can be reached at brad@bondisoftware.com. Steve is based in Atlanta, Georgia, and can be reached at steve@bondisoftware.com.*

brad@bondisoftware.com steve@bondisoftware.com

# SYS-CON RADIO INTERVIEW

## *Broadcast live at Java Business Expo in the Jacob Javits Center in New York City, SYS-CON Radio's Chad Sitler spoke with David Dewan of SilverStream*

David Dewan
VP, product strategy ***SilverStream***

**JDJ:** *How is Java used in SilverStream?*

**Dewan:** Java is used throughout SilverStream. The application itself is written in Java completely. It is now over a million lines of code., so it is one of the largest, if not the largest, commercial Java applications that exists. It is a full application server as well as a full development environment. All of the development tools, all of the interfaces, also are built in Java. The final thing is that Java is the programming language that our customers use. Since SilverStream is a development environment for building applications, the programming language in SilverStream is also Java. Of course, that ties in very nicely because you can build an application, just compile it and use it immediately. You don't have to go through the normal linking steps and so on. We found it a very productive environment. Our developers – and they are pretty conservative developers – estimated a 2 to 1 improvement in time to market doing SilverStream and Java as opposed to doing it in C. We have done a lot of C code over the years, so I think that assessment is probably correct.

**JDJ:** *You have a new release. Was it at the Expo? Did you debut it?*

**Dewan:** Just a little before the conference. So SilverStream version 2 is out now and of course in any version 2 you have a lot of features. The important areas were in scalability and reliability. Scalability... allows you to build a cluster of SilverStream Servers. So you put together an application that gets more popular than you thought – and that happens a lot on the Web – you can simply add another SilverStream server or three or four or five. And it scales almost linearly. We will get almost five times the performance out of a five-server cluster as with a single server.

Of course another benefit of that is reliability. If you are running a cluster of servers and one of them goes down, then the other three or four or whatever can continue running the application. And that's only part of it in terms of reliability. We have server failover, which means if a server goes down the applications can keep running. We also put in SilverStream 2.0 session-level failover, which means if you are halfway through entering your book order, for example, and a server goes down, the session itself from all the customers or users who are on that particular computer will get transferred with their session state information to another server and then the work will continue as usual. So that was certainly one of the big features in 2.0 – enterprise scalability.

The other thing in terms of major features we found is connectivity. We found that an application server is a piece of infrastructure. It needs to connect not just to relational databases for new applications but also to all the existing data sources. So for example, we added in DB2 connectivity for databases. We also added connectivity to Lotus Notes, for SAP, and for PeopleSoft and the ability in fact to connect to any data source. You just write some code in Java and you can connect to the data source. From the client developer's point of view, it looks just like a relational database table. So it is a very familiar way of programming but can connect to just about anything that's out there.

**JDJ:** *What kind of applications are your customers developing?*

**Dewan:** Well, it's all over the map. They have financial applications, accounting applications, workflow applications, distribution, some that are working internally within a company, perhaps monitoring new products or drug testing, for example, and other products that are doing e-commerce and are running literally all over the world. And we have two airlines that are doing scheduling and ticket sales over the Internet with SilverStream applications. It's a very wide range from internal control and workflow applications out through Web and e-commerce applications that run all over the world.

> ***"Since SilverStream is a development environment for building applications, the programming language in SilverStream is also Java."***

**JDJ:** *What makes your product more appealing to the consumer? What are they going to look at and say, wow, that's what I'm looking for?*

**Dewan:** One is the ability to build very large scalable applications. We found over and over again that an application that starts out for a certain sized audience gets loose and ends up with many more. For example, we have an application in the state of Texas to monitor children that are placed in foster homes. That was originally designed to work for just 12 judges. It turned out to be very popular; it is now used by 300 judges and the people who work in the court system all over the state, and of course they needed to scale up the support for that application. It is used all day by hundreds of hundreds of users throughout the state. So scalability, reliability, the ability to build large applications are important.

The second is more related to the development side. SilverStream has a full set of development tools for building Java-based forms, for building the Java logic – a color-coded editor and compiler and so on – all built in as part of the SilverStream product. And time-to-market is so important now, whether it might be time-to-market for an internal application or for something that's really going out to the market. It is always important to get these done quickly, to get it done well, then to be able to change it, to be able to modify it, to enhance, to add to it, to do that just about at real time. So the development tools that are built into SilverStream, I would say, are the second major area in terms of appeal of the product. You see it at the booth here at the show and see it on the Web site in terms of how that actually works.

The third big area has to do with business objects, the ability to encapsulate logic, such as server-scheduled objects connecting into a mail server. We've got CORBA support because there is plenty of CORBA in the world and the more enterprise an application is, the more likely it is to use CORBA. We put full CORBA support into the version 2.

And the final feature I think is the broad connectivity. Every single customer has a variety of data sources ranging from traditional databases, many different brands usually, to the more nontraditional sources. That could be CICS or MQ Series or it could be applications like SAP and PeopleSoft. SilverStream provides the ability out of the box to connect to a wide variety of data sources. Those have been the things I think that have been most appealing. Couple that with support that's worldwide, the strong training that's available – again worldwide – and a management team that has been through this before. So on the one hand we are not a group of 25-year-olds that can work all night. No, our all-nighters are behind us. But on the other hand, we have been through the experience of building a large successful software company and understanding the development process and so on, and I think certainly our large customer base. That is very important too. They know they are dealing with a management that understands their needs and concerns and has a track record of dealing with this.

# Java Developers Journal

# SYS-CON Interactive

# Java Developers Journal

# SYS-CON Interactive

# How to Convert from AWT to Swing

## *Will your Java code FINALLY fulfill the "write once, run anywhere" promise?*

*by* **Doug Porter**

Okay, you've spent a year building a Java app that will take over the market, but since Swing wasn't ready and you didn't have time to wait, you used the Abstract Windowing Toolkit (AWT). Now Swing is shipping, and any app using AWT is a dinosaur. You've got to convert your work – and fast. Here's how, step by step.

To follow these procedures you need to know Java and AWT fairly well. If you used a visual tool to build your user interface and aren't familiar with concepts such as components and containers, brush up on the basics first. Keep Swing's docs handy and you can probably pick up what you need to know about it as you go.

First, in each of your source files using AWT add:

```
import com.sun.java.swing.*;
```

If you're using panel borders, also add:

```
import com.sun.java.swing.borders.*;
```

If you're using the Standard Extension version of Swing, remember to substitute "javax" wherever you'd use "com.sun.java" – except, however, for the import statements, where you change "java.awt." to "com.sun.java.swing.J". Be sure to include the trailing "." in "java.awt.". If you're not concerned about name space collisions, you can just change "java.awt." to "J". Then you can change any other occurrences of "Frame" to "JFrame", "Dialog" to "JDialog", "List" to "JList", etc. In general, you simply need to add a "J" to the type. The AWT component's member functions are almost the same for Swing. For reliability, don't mix AWT and Swing components; instead, convert them all to Swing.

After you've added "com.sun.java.swing.J" to "Checkbox", change "JCheckbox" to "JCheckBox", with a capital "B". Next, replace any JChoice with JComboBox, and any Checkboxes (which are actually radio buttons) with JRadioButton. Then replace each corresponding CheckboxGroup with a ButtonGroup.

You also must replace any JChoice with JComboBox, change "getState()" to "isSelected()" and "setState(state)" to "setSelected(state)".

For in-source files built with Visual Café you must delete any lines that begin with //{{ or //}}. Then, in the constructor, remove:

```
setLayout( . . . );
addNotify();
resize( . . . );
setBackground(new Color( . . . ));
```

And for each component remove:

```
component.reshape( . . . );
component.setForeground( . . . );
```

and if present

```
component.setLayout(null);
```

Don't delete any other setLayout() calls.

Then change "new BorderLayout(0,0)" to "new BorderLayout()". If the show() member function was automatically generated,

### LISTING 1

```
void addListeners () {
    Window lWindow = new Window();
    addWindowListener(lWindow);
    Action lAction = new Action();
    // add one action listener for each object
    // for example, if you have an OK and CANCEL button,
        // you might use the following code
        okButton. addActionListener(lAction);
        cancelButton. addActionListener(lAction);
    }

    class Window extends java.awt.event.WindowAdapter {
        public void windowClosing ( java.awt.event. &
         WindowEvent event ) {
            Object object = event.getSource ();
            // replace Interaction with your class name
            if ( object == Interaction.this )
                Interaction_WindowClosing ( event );
        }
    }

class Action implements java.awt.event.ActionListener
{
    public void actionPerformed(java.awt.event.Action
     Event event) {
        Object object = event.getSource();
        // if you have OK and CANCEL buttons, then you
        // need to create the xClicked methods
if ( object == okButton ) {
okButton_Clicked(event);
   }
   else if ( object == cancelButton ) {
 cancelButton_Clicked(event);
   }
        }
    }

   void Interaction_WindowClosing( Event event ) {
       hide();
          // only call System. exit() if you're closing your
              application
       System. exit ( 0 );
   }
```

remove it, but if you created your own code to show(), leave it and be sure to override setVisible() to call your code. Next, change "JList(0,false)" to "JList()" and change "FlowLayout(FlowLayout.CENTER,5,5)" to "FlowLayout()".

In the parameter for event-handling functions, such as a button click or menu item, change "Event" to "java.awt.event.ActionEvent" and replace any symantec.itools.awt components with the appropriate Swing component.

For example, if you're using TabPanel, replace it with JTabbedPane. Symantec's components don't map closely to Swing components.

Also, change the order of component initialization so all components are added depth first. This means all components in a container must be added before that container is added to an enclosing container. Just look for calls to "add" and arrange them so the innermost appear first and the outermost last. Then replace any AWT listeners and handleEvent methods with Swing listeners.

Add a call to addListeners() in the constructor. Listing 1 contains some example code.

For containers with a BorderLayout, change:

```
container.add("Center", component);
```

to:

```
container.add(component, BorderLayout. CENTER);
```

Change East, West, North and South the same way.

For readability you may want to delete "com.sun.java.swing." except in the import statements. Pay close attention to the Swing threading rules. Swing components don't automatically add scroll bars as needed. If you want a component to scroll, first add the component to a JScrollPane, then add the scrollpanel to the container. Here's an example:

```
JScrollPane scrollPane = new JScrollPane ();
scrollPane. getViewport(). add ( myComponent );
myContainer. add ( scrollPane );
```

In some Swing containers, including JApplet, JDialog, JFrame, JInternalFrame and JWindow, you can't use add(), setLayout(), etc., directly. Instead, use getContentFrame() to get the right container. This is one of the few serious design errors in Swing. Set a layout explicitly for the container. The code used to handle it looks like this:

```
Container c = myContainer. getContentPane ();

c. setLayout ( new BorderLayout () );
c. add ( myComponent, BorderLayout. CENTER );
```

Don't forget that BorderLayout now has constants, such as CENTER and SOUTH, which are used for positioning in place of the old string literals. These constants are now the second parameter to add(), not the first.

In AWT applications there were advantages to laying out your components in the addNotify() member function. Lay out Swing applications in the constructor, as you always have for applets. Remember to lay out inner containers first.

Although you should be able to set up a ListModel or just use DefaultListModel with JList, in practice it seems to be more reliable to maintain a separate data Vector for each JList. Pass that Vector to the JList constructor. When you want to add, insert or delete data in the JList, make the changes to your Vector. Then reset the JList's data model to the changed Vector using setListData(). It's important to make changes only to the Vector that you passed to JList's constructor. Be careful that you don't accidentally pass a different Vector to setListData(), or it won't work.

To add an item to a list, you'll use code such as:

```
JList list;
Vector listVector;

public MyClass () {
    listVector = new Vector ();
    list = new JList (
}

void addItemToList ( String s ) {
    listVector. addElement ( s );
    list. setListData ( listVector );
  }
```

After you add all your components to their containers, call pack() on the outermost container. Do this before you do anything that needs the size of the container, such as centering it on the screen. Make sure Swing is in the classpath and run your application. You should see how it looks in Swing, and it should look about the same on every platform. Now your Java code is finally "write once, run anywhere"!


**About the Author**

*Doug Porter is cofounder and director of research and development for DeNova, Inc., manufacturer of the Java installer J'Express. He has more than 19 years' experience as a professional software engineer with three years experience in Java. He's just finishing his third commercial Java application. Doug can be reached at dporter@denova.com.*

dporter@denova.com

# NuMega DevPartner for Java

## by CompuWare NuMega

*Find out how fast your applications are really executing*

*by* **David Jung**

**NuMega DevPartner Studio 1.2**
**Compuware NuMega**
9 Townsend West
Nashua, NH 03063
Phone 800-4NUMEGA
1 603 578-8400 (International)
E-Mail: info@numega.com
Web: www.numega.com
Price: $599

Spaghetti code, sloppy algorithms, irrelevant code execution, dead code and so on can all lead to poor application performance. No matter how much planning goes into the design of an application, there always seem to be some gray areas where code performance runs slower than expected or doesn't execute at all. If you've ever encountered this, ever wondered how fast your application's functions and events are really executing, or if they are executing at all, then NuMega's DevPartner for Java is a set of tools you should definitely look into.

NuMega is best known for software and system debugging tools, such as SoftICE and BoundChecker. DevPartner for Java is a suite of tools that will help you analyze the performance of your application, discover where performance problems might be occurring and if the code was tested at all. It also offers feedback on your application's thread activity. This suite consists of three products: TrueTime, TrueCoverage and JCheck.

Earlier versions of the DevPartner for Java worked only with Microsoft's version of Java. At the time this article was written, you could use both the Java Virtual Machine (JVM) and the Microsoft Virtual Machine. For the JVM, DevPartner is compatible up to JDK 1.1.7a. As a bonus, both TrueTime and TrueCoverage aren't limited to analyzing applications written in Java; they can also analyze programs written in C++ and Visual Basic, as well as Java classes, DLLs and ActiveX components.

DevPartner comes on a CD and the installation is straightforward. NuMega uses InstallShield for the installation process, and there's a menu so you can choose which products you wish to install. By products, I'm referring to TrueTime, TrueCoverage, JCheck, the Microsoft Java Virtual Machine and Sun's Java Software Developers Kit 1.1.7a.

### The Test of Time

TrueTime comes with an intuitive user interface, as illustrated in Figure 1. The Project Pane on the far left contains all the active files pertaining to TrueTime projects including all the Java classes that are used. The pane on the right is where all the timing information for the sessions is displayed. The figure shows there are two session windows open. The name of the program that was executed is displayed as part of the information on the session window's caption bar. If you ran a Java applet, it would contain the name of the applet viewer. The session window contains panes of its own. The left pane is for filter information and lists all the functions used by the tested application, as well as the "Top 20" functions. These are Source Functions, Functions, Called Sources Functions and Called Functions. In order to view the source functions, you need to have a copy of the program's source files.

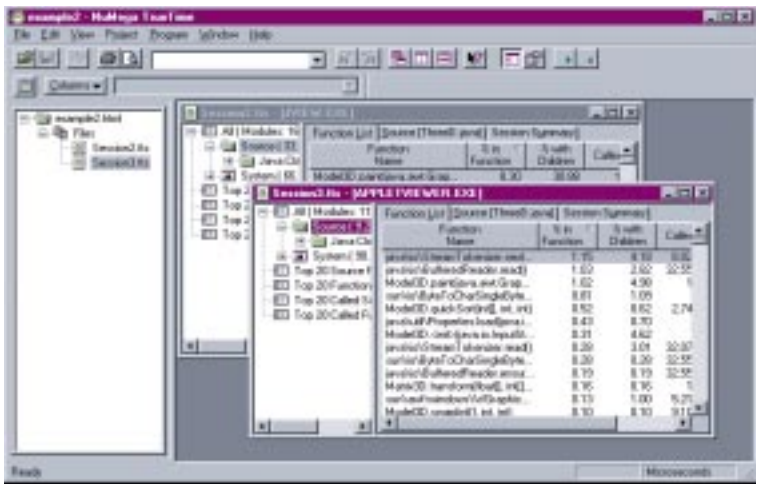


*Figure 1: TrueTime showing times for the same applet using JVM and Microsoft's VM*

An examination of the information on the session windows shows each window with three tabs: Function List, Source (with the name of the source file) and Session Summary information. When examining the Function List, double-click on any of the listed functions and you will receive detailed information about the function in a Functions Detail dialog window. It details all the parent and child functions and methods the function uses, how many times it uses them and the average time the function was executed. The Source tab describes source code for each function you're examining, how many times the line of code was executed and the percentage of time TrueTime spent executing the line as well as any child functions it may have branched to.

Since TrueTime works with various JVMs, you can perform benchmark tests to determine if there are any coding inefficiencies with different VMs. Not every VM is created the same nor are they all equal. TrueTime's Session Summary is extremely useful to get all the Java classes, their function and the VM in one report.

Many developers have the latest and greatest computers; therefore, their applications seem to run at light speed on their systems compared with their clients' slower systems. This is where TrueTime really shines because it doesn't matter if you're using a Pentium II 450 MHz system or a Pentium 75 MHz system. All percentage calculations are based on the CPU cycle times of your processor; thus your results will

be the same no matter what processor you run your tests on. This is very useful for establishing standardized benchmarks.

## Covering the Spread

In many ways, TrueCoverage's interface is similar to its counterpart, TrueTime. It sports the same Explorer interface that many tools do nowadays. The left pane, the Project Pane, contains a tree-view of the elements that make up the active project. In the right pane, all the information relevant to TrueCoverage appears. An Output window provides feedback for what system resources, such as DLLs, are used and where in memory they are stored. Another window, a session window, also has the Explorer metaphor. The left pane is called the Filter pane and it provides a tree-view of files that make up the project being tested – EXEs, DLLs and Java classes. The right pane is called the Session Data pane. This pane is the heart of TrueCoverage's reporting. The top part of this pane has a progress bar that illustrates the number of lines and functions that have been executed during the testing phase. There are a number of tabs that help further back down session information, like a list of functions, view source code and a summary sheet that provides statistical information about the session report.

Just like TrueTime, TrueCoverage will also report on any executable or resource file used by the project, such as ActiveX Controls and DLLs. In addition, if you have source code for the project's components, it will provide feedback on this usage percentage and the number of lines executed. If you double-click on the Java class, the source code will be displayed in the Source code tab in the Session Data pane so you can see exactly what line of code was executed and how many times it was executed.

The true power of TrueCoverage is its ability to compare and merge multiple sessions with one another. Why is this important, you ask? Within an application's development cycle, developers need to be sure that when they make a change to a program, they test the functionality they just created, added or modified. The first time TrueCoverage is run and a test script is used against it, the generated report becomes the benchmark. As changes and enhancements are made to the application, test scripts must be modified and performed again. Each new report will cover which procedures were run, which ones weren't, how many times the functions, procedures and methods were used, and so on. You can then merge these results with previous reports to determine if the application is executing the correct logic. Unfortunately, it doesn't tell you in what order the functions or procedures were executed, which can be helpful in determining execution order. In event-driven programming, code doesn't always execute in the order you think it does.

## Check, Please!

Now for something completely different, but extremely useful. At first glance JCheck might seem a bit awkward, but that's because most developers have never seen a thread model, even though they may understand threads. The folks over at NuMega obviously have, as illustrated in Figure 2, because the purpose of JCheck is to provide graphical representation of a Java class's thread activity. It also detects and displays events in the order they were executed, and detects and reports Java thread errors and thread leaks.

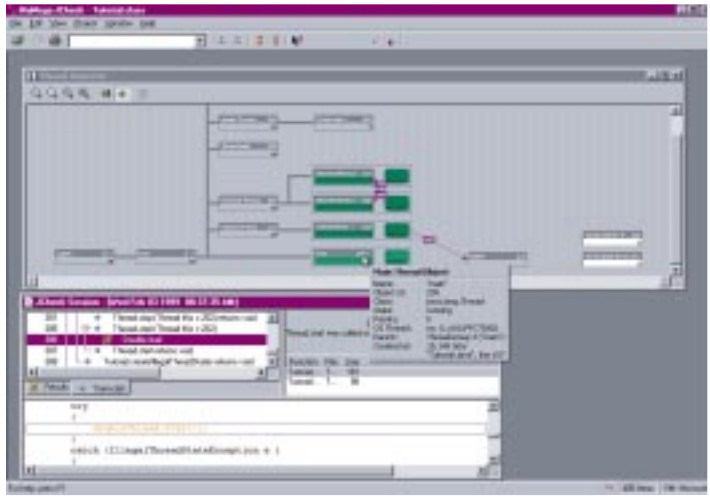


*Figure 2: JCheck's thread model*

The graphics representation of thread activities is displayed in a window called the Thread Inspector. It provides information about the Java class's state and interaction with Windows threads, synchronization objects, thread groups and other threads. This is very useful in determining problems with thread synchronization and timing issues, thread problems like deadlocks and thrashing, and run-time issues. To find out more about a thread object, simply place your mouse over the object and a pop-up window displays some basic properties about it.

Another window, called the Session window, is used for monitoring the method calls, class loads, thread states, context switches, synchronization objects and exceptions while you analyze your applet or application. The output of this is placed into two tabs – Results and Transcript. The Results tab is where detailed information about the source code's errors and leaks are reported. These are not errors that would be caught when an application is compiled. They are due to bad coding, like coding a "double-start," which occurs when you start a thread that was previously stopped. Not only does JCheck report these errors, it also attempts to offer solutions to fix the problem. Now it's not going to be able to offer solutions for every problem, but it does a good job solving most of the major threading problems.

## Conclusion

The only major drawback I could find is that NuMega does not make a version of this suite for the Unix environment. However, many developers have multiple operating systems at their disposal, so this shouldn't be the real issue, nor should it stop you from considering this product.

The documentation is a sparse booklet that describes the functionality of each product. More information is contained in Adobe Acrobat PDF files, but don't expect to gain extensive knowledge from them. Don't let the limited documentation fool you, though. After about an hour with each product, I really felt I had mastered them and was able to get the information I was looking for. The fact that each product links the errors it finds to your source code is a major advantage and a big time-saver. For Visual J++ users, the fact that both TrueCoverage and JCheck have toolbar buttons for the Visual InterDev development interface is a nice touch.

With applications developed in components and distributed throughout the enterprise, the last thing you want is for your application to be the weak link in the chain. By using TrueTime, you can isolate areas of your program that are running more slowly than anticipated. You can also find out which functions are being called most often and determine if they are running as optimally as they should.

Like all the NuMega products, all three are solid, useful utilities. Whether you're developing 100% Pure Java programs or applications specific to the Microsoft VM, this tool is a must-have in any developer's toolbox.


**About the Author**

*David Jung is an application developer specializing in client/server development using Visual Basic, Java and other Internet technology. He is also co-author of several Visual Basic books and can be reached at Davidj@vb2java.com.*

davidj@yb2java.com

# SYS-CON RADIO INTERVIEW

***Broadcast live at the Java Business Expo in the Jacob Javits Center in New York City, SYS-CON Radio's Chad Sitler spoke with Dean Guida of ProtoView Development***

Dean Guida
President, CEO
***ProtoView Development***

JDJ: *Can you give us an idea of what you have to offer and what you have out on the market?*

Guida: ProtoView has been in the component market for over 10 years now. We started off with ActiveX components, and about two and a half years ago we got into building Java components for professional developers. Right now we are providing a lot of GUI components, and we just came to market with the JFCSuite, a package of GUI components that create added value for the JDK 1.2 and, as some of us know it, the Swing Tool Set. We found a lot of missing holes in the package. For example, we are geared toward professional businesses so we created some currency components, time, date, and we focus a lot on the richness of creating some very visual effects. We have a very graphical calendar with over a hundred different properties that you can set through the customizer. We really focused on an easy API to program, snap-in components, the reuse of it, and providing the missing holes in the JFC.

In our JFCSuite we feel probably the biggest component in there is our JFCDataExplorer, and that metaphor came over from the Windows platform where you are seeing a lot of this Explorer metaphor. We have the left-hand pane of a tree view, the right-hand pane being a container that could be anything, a panel, another component, a calendar. So as you are clicking on nodes in this left-hand pane of the tree, you are displaying some data input screens or other types of components in the right-hand pane with a splitter bar. What ProtoView did was to take the base JFC classes of the tree and the table in there and we put it together into one coupled component, integrating these two with some advanced data model views to make it simple to add data to it through this architecture and giving that familiar user interface of the JFC look and feel.

JDJ: *Who would you consider your biggest customer base? Mainly for the enterprise, or can someone who is just starting out with Java or a similar language jump into something like that?*

Guida: When they finally buy a ProtoView component, they are usually a professional developer solving business problems. They are getting paid to solve a problem and have already chosen Java to solve these problems. They usually don't come to us until a specific problem arises. Really, we are having developers when they buy our product…it is not an IDE where they are learning it as they go. ProtoView customers are actually at a point where they need to solve some problems and they have to go outside their current box, be it Visual Café, JBuilder, Visual Age, etc…. Our developers range from a one-man consulting company to your Fortune 100 companies.

JDJ: *What do you foresee in the near future for Java and the entire Java industry.*

Guida: We are very excited about the Java industry. As a company we provide added-value components, so we are actually creating ActiveX, architecture-type components and Java components. We are having some very large companies deploy enterprise-type systems with either or both of our components. We only see Java growing and maturing. Right now everyone talks about Web applications and Web technologies, but soon it is just going to be about application development. The lines are going to go away and we will begin to see pure software development; regardless of the platform or technology used to solve the problem. I think Java is going to be the cornerstone of that. So it is really – it is computing, it is software..

# JSP vs JSP

## *A comparison of two Java Server Pages – their similarities and differences*

*by* **Rob Tiffany**

Java Servlets provide a number of significant benefits to Web and application servers everywhere:

- The ability to write a server-side application that can run without regard to the hardware, server operating system or Web server
- A dramatic performance boost over CGI or interpreted script applications
- Increased productivity using the Java language to build such applications

The third benefit has seen an even greater boost in the form of dynamically compiled Web pages when using the Java Web Server from Sun Microsystems; this technology finally gave servlet developers the RAD technology they've needed to produce complex Web applications faster.

The next step in the trend toward greater productivity has been taken and is called JavaServer Pages, or JSPs. This time around you're not limited to using the Java Web Server. Third-party servlet engine providers, such as Live Software and IBM, allow you to deploy JSP applications on every major Web server and platform. All the players in this arena adhere to a basic specification as defined by Sun, but not all JSPs are equal. (For example, servlet pioneer Live Software has added objects that closely resemble the objects found in Microsoft's Active Server Pages.) Obviously, this lends itself well to a direct comparison between what Sun and IBM have to offer and what Live Software brings to the table. This article will give the reader a comparison of the two JSPs by delving into the details of their respective specifications and bringing to light their similarities and differences.

Let's begin with the JavaServer Pages specification, as defined by Sun Microsystems. Essentially, this technology allows you to embed the Java language in Web pages that end with the .jsp extension rather than .html. What they're doing here isn't terribly different from what they've done with their .jhtml pages on the Java Web Server. (See my ***JDJ*** article [Vol. 3, Issue 6] on "Dynamic Page Compilation with the Java Web Server" to update yourself on this technology.)

Sun begins by changing .jhtml tags like <java>...</java> to Active Server Page tags like <%...%>. Sun divides the JSP spec into five different areas, the first of which is Directives. Table 1 describes the six JSP Directives to be inserted at the top of your Web page.

The second area defined by the spec is JSP Declarations, which define class-wide variables and methods using the SCRIPT tag. The SCRIPT open tag is <SCRIPT runat=server> and the close tag is </SCRIPT>. The "runat=server" part of the tag is required to ensure that your Web browser won't think that your JSP is defining client-side JavaScript code. An example is shown below:

```
<SCRIPT runat=server>
int I = 0;        String X = "Hello";
Public void Send() {
//code
</SCRIPT>
```

JSP Scriptlets define the body of the generated Servlet's Service method, and are the third area defined by the JSP spec. They look like this: <%...%>. Most of the Java code to be written will end up between these two tags. Table 2 describes the four

| | |
|---|---|
| 1. The scripting language being used: | <%@ language="java" %> |
| 2. The interfaces being implemented: | <%@ implements="com.insource.MyInterface" %> |
| 3. The class that the Servlet extends: | <%@ extends="javax.servlet.http.HttpServlet" %> |
| 4. Packages that the Servlet imports: | <%@ import="java.io.*,java.util.Hashtable" %> |
| 5. Content type: | <%@ content_type="text/html;charset=UTF-8" %> |
| 6. Servlet method being used: | <%@ method="doPost" %> |

*Table 1: The six JSP directives*

| Object | Explanation | Example |
|---|---|---|
| request | The servlet request class as defined by javax.servlet.http.HttpServletRequest | <% String Name = request.getParameter("Name"); String Phone = request.getQueryString("Phone"); Cookie inCookie[] = request.getCookies(); %> |
| response | The servlet response class as defined by javax.servlet.http.HttpServletResponse | <% Cookie outCookie = new Cookie("Name", "Value"); Response.addCookie(outCookie); Response.sendRedirect("http://www.conoco.com"); %> |
| out | The servlet output writer class as defined by java.io.PrintWriter | <% out.println("Insource is great!"); %> |
| in | The servlet input reader class as defined by java.io.BufferedReader | |

*Table 2: The four predefined variables used throughout JSP applications*

| Method | Explanation | Example |
|---|---|---|
| setAttribute(String objName, Object object) | Stores the object with the given object name in the application | <% Application.setAttribute ("AppArray", "MyArray"); %> |
| removeAttribute(String objName) | Removes the object with the givenname from the application | <% Application.removeAttribute ("AppName"); %> |
| get(String objName) | Returns the object with the objName | <% String X = Application.get("AppName"); %> |

*Table 3: Syntax for using Application object and its associated methods*

| Method | Explanation | Example |
|---|---|---|
| Cookies(String name) | This method returns a single cookie object with the given name | <% Cookie c = Request.Cookies("CookieName"); %> |
| Cookies(String name, String subkey) | This method accesses the subvalue of a multivalued cookie | <%=Request.Cookies("CookieName", "type1"); %> |
| Form(String name) | Returns the form value for a given name | <% String x = Request.Form("StringName"); %> |
| QueryString(String name) | Returns the querystring value for a given name | <% String lastname = Request.QueryString ("QueryStringName"); %> |
| ServerVariables(String header) | Returns the header value for a given header name | <% String lastURL = Request.ServerVariables("HTTP_REFERRER"); %> |

*Table 4: Request methods*

| Method | Explanation | Example |
|---|---|---|
| CharSet(String charSet) | Appends the name of the character set to the content-type header in the Response object | <% Response.CharSet("ISO-LATIN-7"); %> |
| ContentType(String contentType) | Specifies the HTTP content type for the Response. If no content type is specified, the default is text/html | <% Response.ContentType("image/JPEG"); %> |
| AddHeader(String name, String value) | Adds an HTTP header and will replace an existing header of the same name. This must be sent before any other output is sent to the client unless the Buffer property is set to TRUE | <% Response.AddHeader("WARNING", "Error with your DHTML"); %> |
| sendRedirect(String url) | Causes the browser to attempt to connect to a different URL | <%Response.sendRedirect("http://www.insource.com"); %> |
| buffer(boolean buffer) | Indicates whether to buffer page output and not send a to the client until all server code has been response processed, or until the Flush or End method has been called | <% Response.buffer(TRUE); %> |
| appendToLog(String logString) | Adds a string to the end of the Web server log entry for this request | <% Response.appendToLog("I love server-side programming"); %> |
| binaryWrite(byte[] data) | Writes the specified information to the current HTTP output without any character conversion. This method is useful for writing nonstring information such as binary data required by a custom application | <% Response.binaryWrite(BarChart.Image); %> |
| clear() | Erases and buffered HTML output with the exception of the response headers | <% Response.clear(); %> |
| end() | Causes the Web server to stop processing the code and return the current result | <% Response.end(); %> |
| flush() | Sends buffered output immediately but will cause a runtime error if Response.buffer has not been set to TRUE | <% Response.flush(); %> |
| write(String s) | Writes a specified string to the current HTTP output | <% Response.write("Hello World"); %> |
| Cookies(String name) | Returns the Cookie object with the given name and is also used to set the Cookie's properties | <% Response.Cookies("CookieName").setMaxAge(60*60*24* 7*8); %> |
| Cookies(String name, String value) | Assigns the value of the cookie of the given name and overwrites old values if the cookie already exists | <% Response.Cookies("CookieName", "CookieValue"); %> |

*Table 5: JRun Server Page Response methods*

predefined variables that you'll use throughout your JSP applications.

The fourth area defined by the spec is JSP Expressions, using tags like this: <%= %>. Expressions specified within these tags will first be evaluated, with the result converted into a string and displayed. Any primitive type between these tags will be converted to a string, as follows:

```
<% int I = 25; %>
<%I %>  //prints out 25 as a string
```

The final area defined by the JavaServer Pages spec is the Bean tag. This tag gives your page a conduit to regular or Enterprise JavaBeans that will encapsulate business logic separately from the content presentation. Although it's not a requirement, the bulk of your Java code should be inside JavaBeans and not on the JSP page; JavaBeans will give your JSPs their database access through JDBC or access to other distributed objects through CORBA or RMI. Utilizing JavaBeans from JSPs is the next logical step from using imported classes, as was the norm with JHTML pages in the Java Web Server 1.1. The Bean tag provides the

| Method | Explanation | Example |
|---|---|---|
| CreateObject(String classname) | Instantiates an object from the given classname. This method is here only to provide compatibility with the same ASP method. It is recommended that you use the JSP Bean tag to instantiate an object | <% Server.CreateObject("com.insource.beans.ecom"); %> |
| HTMLEncode(String string) | Applies HTML encoding to a specified string | <%=Server.HTMLEncode("The bold tag: <B>"); %> |
| URLEncode(String string) | Applies URL encoding, including escape characters, to the string | <% Response.write(Server.URLEncode ("http://www.javasoft.com")); %> |
| MapPath(String string) | Maps the specified relative or virtual path to the corresponding physical directory on the server | <% Server.MapPath("script/data.txt"); %> |

*Table 6: Callable methods of the Server Object*

| Method | Explanation | Example |
|---|---|---|
| getSessionID() | Returns the unique sesssion identification for this user | <% int I = Session.getSessionID(); %> |
| putValue(String name, Object object) | Stores the object with the given name in this session | <% Session.putValue("Name", "Value"); %> |
| getValue(String name) | Returns the object stored in this session with the given name | <% String ShoppingCart = Session.getValue("Name"); %> |
| removeValue() | Removes the object bound to the given name in the session | <% Session.removeValue("Name"); %> |

*Table 7: Available Session methods*

syntax that allows the JSP to refer to the bean, as follows.

```
< BEAN name="lookup name" varname="alter-
nate variable name" type="class  or inter-
face name" introspect="(yes/no)"
beanName="filename" create="(yes/no)"
scope="(request/session)" >
<PARAM Property="Value">
</BEAN>
```

Let's take a closer look at the attributes found in the Bean tag. The name attribute identifies the key by which the bean is looked up. The optional varname attribute identifies the variable name that will refer to the bean. The type attribute – again optional – specifies the name of the bean's class, or interface, and defaults to the Object type. The introspect attribute is also optional and is set to either yes or no; by default the introspect attribute is set to yes, which means that the appropriate property – setter methods from the bean's BeanInfo – is called for each matching property. The optional create attribute is also set to either yes or no and defaults to yes. When yes is set, the bean is created if not found in the specified scope, whereas an error returns if no is specified and the bean isn't found. The scope attribute is optional and is set to either request or session. Scope is set to request by default where the bean is retrieved from the request context. If scope is set to session, the bean is reused from the current session, if present. The beanName attribute is the name of the serialized file or class file that contains the bean and is used only when the bean is not present in the scope of the Bean tag and the value of create is yes. Finally, the PARAM tag can be optionally inserted inside the Bean tag to define values for a list of properties that will be set automatically through introspection. An example Bean tag would look like this:

```
<BEAN name="db" type="com.insource.beans.db"
scope="session">
<PARAM LastName="Flatt" FirstName="Darren">
</BEAN>
<%=db.LastName %>
```

Remember that this spec is layered on top of the Servlet API, so any code you've included with servlets can be included in your JSPs. As always, make sure the classes and beans you call from your JSPs are in your Classpath or they won't work. While objects like Request, Response and Out are included with the Sun JSP spec, you'll have to utilize the Servlet API to build full-blown Web applications. Somehow, the most important object for building cohesive Web applications, the Session object, has been left out of the spec. To rectify that problem you'll need to add this line of code on your Web page before any other session-related code:

```
<% HttpSession session = request.getSes-
sion(true); %>
```

With the addition of this code, you can add, retrieve and remove Session variables throughout your Web application, as follows:

```
<%
session.putValue("Item", "Hat");
//add a Hat to the Item session variable
session.getValue("Item");
//retrieves the value of the Item session
variable
session.removeValue("Item");
//removes the Item session variable
%>
```

This sums up the major features of the JSP spec as laid out by Sun Microsystems. With this spec and a little help from the Servlet API, you can build full-featured Web applications that run in the Java Web Server and any server supported by IBM's servlet engine.

The other main player in this JSP drama is Live Software. Their most notable product is a freely distributed servlet engine called JRun that allows users of most major Web servers to run Java Servlets. This has been a great deal for Web application developers who don't want to be tied to proprietary server APIs such as NSAPI or ISAPI, but want a performance boost over CGI. Of course, IBM also provides a servlet engine that runs on many Web servers and implements the Sun JSP spec, but Live Software has gone a step further by adding objects that will look familiar to Active Server Page developers. Since we've already covered the Sun spec, which also provides the basis of Live Software's JRun Server Pages, we'll move on to the Objects and features that JRun adds above and beyond what Sun provides.

Just like Sun's JSPs, JRun Server Pages are made up of Directives, Declarations,

| Event | Explanation | Example |
|---|---|---|
| Session_OnStart | Can access any object a "normal" JSP file can | <script runat="server" event="Session_OnStart"><br>Response.write("Hello This is a new Session<BR>");<br>System.out.println("Session_OnStart called for: "+Session.getId());<br></script> |
| Session_OnEnd | Can only access the Application, Session, and Server objects. The Server object cannot call the MapPath() method | <script runat="server" event="Session_OnEnd"><br>System.out.println("Session_OnEnd called for: "+Session.getId());<br></script> |
| Application_OnStart | Can only access the Application and Server objects | <script runat="server" event="Application_OnStart"><br>System.out.println("Application_OnStart called");<br>Application.setAttribute("greeting", "hi mom");<br></script> |
| Application_OnEnd | Can only access the Application and Server objects | <script runat="server" event="Application_OnEnd"><br>System.out.println("Application_OnEnd called");<br>Application.removeAttribute("greeting");<br></script> |

*Table 8: Syntax and restrictions on global.jsa events*

Scriptlets, Expressions and the Bean tag. JRun Server Pages Directives add server-side include functionality through the use of the include and vinclude variables. The include variable allows you to insert the contents of a file relative to the local file system, and the vinclude variable does the same thing relative to the Web document root.

More important, JRun Server Pages add ASP Compatibility Objects to ease the migration of Active Server Pages Web applications to JSP. The Application object allows you to share information among all the users of a Web application, as defined by the JSP files in a virtual directory, as well as its subdirectories. The syntax for using this object and its associated methods appears in Table 3.

The Request Object retrieves the data passed to the server by the Web browser. With the Request Object you can use ASP Compatibility methods as well as all other methods made available in the HttpServletRequest class. Any variable may be accessed by calling Request(String name), whether it comes from a Form, QueryString, Cookie or Server Variable. If a variable with the same name exists in more than one collection, Request will return the first instance it encounters. You can be more specific by calling one of the Request methods in Table 4.

The Response object sends data to the client Web browser. As with the Request object, you can use ASP Compatibility methods in addition to all other methods made available in the HttpServletResponse class. JRun Server Page Response methods include those in Table 5.

The Server Object provides access to methods and properties on the server; most of them serve as utility functions. Callable methods of this object are found in Table 6.

The Session Object allows you to store information throughout the life of a particular user session. Variables stored in the Session Object persist, as the user jumps from page to page in your Web application. A Session Object is created when a user who doesn't already have a session requests a Web page. The server destroys the Session Object when the session expires due to inactivity or abandonment in code. (It should be noted that Session state is maintained only for browsers that support cookies.) Available Session methods are shown in Table 7.

The final ASP-related object in the JRun Server Pages is the global.jsa file; this file mirrors the global.asa file found in Active Server Pages and performs the same functions. Your JSP application can have only one global.jsa file, which must then be located in the root directory of the application. JSP reads the global.jsa file when the Web server receives the first poststart-up request for any .jsp file in an application, or when a user who doesn't have a session requests a .jsp file in an application. You can include Application start/end events, as well as Session start/end events, in your global.jsa file. The syntax and restrictions on global.jsa events are described in Table 8.

Live Software's JSPs also include a few other features that further reduce the time it takes to build Web applications. Taglets allow programmers to define their own custom HTML tags that perform a certain function. A Java programmer can define a reusable, server-side Taglet that will execute Java code to do anything you can do using Java. A nonprogramming Web page designer can then use that Taglet in his or her Web pages without having to understand the complexities of the Java code it's executing. This method of creating custom HTML tags goes a long way toward separating the Web presentation from the underlying business logic. Last, Live Software goes beyond merely allowing you to communicate with JavaBeans from your Web page; it adds several useful beans to get you started: a File bean, an HTTP bean, an SMTP mail bean, a Shopping Cart bean, and Database Connectivity beans with built-in connection pooling.

Clearly, both Sun's and Live Software's approaches to building JSPs go further than any other tool currently used to build sophisticated Web applications. The spectacular thing is that this is accomplished without tying the Web developer to a particular Web server or operating system. Sun should be applauded for identifying the fact that not everyone wants to build servlets from scratch. JSPs allow novice Web programmers to learn Java a little bit at a time, using their favorite HTML editor. The JSP/Bean model of development allows teams of Web page designers and artists to do what they do best, while the Java programmers develop JavaBeans that interface with databases, distributed objects and other systems.

The full suite of Enterprise Java APIs is available for use with JSPs. Java Server Pages, as defined by Sun and IBM, are a great step forward from JHTML Dynamic Page Compilation in terms of ease of programming. Live Software has taken the torch and run even farther with its JSP implementation. Its ASP Compatibility Objects, Taglets and beans dramatically reduce the amount of coding a Web developer has to perform to get a powerful Web application out the door quickly. The productivity gains achieved by using JRun Server Pages are rivaled only by Microsoft's Active Server Pages. Of course, your JSP application won't be tied to Internet Information Server; it will be portable to any Web server you desire. That's why we use Java.


**About the Author**

*Robert Tiffany is a senior technology consultant with Insource Technology in Houston, Texas. He is working on an e-commerce Web site using servlet technology for the George Bush Presidential Library. Robert has worked on Internet/intranet/extranet development with both Active Server Pages and Enterprise Java technologies. You can reach him at robertt@insource.com.*

robertt@insource.com


## Optimizeit 3.0 Professional Ships

(Sunnyvale, CA) – Intuitive Systems, Inc., is shipping Optimizeit 3.0 Professional, the Java technology-based performance tool that allows developers to test and improve the performance of most Java applications, applets, servlets and JavaBeans.

The new product includes full support for Sun Microsystems' Java Development Kit (JDK version 1.2). It is also easily integrated with development environments JBuilder from Inprise Corp. and VisualCafé from Symantec, Inc., allowing developers to directly profile their Java programs from their development environment of choice.

Optimizeit Professional 3.0 is available at $449 for Windows 95 and 98 platforms, and will soon be available for Sun's Solaris operating environment.

For additional information, call 408 245-8540 or visit www.optimizeit.com.

## Objective Toolkit for ATL from Rogue Wave

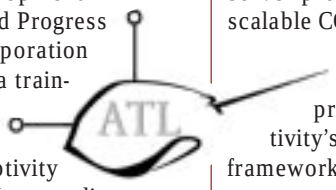
(Research Triangle Park, NC) – Rogue Wave Software, Inc., has added to its Stingray product line with Objective Toolkit for ATL, which extends Microsoft's Active Template Library (ATL). ATL enables C++ developers to more easily develop reusable COM objects. Objective Toolkit for ATL enhances that ability by maximizing code reuse through GUI, COM and productivity enhancements in familiar ATL-like implementations, increasing the ability to create solid, feature-rich components for the enterprise.

Objective Toolkit for ATL is available this quarter for $995 in North America. This includes full source code, 60 days of technical support and a 30-day money-back guarantee. Annual support licenses are available for $495 in North America and include a full year of technical support as well as product updates.

For more information call 800 487-3217, e-mail sales@roguewave.com or visit the company Web site at www.roguewave.com.

## JHL Computer Consultants Sign Agreement with Progress Software

(Fort Lauderdale, FL) – JHL Computer Consultants, a training and development company, and Progress Software Corporation have signed a training agreement for Progress Apptivity version 3, a Java application server with an integrated development. Apptivity enables IT and ISV organizations to rapidly deliver new and enhanced applications for intranets, extranets and the Internet. JHL will also provide development services in Apptivity.




The Apptivity application server provides a secure and scalable CORBA-based server architecture that supports Enterprise JavaBeans. Apptivity's SmartAdapter framework allows applications to access external data sources through a standard data interface model.

For details call Progress Software at 800 477-6473 or visit www.progress.com.

## KL Group JProbe Adds Memory Debugger to Performance Profiler

(Toronto, Ont.) – KL Group Inc., a provider of Java components and advanced development tools, will ship its advanced Java performance profiler and memory debugger, JProbe 2.0, this quarter. The profiling tool makes it easy to identify and eliminate performance bottlenecks and memory leaks in Java code, and provides heap analysis tools that help find memory leaks to reduce development time and improve code quality. JProbe profiles applications written in JDK 1.1 or Java 2 software for Windows and Solaris.

JProbe technology leverages the Java Virtual Machine to capture all objects created and any method calls performed by the Java code being profiled. The profiler reports application performance on a per-method or per-line basis, speeding the process of performance tuning. The memory debugger accuracy combined with graphical analysis tools makes a powerful and easy-to-use Java performance profiling and exploration tool.

JProbe 2.0 will be generally available this quarter. The profiler will start at $499 for a single developer license. JProbe Profiler and KL Group's 100% Pure JClass JavaBeans are available from qualified resellers and their Web site at www.klgroup.com.

JPROBE Profiler

# Novera Announces jBusiness for SilverStream

(Burlington, MA) – Novera Software Inc. has integrated jBusiness 4 with the SilverStream application platform. Customers can now extend the Novera and SilverStream environments by integrating Novera's Enterprise Business Objects into SilverStream solutions.

jBusiness is an application and management framework that allows customers to create Enterprise Business Objects as the foundation for distributed enterprise applications. These objects encapsulate enterprise data from legacy systems into reusable software objects based on CORBA and Enterprise JavaBeans standards. The jBusiness Management Server also provides end-to-end management of applications built with SilverStream that access Enterprise Business Objects, including configuration, security and monitoring.

For more information call 888 NOVERA1 or visit www.novera.com.

# *Oracle Expands Relationship with Inprise Corporation*

(Scotts Valley, CA.) – Inprise Corporation, a provider of enterprise integration software and services, has signed a worldwide, multimillion-dollar licensing agreement with Oracle Corporation. Under the terms of the multiyear agreement, Oracle has selected Inprise's VisiBroker as one of its worldwide standards for CORBA object request broker (ORB) technology. To date, VisiBroker has been integrated into Oracle8*i*, Oracle Application Server and other Oracle products.

For more information visit www.inprise.com.


THE GRIND

*by* **Java George**

# Java – Into Its 4th Year

In the final months of 1998 it was worth reflecting on the networked application platform and its star player, Java, as we headed into the new year. Many of you have written and put forward your own assessment of the Java movement as it rode into 1999, representing the fourth year of Java as a commercially available development platform.

This is the first installment of our theme "Java – Into Its 4th Year," and we will review the platform independence promise of the Java platform: run anywhere, anytime. It's not an all or nothing proposition.

Never has more hype surrounded a basic notion of platform independence as Java's "run anywhere, anytime" promise. This was particularly acute with Java on the client side (in browsers). Unfortunately, Sun's marketing messages were taken completely out of context and Java developers assumed that any piece of compiled Java code would magically run not only on all virtual machines and platforms, but in all circumstances as well. Why would any experienced developer fall into such a predicament?

The marketing message should probably have been "Java, run anywhere, anytime – with the appropriate amount of effort!" It's certainly possible to create a Java client that runs across the vast majority of possible client configurations. Creating a Java client that runs across all possible client configurations is not significant, that is, it's not necessary to cover all permutations to achieve the desired effect. In other words, if out of 100,000 end users 10 can't access your Java client because they're running OS/2 and the Navigator 2.0 browser, does that constitute a failure in deployment? I should say not.

A well-engineered Java client is extremely successful against a wide cross-section of possible client configurations easily covering 90% of possible end users. Witness the Yahoo! game site and its multiplayer Java games: chess, backgammon and checkers. The Yahoo! games constituency runs across Macintosh, Windows and Solaris Workstations, and various versions of browsers across each. Witness the PROGRESS Apptivity 3.0 Java client that supports Netscape and the IE 3.x and 4.x browsers across all major OS platforms. "Support" doesn't translate into "works every time without any effort on your part"; it translates into "will be portable with a reasonable amount of effort by the developer and the vendor."

On the server side, Java platform independence has received additional scrutiny. While it's clear that the UI layer presented the toughest platform challenges, the server side is not immune from slight discrepancies in the JVM. The most important thing to remember is that server-side processes, particularly those oriented at high-transaction business functions, are very reliant on the robustness of the JVM implementation. A server process with 1,200 threads and 1,500 object instantiations is no angel; it's significantly dependent on its Java container. Such a beast running on JVM 1.0.2 would most certainly come to a crashing halt, while under JVM 1.2 it would most likely hold its own. It would perform even better if such a complex process was abstracted on top of a competent CORBA infrastructure.

Once again, this shows the importance of layering the Java solution and insulating the developer from the challenges faced in achieving platform independence and performance. On the client side, the developer using a proven, supported Java client architecture will achieve success far and above a counterpart creating the client from scratch. At the very least, the Java client should take advantage of a foundation class such as JFC/Swing, AFC or Netscape's IFC. On the server side, the developer building on top of JDK 1.2, Corba and EJB will get to successful deployment and scalability while the developer building from scratch will most likely never achieve deployment at any reasonable scale!

Java is heading into its fourth full year as a legitimate language and platform, and with it, developers can achieve platform independence with a reasonable effort. How much effort will be required to port an ActiveX control across Unix, Windows and Mac platforms? How much effort will be required to port a significant C or C++ executable across these platforms?

Platform independence is not an all or nothing proposition. It's a question of practicality, and a question of reasonable effort leading to significant value!

*"A well-engineered Java client is extremely successful against a wide cross-section of possible client configurations..."*

*Java George is George Kassabgi, director of developer relations for Progress Software's Apptivity Product Unit. You can e-mail him at george@apptivity.com.*

george@apptivity.com