JDJ APRIL SNEAK PREVIEW

JAVA™ DEVELOPER'S JOURNAL

JavaDevelopersJournal.com

Volume: 4 Issue: 4, 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 K. 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

???, Gene Callahan, Daniel Dee, Robert Dodson, Claude Duguay, Stephen Gould, Niraj Jetly, Panos Kougiouris, Pascal Ledru, Balaji Natarajan, Shrideep Pallickara, Sean Rhody, Rick Ross, Mark Steenbarger, Alan Williamson, Ed Zebrowski

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
Accounting Manager: Ignacio Arellano
Circulation Manager. Mary Ann McBride
Advertising Account Manager Robyn Forma
Advertising Assistant Megan Ring
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

What's the Code?

I remember the first time I saw *Jurassic Park* and watched as the little girl proceeded to hack into a UNIX system, quickly taking control of the entire park. I couldn't help but laugh at how unlikely that whole scenario was, but it does serve to illustrate the way many people think of programming. With this month's focus on code and things related, I thought it would be apropos to discuss what we do for a living.

Unlike the Hollywood stereotype, no programmer, however good, can simply take control of a system. I'm sorry, but even if Jeff Goldblum was the greatest programmer in the world, I doubt he could create a virus to crash the alien mothership. Yes, I've switched movies, but stayed on my theme, though it may be hard to tell since Jeff was in both. I guess to the movie crowd, Jeff Goldblum is everygeek – the nerdy computer guy in all of us.

Sorry, it just doesn't happen that way. You know it. I know it. But the long grind and late hours that it takes to make a good program great just don't make for good movies. Imagine if we had a large-scale development team called in to write the virus for the mothership – we'd all be fertilizer. That's because there's more to coding than code, and there's more to development than development.

Good coders aren't born, they're made. It takes education and practice to make a good coder, as well as skill in logic and reasoning. Education provides the basic concepts and theories; practice hones their use. It also helps if they have good tools.

I recently had the opportunity to speak with KL Group concerning their JProbe product. JProbe is a thread and memory debugger and profiler for Java. It's one of those tools that can help a good programmer become a better one. Most of our IDEs have built-in debuggers, and the use of these tools should be encouraged. But typically it's hard to see the flow of a program, particularly a multithreaded program, within a debugger. JProbe provides another level of debugging help that will allow you to see what's happening inside the JVM as a whole.

Other tools and utilities are also useful for coding. Each IDE product does different things well. Some are better at two-way coding, some concentrate on integrating distributed computing models and some concentrate on being the best code editor there is. Each fits a particular style of coding. Some coders are devoted to writing every line of code themselves. Other coders prefer to have a tool do as much work as possible. Put these people together in a room and you can get some interesting debates. I recently gave a presentation concerning the project I'm on, and even though we've already selected and used one IDE for over six months, one of the attendees wanted to know if we'd considered the IDE he uses. That's one of the funny things about coders – they're very territorial.

I also think that good, focused subject matter training is essential to being a good programmer. We're Java programmers, right? A good background in object-oriented programming is essential, but there's still a strong need to get familiar with the specifics of Java, such as AWT and Swing. Work in a three-tiered system and you need to understand RMI (or IIOP) and JNDI, as well as the way an application server works. Layer on top of that a particular development framework (homegrown or purchased) and you should be able to make a good case for training for anyone who joins the project.

I hope you enjoy this issue's focus on code. While you're reading this, I'm going to be reconfiguring an old Cray to run Linux so I can access the mothership and get them to let me fly one of those cool fighter planes. May The Force be with you.

About the Author

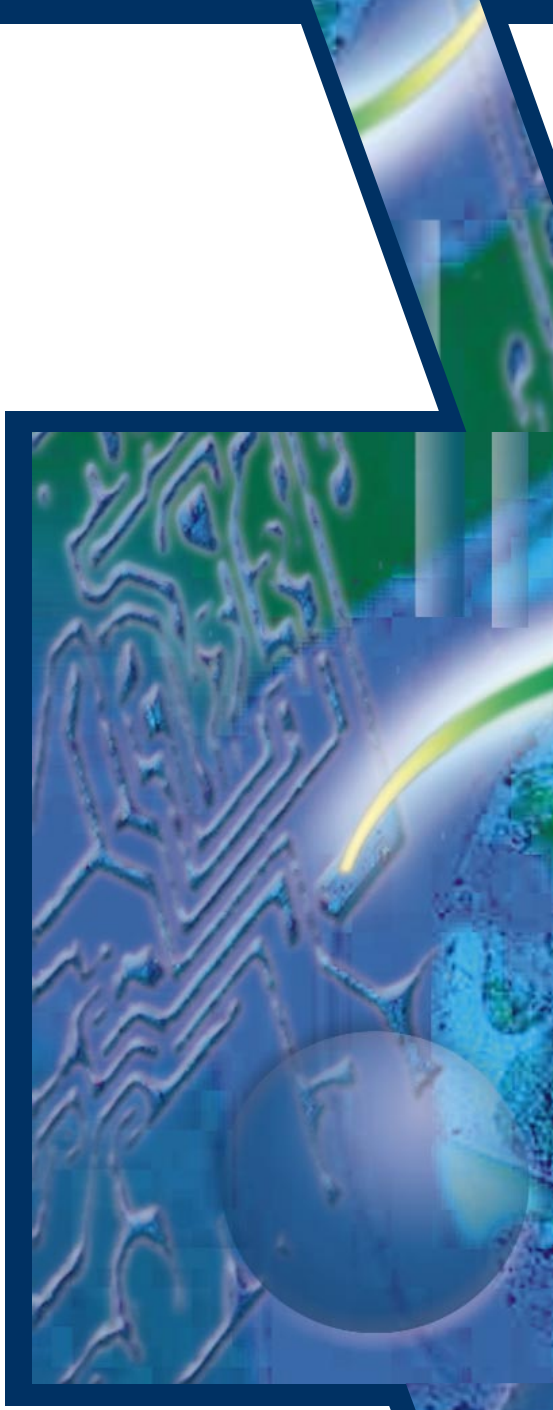
Sean Rhody is the editor-in-chief of **Java Developer's Journal**. *He is also a senior consultant with Computer Sciences Corporation, where he specializes in application architecture – particularly distributed systems. He can be reached by e-mail at sean@sys-con.com.*


puter
ciates
ds/jasmine/dev

JSDA, CORBA and HLA-Based Mutechs

Distributed computing technologies for building multiuser scalable televirtual environments on the Internet

by **Balaji Natarajan & Shrideep Pallickara**



Java scripting support in VRML2/VRML97 set the stage for experimenting with multiuser distributed virtual environments on the Internet, hereafter referred to as televirtual, or TVR, environments. A typical minimal configuration of such a system would include a few VRML2 browsers, downloading a common VRML world and opening Java node-based connections to a collaborative server that maps user input such as mouse motions on suitable movements of the corresponding avatars.

We developed a simple prototype TVR environment of this type at Syracuse University's Northeast Parallel Architectures Center (NPAC), as part of a joint project with IBM T.J. Watson, using the JSDA (Java Shared Data Architecture) framework for building Java collaborative services.

Several prototype TVR environments of a similar type have been developed by groups such as Sony, Paragraph (Mitra), BlackSun and MERL, and a set of VRML SIGs was formed, including Universal Avatars, Humanoid Animation and Living Worlds, which focused on standardizing various aspects and software layers of VRML-based networked VR.

The detailed architecture of collaborative servers has not been addressed directly by the VRML community. For example, Living Worlds encapsulates various multiuser technologies in terms of a Mutech (multiuser technology) node and focuses on its interactions with local/client-side VRML nodes in the scene graph. There are some associated ongoing standards efforts in the Mutech domain. For example, Open Community led by Mitsubishi Electric Research Labs (MERL) released an open standards proposal.

In the VRML community framework, we can express our work and the content of this article as research into promising Mutech technologies based on stable open standards and capable of enabling or facilitating the design or development of truly scalable TVR environments.

We are exploring the following collaborative server technologies of relevance for TVR within the ongoing R&D activities at NPAC:

- JSDA from JavaSoft
- CORBA objects and Event Services
- HLA/RTI (High Level Architecture/Runtime Infrastructure) by DMSO (Defense Modeling and Simulation Office)

In this article we examine the CORBA domain and discuss the relationship between VRML and the emergent distributed object technologies.

TVR Front-End Description

Our current TVR prototype has two versions of the VRML+Java front end: one is based on Script Nodes; the other, on the External Authoring Interface (EAI).

The EAI version of the prototype was tested on SGI's CosmoPlayer (version 1.0.2), running as a plug-in to the Netscape 3.0 Web browser on a PC platform. The Script Node version of the prototype was tested on Sony's Community Place, running as a plug-in to the Netscape 3.0 Web browser, and also SGI's CosmoPlayer (1.0 beta 3a), running as a plug-in to Netscape 3.0.

Our current "world" metaphor is given by a set of rooms with avatars represented by simple geometrical objects (such as colored cones). We've added realism in terms of more humanlike avatars with custom behaviors, conforming to the specifications of the Humanoid Working Group Proposal. We're also exploring add-on audio-conferencing capability using commodity APIs like Microsoft NetMeeting.

TVR Back-End Description

JSDA provides a shared framework for Java at the data level. Data objects are shared over specific instances of channels (broadcast communication paths) between two or more clients (objects that are the source or destination of data) in a collaborative environment. Any client object that needs to register its interest in receiving messages sent over a channel must implement the Channel Consumer. Similarly, if a client is interested in being notified about changes in the state of some other object, it should implement the Channel Observer interface. To register interest in a certain channel, a client first needs to join the session that hosts this channel and then can join the channel.

JSDA allows objects to be shared using object serialization mechanisms since it has RMI-based implementation. JSDA also has objects that encapsulate management policies for application objects. One example is the Session Manager, which authenticates clients to determine whether they can join a session. Currently, JSDA is a research-oriented API at JavaSoft Corporation and our TVR prototype is being packaged as an effective demonstration of JSDA's capabilities, along with the main distribution.

Toward Multiserver JSDA Environments

Figure 1 illustrates a more complex TVR world (currently under development at NPAC) that includes *n* avatars in *m* rooms where both *n* and *m* can be large (Internet clubs, malls, etc.). Rooms are mapped to sessions (1, 2, etc.) running on individual servers. Each room/session publishes local sensory channels used to exchange coordinate/visual information between avatars in this room. Some rooms can also publish long-range channels (e.g., audio) that are accessible from other rooms.

Figure 1 shows an avatar moving from room 1 to room 2. It detaches from the room 1 visual/sensory channel and attaches to the room 2 visual/sensory channel and retains the radio channel to listen to news/ads broadcast from room 1. JSDA sessions are mapped on "rooms" and JSDA channels are assigned to individual avatars present in a given room. Only a limited number of avatars are allowed per room, and the number of sessions per collaborative server is also limited. This simple model assures worldwide scalability, assuming that new rooms join with their own session servers and that most interactions are local.

Toward CORBA-Based Collaborative Environments

JSDA is a useful framework for prototyping simple collaborative applications, but it doesn't offer either a wire protocol or a high-level API for client/server communication; messages are typically passed as strings, custom encoded/decoded by JSDA clients/servers. The family of T12x protocols (which in fact influenced the JSDA design and was adopted by Microsoft's NetMeeting) could be a natural candidate for a TVR protocol. Another possibility is that such a protocol would be developed in the course of current interactions between MPEG-4 (Moving Picture Experts Group) and VRML streaming groups. However, a tempting alternative would be to select one universal wire protocol for the Internet that would be capable of supporting all required communication patterns in all relevant application domains.

At the moment, the most promising candidate for such a lingua franca on the Web is the Internet interoperability protocol (IIOP) by OMG that enables interoperation between ORBs from various vendors and is also frequently used as internal inter-ORB protocol between clients and servers within a single-vendor CORBA environment. In the 100% Pure Java sector, similar support is offered by RMI (in fact, it is supported as one of the JSDA implementation modes), whereas CORBA offers both multiplatform and multilanguage support in terms of the universal IDL interfaces/mappings and language-specific bindings. With the onset of

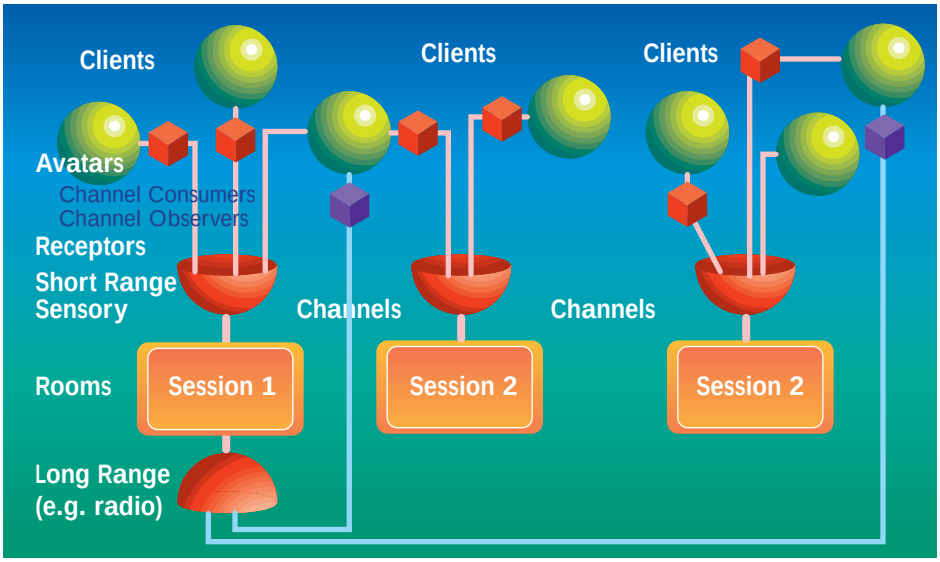


Figure 1: JSDA-based TVR

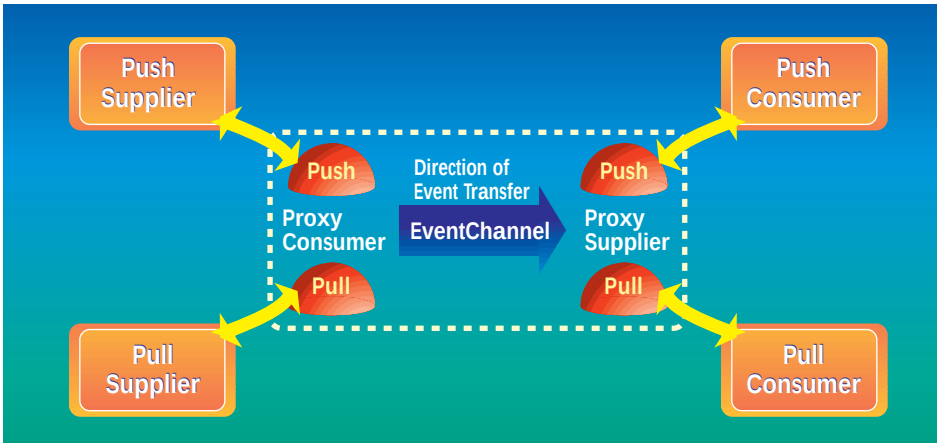


Figure 2: CORBA Event Service

ORBlets, dynamically downloadable or resident in Web browsers such as those supported by Netscape/Visigenic, CORBA now gets even more tightly integrated with Java toward a new, powerful computing paradigm often referred to as Object Web.

To operate in today's heterogeneous computing environments, distributed applications must work on a plethora of hardware and software platforms. Suitability to business class applications calls for capabilities beyond conventional Web-based computing – scalability, high availability, performance and data integrity. This is where Java and CORBA complement each other: Java provides for easier distribution of CORBA-based applications that have the wherewithal of a distributed infrastructure.

Java–CORBA Combination

Java's multithreading support encouraged developers to write Web-based distributed software based on proprietary server protocols. Each such server can be viewed as a specific remote computational object. On the other hand, CORBA offers generic support for such server objects based on distributed object technology. Instead of encoding low-level messages, sending them through the network and decoding them at the receiver side, the programmer just calls an appropriate high-level method on a distributed object without the need for any specific low-level network programming. This high-level abstraction capability is definitely a promising framework for future distributed solutions. The only two alternatives that can be viewed as competitive are Java RMI and Microsoft DCOM – but only CORBA is both language- and platform-independent.

However, rather than a competitor or alternative to CORBA, Java is now viewed by many as a complementary technology that forms a perfect match with CORBA within the emergent "Object Web" trends. In a nutshell, the master plan of the Object Web (supported by Netscape, Oracle, IBM, Sun and others) is to implement CORBA control; that is, the middleware layer (including ORBs and some core services) in Java.

Since Java has an inverse mapping to IDL, a programmer can stay in the Java environment during the software development. Java–CORBA implementations can run on thin network computers and low-end consumer devices because of their low complexity and footprint. Java's mobile byte code and CORBA's Dynamic Invocation Interface (DII) simplify upgrades of clients' software in large distributed systems. The Java and CORBA combination truly provide the right building blocks for distributed object computing: (1) platform independence, a strong security model, etc., in the Java language; and (2) static and dynamic interfaces, and synchronous and asynchronous method calls with the comprehensive set of facilities and services in CORBA.

These factors might result in the CORBA–Java combination's assuming a central role in shaping the Internet during the next phase of its evolution. Such emergent Object Webs could have impact in several areas, including multiuser collaborative environments.

In particular, the collaborative environments can be naturally addressed by CORBA in terms of the Event Service – one of the standard 15 services developed and sustained by OMG (together with Security, Persistence, Concurrency, Naming, LifeCycle, Relationships, Trading and other such fundamental object services). The following sections describe the CORBA Event Service, which offers functionality similar to the JSDA Channel discussed above.

CORBA Event Service

The Event Service (ES) (see Figure 2) allows for decoupled communication between objects. Instead of a client's directly invoking operation on a target object, it can send an event that can be received by any number of objects. The sender of an event is called a supplier, and the receivers are called consumers. Suppliers and consumers are decoupled; that is, they don't know each other's identities.

The ES introduces the notion of Event Channel. Suppliers send events to an Event Channel and consumers receive these events. Each channel can have multiple consumers and suppliers, and all events sent by a supplier are made available to all consumers of that channel.

The ES supports four different modes of consumer–supplier interactions. The consumer could be a push/pull Consumer, and the supplier could be a pull/push Supplier. One advantage of using the Event Channel is that the events can be buffered to accommodate consumers of differing speeds. Suppliers and consumers both register with the Event Channel; otherwise it isn't possible to determine the source of the event should the supplier not be able to invoke the appropriate notification method onto the consumer.

Supplier objects request an appropriate

ProxyConsumer object from the Event Channel's ConsumerAdmin object. Whenever the supplier wants to send an object to the Event Channel, it uses the corresponding ProxyConsumer object. Similarly, consumer objects ask an appropriate ProxySupplier object from the Event Channel's SupplierAdmin object. Whenever a channel receives an event, it informs all the ProxySupplier objects. Each proxy object then notifies its consumer.

CORBA-Based Collaborations

So far, in our work on CORBA-based collaborations, we have developed an initial API for CORBA-based collaboration. The IDL definition of this API is given in the following section. A prototype version of this API, using CORBA objects as "shared data," and CORBA servers as "collaborative servers" and Netscape/Visigenic-based ORBlet front ends, has been developed at IBM.

We are currently extending this design and preparing a refined implementation using CORBA Event Service, which plays an event-filtering role similar to that of the JSDA Channels (see Figure 3).

These are the two most significant IDL definitions in the Collaborative System. The IDL definitions signify the operations a client can invoke on a remote instance of these objects. Nevertheless, invocation of any of these previously mentioned operations should be preceded by the successful reception of a remote handle to these objects. Acquisition of a handle to the PartyCoordinator requires the client to invoke a bind to that object. To digress on the semantics of the bind, it should be clear that in a high-availability scenario there would be multiple instances of the PartyScheduler with a static hashtable containing the list of updated Parties; that is, Coordinator Objects (see Listing 1).

Elucidating further on the semantics of operations on these remote objects, the PartyScheduler is the one that schedules the appropriate instance of the Coordinator Object to coordinate clients logged onto a specific session (Party) composed of different possible applications. Basically, the PartyScheduler is responsible for spawning instances of the Coordinator, possibly across a different subnet, and also for returning a remote Coordinator handle to the client. A brief description of the sequence of operations in the Collaborative System follows.

The client initiates a bind to the PartyScheduler Object. If this is successful, and if there is a Distributed Directory service and the Active Object server is in place, the client is now ready to invoke the IDL-defined operations.

It starts with the createParty(String partyName) function, which would return a true in the event that a new Coordinator Object has been instantiated or a false to signify the prior existence of the desired party. It is the Scheduler's job to signal the appropriate notification to the clients and perform appropriate housekeeping to reflect new instances of Coordinators. All Coordinator objects scheduled by the PartyScheduler are identified by an ID.

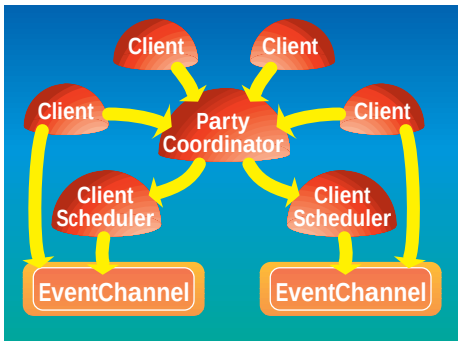


Figure 3: Event Service-based Collaboration Framework

The client now has the option of deciding whether to join an existing Party or initiate a new one. In the latter case, the first step is repeated, as mentioned earlier. Once the process is over, the client gets a handle to the Coordinator Object by invoking long getPartyID(in string arg0); MultiCoordinator::Coordinator getPartyHandle(in long arg0); in succession. This is in keeping with the policy of the PartyScheduler to identify Coordinators on the basis of the ID that it assigns during their instantiation.

Once the first two steps are over and done with, the client in a Distributed Collaboration mode can invoke operations specified in IDL definitions for the Coordinator. These include Boolean broadcast (in string arg0) and Boolean whisper (in string arg0, in long arg1); among other functionalities offered by the PartyCoordinator Object.

This is just another demonstration of the complementary roles Java and CORBA can play in distributed environments. Java provides for easier distribution of CORBA-based applications, with CORBA providing what is necessary for a distributed infrastructure. To summarize, CORBA offers both a potential candidate for universal wire protocol – IIOP – and a natural collaborative framework based on shared CORBA objects and flexible message-filtering mechanisms offered by the Event Service.

Scalability and Fault Tolerance in Collaborative Systems

Scalability and fault tolerance, the essential features required in Collaborative Systems, can be naturally addressed in the CORBA model. The replication of servers on different participating hosts in a collaborative environment answers the scalability problem when the load (participating sessions) on a server crosses a certain threshold. The current API supports two different approaches. The notion of groups within a certain session allows us to define one object for each group and easily place these objects on different machines. It is also possible to split the Event Channel if it exceeds a certain capacity and connect two Event Channels to each other as supplier and consumer since one Event Channel can be a consumer/supplier of another Event Channel.

Fault tolerance in Collaborative Systems can be solved by migrating sessions to a different participating host with minimal or little disruption whenever the machine hosting the server crashes. The ObjectServices agent, which is a Distributed Directory service, allows for migration of sessions to a different participating host in case a session terminates unexpectedly on one of the hosts. The events can be stored persistently by the Event Channel to ensure that they are not lost because of system failures.

Using CORBA Event Service for Message Broadcasting

At NPAC we implemented the Event Service for omniORB2, a free C++ Object Request Broker under development by Olivetti and Oracle Research Labs. We wrote the standard Event Service with C++ and omniThread thread library. We tested this software with a chat program using Netscape 3.0 with OrbixWeb-based ORBlets.

We also started exploring the issues related to using CORBA Event Service for Distributed Interactive Simulation (DIS) PDU broadcasting and for HLA/RTI support. Some additional services are required to support event handling in a multiuser environment. For example, Event Service doesn't care about the originator of the event. But for a multiuser environment, we need to know who sent the message. DIS PDU format includes this information in its own body. The ES is a central server-based approach compared to peer-to-peer architectures. We can solve this problem by providing an Event Channel for each active object in the virtual environment.

For large-scale interactive simulations, it is imperative that we support event filtering to adjust the frequency of events received from the supplier. The obvious choice is to make this decision a responsibility of Event Channel so the messages are handled before they are put on the network. Event filtering could be based on time stamping. However, this simple solution has a profound problem with synchronizing time values in distributed simulations. The best-known solution is to add the

Global Virtual Time (GVT) calculation to the system so that messages that are out of order can be handled properly with the rollback mechanism. GVT calculation allows us to release the storage for the logged events since nobody expects to receive an event timestamped earlier than the current GVT.

Another intriguing option for message transfer is to use multicasting. This requires some changes in the ES implementation. For PushConsumer, instead of giving a separate PushSupplier for each consumer, it is possible to give one PushSupplier for each multicast group so that multicast PushSupplier can serve multiple consumers. This change also reduces the computation requirement on the Event Channel server.

Several advanced event-handling features are currently in the OMG standardization pipeline in the form of the CORBA Notification Service. New capabilities include support for Quality of Service, integration with Transaction and Security Services and a more flexible user-adjustable format for event objects.

Emerging Collaborative Server Technologies Based on Distributed Object Technology

Experiments with Java and CORBA-based collaborations previously described represent our first initial steps toward systematic Object Web support for HLA-based modeling and simulations.

HLA is a next-generation framework for distributed simulation systems promoted by DMSO to replace the current DIS standard. HLA's enabling middleware (RTI) is based on distributed object technologies. DMSO is promoting HLA/RTI within the OMG toward a vertical CORBA facility for interactive modeling and simulation.

At NPAC we have been working with the DOD's High Performance Modernization program to integrate advanced Web-commodity technologies with large-scale Forces Modeling and Simulation systems being converted to or already based on HLA by DMSO and the enabling RTI middleware.

As part of this project, we are building an Object Web-based implementation of IIOP and HTTP server called JWORB (Java Web Object Request Broker) and the Object Web-based RTI layer that will operate on top of JWORB to provide Web-based simulation support for HLA and a natural linkage to front-end technologies such as VRML. For large, geographically distributed M&S systems, middleware must be given by a mesh of scalable collaborative servers running on heterogeneous platforms and supporting specific simulation components written in various languages. A Java–CORBA-based RTI middleware such as JWORB with a VRML front end seems to offer an attractive pervasive architecture for such systems.

Of particular interest within the DOD M&S systems are the simulation-based acquisition or virtual prototyping environments in which new systems are engineered and tested in virtual space before the first real prototype is manufactured.

Figure 4 illustrates a sample of such a system with JWORB-based middleware and a collection of front ends, including XML-based data analysis, Java-based data flow visual authoring software and VRML-based visual 3D display. Each of these activities can be made collaborative via the base RTI mechanism, CORBA Event Service or JSDA, and they can all cooperate via the JWORB-based componentware.

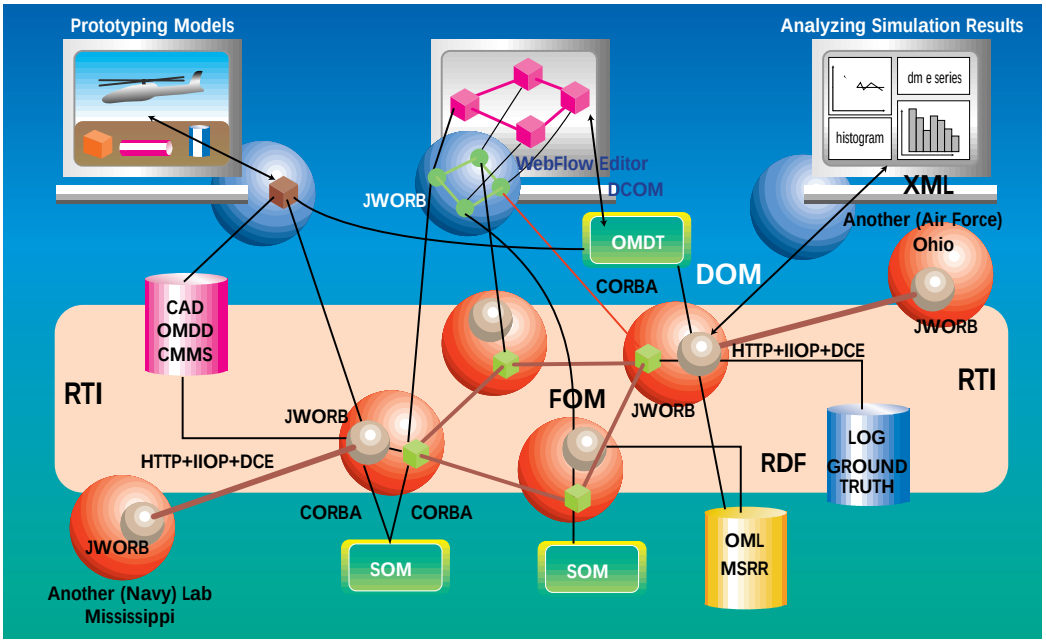


Figure 4: Framework for virtual prototyping environment

Summary

We've discussed here a set of promising distributed computing frameworks – JSDA, CORBA, RTI – that offer open standards based on support for building scalable multiuser virtual environments on the Internet. The essential feature of such an environment – communication locality – is enabled via event filtering in terms of JSDA channels, CORBA Event Service and RTI routing spaces.

So far, we've acquired a few early prototyping experiences using JSDA and CORBA technologies, and are now exploring the HLA/RTI environment. In the JWORB middleware framework currently under development, we'll be able to integrate, experiment with and conduct comparative analysis of all three collaborative technologies discussed here: JSDA, CORBA and HLA/RTI.

References

Syracuse University/IBM Technical Report on Prototype for Scalable TeleVirtual Environments for the Web.

D. Harkey and R. Orfali (1997). Client/Server Programming in Java and CORBA. Wiley.

S. Baker (1997). CORBA Distributed Objects, Addison-Wesley/ACM Press.

J.O. Calvin and R. Weatherly (1996). "An Introduction to the High Level Architecture(HLA) Runtime Infrastructure (RTI)." March, 14th DIS Workshop, 96-14-103.

D.C. Miller (1996). "The DOD High Level Architecture and the Next Generation of DIS." March, 14th DIS Workshop, 96-14-115.

K.L. Morse (1996). "Interest Management in Large-Scale Distributed Simulations." University of California, Irvine. Information and Computer Science Technical Report, ICS-TR-96-27.

About the Authors

Balaji Natarajan has a master's degree in computer engineering from Syracuse University. He is currently a graduate research assistant at Northeast Parallel Architectures Center (NPAC) at Syracuse University in New York. You can e-mail Balaji at balaji@npac.syr.edu.

Shrideep Pallickara is a Ph.D. candidate in computer engineering at Syracuse University and a graduate research assistant at NPAC. You can e-mail Shrideep at shrideep@npac.syr.edu.

balaji@npac.syr.edu shrideep@npac.syr.edu

Today Is a Good Day

Giving thanks for Java

by **Alan Williamson**

Just in case you don't know, I love Java. This month was a good month for loving Java. Some months, I have to confess, one does curse the little guy, but this month he was standing tall. Nothing of particular note happened in the media world, but it was something we did that made us sit back and be glad we chose this crazy Java world to try and make some pennies. But more on that later...I have a column to write.

How Are We All?

Are we doing well? I do hope so. It's springtime and the wee flowers are just rolling over from their winter nap, looking forward to a joyous summer. Isn't spring a wonderful time of year? We're coming out of winter and we can look back and see what riches the colder months have brought us, and have a guess what the warmer months are going to be offering.

This column is the last roadstop of controversy on the superinformation highway. The purpose of this column is to make you think. I'm not out to change your views, or even push my opinions on you. What I write you probably won't agree with, and this is good. This stimulates the brain and gets people talking. I want to unite people and get us all talking about various different topics. 'Tis why I drop a little variance into the column every so often. I want to anger some of you. I want to hear your views on subjects and if I have to push your morality a little to do so, then so be it.

Each month, you fill my inbox with e-mail and this is wonderful. I love debating issues with you all and on the way I learn more about the type of reader who reads this column. Do you have any idea the diversity of characters that pick up this journal? The one fact that amazed me was that you're not all Java developers. In fact, many of you have never coded a single line of Java in your lives and probably never will. So for a developer's journal this is good – means we're reaching a much wider audience.

I've had readers take this column to bed, much to the joy of their wives; I've had people reading on a plane; I even had a couple of people who read me while on the toilet (which for some strange reason makes me feel violated!). The diversity was there. But one person started me thinking of how to further unite these people. I was in deep dialog with one Miles Parker. Miles was putting forward some fantastic points and we thoroughly enjoyed debating the issues between us. But I was frustrated because I wanted some of my colleagues to be involved in this conversation as they would have had an awful lot to contribute.

It was then that I decided to create a mailing list dedicated to this column and the issues it brings up. I wish now to formally invite you all to join. It's not a spam list, and we don't abuse your e-mail address. It's just some good, clean conversation. To join, send an e-mail to listserv@listserv.n-ary.com with subscribe straight_talking-l in the body of the e-mail. From there you'll get instructions on how to participate. I will no longer be debating issues on personal mail. If you have a point to raise, join the list and we can all talk around it. I look forward to "seeing" you there.

> *"The most notable addition to our lives was the official release of Java 2.0 just before Christmas. ...On the whole, not a bad winter!"*

What Did Winter Bring Us?

The most notable addition to our lives was the official release of Java 2.0 just before Christmas. Along with Java 2.0 came a whole host of new APIs that finally made it out of the beta phase. The Servlet API got a major upgrade to version 2.1. The Java Communications API made it out into the wide world. The new Apple G3 platform was released, promising much for the support of Java natively. On the whole, not a bad winter.

Summertime plays host to JavaOne in San Francisco. I'm sure a number of new technologies will be released in that one special Java week. I'll be making my annual pilgrimage over the Atlantic to attend, and I hope to meet some of you there.

Speaking of San Francisco, I read the other week that it was voted the most connected Net-friendly city in the world, for the third or fourth year running. Please! Did that come as a big shock to anyone? I suspect not. San Francisco, or to be more precise the Bay area, plays host to the majority of the shakers and makers in our industry. It's like voting for the city with the best Empire State building! A bit pointless, methinks!

Love Is in the Details

I opened this month's rant with a paragraph hailing our love for the little guy. I guess I better explain why. Well, we're close to shipping our flagship product, n-formant. This is a Java application that monitors the performance of servers, alerting via phone/e-mail/fax should a problem arise. The Java application hosted a number of modules including a Web server that provided an intranet-style administration section controlled by a number of Java servlets. The application was quite a complex system, with a number of core technologies that allowed us to generate voice and fax calls. We bundled everything up into nice, small, maintainable JAR files.

Now, my first point of love is the JAR format. Aren't these just gifts from heaven? Much cleaner than DLLs and far easier to upgrade. Let me illustrate the foundation of our love with this problem. Our product, n-formant, ships as a complete hardware/software solution. As a company we made the decision to offer free upgrades. This meant we weren't supporting *x* different versions of the software. But it did give us a little headache on how we were going to ship the upgrades. Sure, the Web site would be utilized for this, but that would necessitate someone regularly checking the site for new versions. We figured out that if anything

needed upgrading, it would be one of the core JAR files we built.

So we built a generic automatic upgrade class that would periodically check a known site for a newer version and, if found, download and install it. Now the installation part was no more complicated than copying a single file. We would flag the administrator that a change was about to be made, and give them the change to yea or nay it. If the update was a small change or bug fix, then we could do this without alerting the client to the problem area. A system that would update itself! A wonderful idea, and one we can't take full credit for. We got the idea when the RealPlayer software decided to throw up a dialog box on my desktop one day indicating that a newer version was available and would I like to upgrade. With a single click of a button it downloaded the necessary upgrade, did the upgrade and restored my session back to normal. I was impressed. I thought, What a wonderful way to update our software!

By packaging our software in a JAR file, we could easily download the small file and restart that part of the system to allow the virtual machine to reload the classes without having to shut down or restart the whole system. A perfect solution. So, to the person who came up with the JAR file format, we salute you.

My tip is thus: don't go packaging up the whole class tree into one big JAR file. Try to break it down into small, logical units. Of course, this all depends on the target application. If you're coding a small applet, then sure – I can see where one JAR file is good. But if you're doing large deployment applications, then have a wee think about the structure of your JARs before you start producing 1 MB size .JAR files.

Getting to Know You

As indicated earlier, I'm all for knowing more about you – more about the person who picks up this column to read. To this end, a number of months ago I developed a simple voting servlet for our Web site that collates your views on a number of issues. The first month, I asked: "Was Sun correct to rename Java 1.2 to Java 2.0?" The results for this one were rather interesting: 51% of the total votes said they were correct. So it was cut cleanly down the middle as to the move of version number. Each month a new poll goes live on the Web site (www.n-ary.com/consultancy/), so check back and let us know your views.

Another recent question was: "Do you still find Java applets to be slow in execution?" This proved to be a bit more decisive. At this writing 70% of the voters believe that applets are still slow to run. Which is surprising, really, considering the amount of work that has been done to improve this state. But at the end of the day, when you have a 450 MHz running an applet the speed of a 33 MHz machine running Windows 3.1, you have to sympathize with the masses. Java 2.0 hasn't really made any significant improvement on this. But more on Java 2.0 in next month's column, when I'll look at another aspect of Java that has made us thankful….

April Book Review

This month's book review comes to you with the letter "C" and the color pink – sorry, had a *Sesame Street* moment there. The book that has captivated me this month is *Customers.com*, penned by Patricia B. Seybold. This book looks at how the big companies have embraced the Web and how they have made it work for them. It is a good book for companies that are thinking about expanding their Net presence and discovering what they need to do to bring their customers closer. However, one of the problems with any Net presence is getting people to visit. Smaller companies simply don't have the brand recognition to pull in hordes of visitors to their site. Therefore, a significant amount of their marketing budgets have to be deployed to raise the profile of their URL. This is an area I feel a lot of analysts have missed when talking about the billions to be made on the Net with e-commerce. Sure, local traders can suddenly start trading on the Net without too much hassle, but they aren't going to see the sort of sales growth that has been evangelized without investing serious money in external brand marketing. Which sort of goes against the grain of the low overhead of the virtual shop.

The days of getting thousands of people to come to your site just because you have a Java applet running are long gone. Who remembers the early Yahoo submission forms, where they asked if you had Java running on your site? This was supposed to increase your rating within the Yahoo indexing system. Looking back at it now seems quite amusing.

One wonders what systems we have today that will provide a chortle in a few years. I think this is what we'll look at next month. So stay tuned....

About the Author

Alan Williamson is CEO of n-ary (consulting) Ltd. A Java consultancy company with offices in Scotland, England and Australia, they specialize solely in Java at the server side. Alan is the author of two Java Servlet books and contributed to the 2.1 Servlet API. He can be reached at alan@n-ary.com (www.n-ary.com).

alan@n-ary.com

Parsing Command Line Arguments with Java

Using an effective Java framework to write command line tools

by **Panos Kougiouris**

One of Java's great appeals is that the language provides out-of-the-box GUI development capabilities. Still, a lot of us use Java to write command line tools. Such tools are great to automate batch and offline processes. This article presents a framework that jump-starts the development of such tools.

Command line tools are usually invoked from a shell (e.g., DOS prompt, sh, ksh, etc.) and perform a certain task. The task can be customized based on the command line arguments. For instance:

```
telnet foo.bar.com
```

attempts to open a telnet connection to host foo.bar.com. It uses the default telnet port. The next example:

```
telnet -p 3434 foo.bar.com
```

attempts a similar connection using port 3434.

Command line tools can be as simple or as complicated as the developer desires. An example of a simple command line tool is the echo command found in most shells. On the other hand, the Java compiler and the Java Virtual Machine (JVM) are complex command line tools.

Java presents the command line arguments in an array of strings. This is already a huge improvement over C and C++, in which the arguments are presented as an array of C strings, i.e., an array of pointers to arrays of characters. Yet it comes short of the developer's desire to get the arguments parsed and ready to use.

Since I published my 1997 C++ command line parsing framework (see Reference), many readers have e-mailed me with requests and suggestions. The top two requests have been for a Java implementation and an improvement to handle arrays of arguments. In this article I present a total rewrite of the framework with improvements for Java programmers.

Using the Framework

Before moving to the implementation I'll demonstrate how to use the framework to write a command line utility. Let's say you want to write a utility called "mycat" – like the UNIX cat – which takes a number of files and concatenates them together into a larger file. A –v option turns verbose output on and off. A –l option allows the insertion of extra empty lines between the files. Ths command would look like:

```
mycat [-l <int>] [-v]  file1 file2 ... .
```

In your Main class you need to add a Token object for each argument. In this example we have three Tokens: the number of lines, the verbosity mode and the input files. In addition, you need to add an ApplicationSettings object. This object is used to contain all the arguments.

The source code for these settings is shown in Listing 1. I first declare the sm_main variable and then the three Token variables: sm_verbose, sm_files, sm_lines. The arguments in the constructor of each token object fully describe the expected usage of the Token:

- Is it a switch or an argument?
- What is the switch's name (e.g., –v)?
- What is its type (integer, string, etc.)?
- Can it appear multiple times (e.g., –l, 1, –l, 2)?
- Is it a required argument?
- If not, when the argument is missing:
  1. Is there an environment variable to provide the value?
  2. Is there a default value?

A static initializer adds the Token variables to the ApplicationSettings variable. By the time the main() function of your application is reached, the ApplicationSettings object knows everything about the syntax of your command line utility.

Listing 2 shows the main program of my example. The first line after the try statement calls the parseArgs() method of the ApplicationSettings object. The actual command line arguments are passed as an argument to the object. When the syntax is incorrect, a usage message is printed and an exception is thrown. Otherwise, the Token objects are set to contain appropriate values. For instance, when the –v option is present, the sm_verbose object will be set. Later, when its getValue() method is called, it will return true.

In a similar fashion, if two files are passed as arguments, let's say foo.cc and bar.cc, the sm_files Token will be set appropriately. Its getValue(0) method will return foo.cc, its getValue(1) method will return bar.cc.

Now compile the example with your favorite development environment and run the resulting code without passing any arguments. You should get the usage message in Listing 3. But wait a minute: you never wrote code to print usage messages; what's going on here? It's very simple. The framework uses the same code that defines the expected Tokens to generate usage statements. Kiss the ugly, always-out-of-date, static String statements that describe the usage of the utility goodbye.

Now let's run the program again with some decent arguments. Let's say we run it with arguments "–v foo.cc bar.cc". The program prints the arguments correctly. Though we didn't pass any value for the –l switch, the Token returns 0. This is the expected behavior because the default value of the sm_lines Token is indeed 0.

Why Use the Framework?

By now some of the advantages of the framework should be obvious to you. The error-prone *while* and *switch* statements that usually parse the arguments have been replaced by a few very readable statements.

These statements:

- Document the usage of the command line utility
- Encapsulate the settings so they can be used by the rest of the program
- Automatically generate usage messages when the user enters incorrect syntax:
  1. missing arguments
  2. unexpected arguments
  3. wrong types of arguments

The stated advantages speed up the original development of any command line utility. They allow the developer to jump to the real code as soon as possible. At the same time, they provide immediate access to the command line settings and usage messages.

Where the framework really shines is in the area where most of a developer's time is spent: software maintenance. If a command line utility is successful, users will ask for changes and improvements. Many of them will translate to more command line options or change the syntax of existing ones. The framework makes adding and modifying options trivial and safe. Compile-time messages will save the developer from runtime embarrassment.

Finally, the framework is extensible. One can define new types of switches that accommodate new data types or anything else a developer desires.

At this point you can go ahead, download the code and start using it in your own applications. The next few sections discuss the design of the framework.

The StringArrayIterator Class

The StringArrayIterator class is a utility class (see Listing 4). It encapsulates an array of strings and a position inside the array. The get() method returns the String at the current position. The moveNext() operation on the array allows the programmer to advance the current position to the next string. The EOF() operation determines when the end of the array has been reached.

The ApplicationSettings object contains a StringArrayIterator object. It gets initialized from the command line arguments.

The Token Class

The Token class, shown in Listing 5, is an abstract class. Each Token object contains a description of an argument or a switch. After a successful parsing it also contains the value or values that were provided for the argument in the command line.

During the parsing phase, the most important methods of the Token class are the parseSwitch() and parseArgument() methods. Both of them take the StringArrayIterator object with the command line arguments as input. If the current command line argument is recognized, three things occur: it's parsed, the pointer of the StringArrayIterator object is moved and a value of true is returned. If it's not recognized, a value of false is returned.

The values that correspond to this switch or argument are stored in a Vector of objects. Subclasses determine their class. For instance, the StringToken subclass will have String objects, and the IntegerToken subclass will contain Integer objects.

While the program is running, the values are accessible using the getValue(int) and getValue() operations.

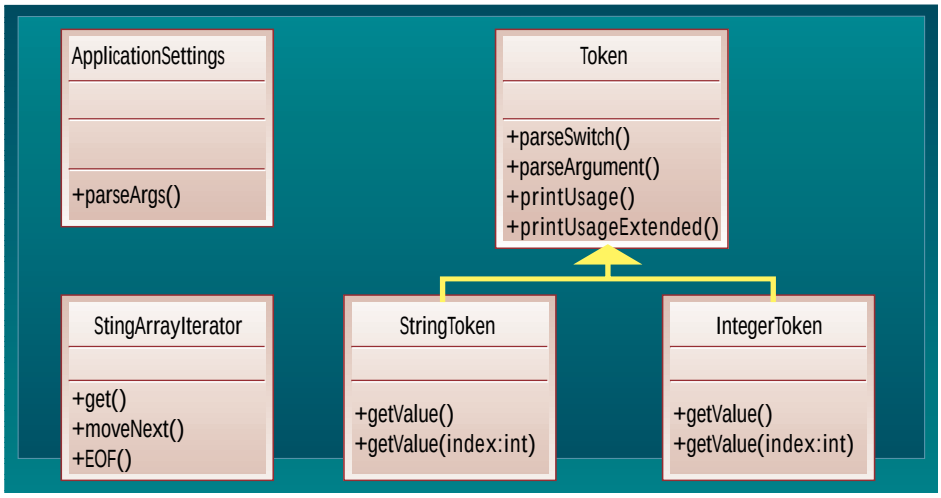


Figure 1: A UML class diagram of the parsing framework

Token Subclasses

A Token subclass encapsulates arguments of a specific type. For example, there's a StringToken, an IntegerToken, etc. Since most of its methods have a generic implementation, each Token subclass has very few methods to implement.

Listing 6 presents the implementation for the class StringToken. A few more subclasses are provided in the downloaded code. You can extend the framework by implementing more subclasses.

The ApplicationSettings Object

The ApplicationSettings object puts everything I've discussed so far together (see Listing 7). It contains all the Token objects and initiates the parsing algorithm. The user triggers the parsing by calling the parseArgs() method.

The command line arguments are assigned into the StringArrayIterator member of the class. Then for every command line argument, each Token object is called and asked to parse it as either an option or an argument.

If no Token object can parse the argument, a usage message is printed. The usage message is printed by iterating through the Tokens and calling their printUsage() and printUsageExtended() methods. Both methods take an OutputStream as an argument. They print their output to this stream.

Pure Java and Impurities

Almost all the code is pure Java. Since pure Java doesn't provide support for environment variables and assertions, I had to use the functions provided in my environment, the Win32 Virtual Machine.

These few lines of code are carefully isolated in the util.java file shown in Listing 8. In a pure Java environment you can comment out three lines of code from this file. You don't get assertions and support for initialization of arguments from environment variables. Otherwise, everything else works as advertised.

Limitations

The framework doesn't provide support for complex scenarios. For instance, there's no support for switches that depend on each other. You can't dictate that the –t option can appear if and only if the –p option appears. You'd have to implement such checks yourself after the arguments were parsed.

Conclusion

In this article I presented an extensible Java framework. The framework simplifies the development and maintenance of code that parses the arguments of command line utilities and tools.

The framework doesn't provide support for complex scenarios. Still, my experience is that the framework covers most common cases. I expect that it will be as useful for Java development as it has been for C++.

Reference

Kougiouris, Panos (1997). "Yet Another Command-Line Parser," C/C++ Users Journal, Vol. 15, No. 4, April.

CODE LISTING

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

About the Author

Panos Kougiouris has ten years' experience in software development for high-tech companies. For the past three years he has been at Healtheon, a Silicon Valley startup, and he has held technical positions with Oracle and Sun Microsystems. Panos holds computer science degrees from the University of Illinois at Urbana-Champaign and the Univ. of Patra, Greece. He can be reached at panos@acm.org.

panos@acm.org

Climbing a JTree FOR THE FIRST TIME

Displaying your business objects in a JTree

by **Mark Steenbarger**

Since the introduction of the Java Foundation Classes (JFC), Java applications have been able to be implemented using a rich set of window components. These components – called Swing – along with customizable "look and feel," allow applications to be implemented without relying on a native windowing system. With the release of Java 2 (a.k.a. JDK 1.2), the JFC has found a permanent home as part of the JDK rather than being distributed separately. Swing includes two very powerful but complex components called the JTable and JTree. This article focuses solely on the JTree and explores various aspects of the JTree by using two examples that show how business objects can be visually represented within the JTree component.

Using the Model-View-Controller (MVC) pattern, originally from Smalltalk, the majority of the Swing classes are implemented by using a variation of the MVC that collapses the view and controller into a single class called the delegate. (For more details on this topic, you can visit an online Swing tutorial, "What is Swing? – 2"). A working understanding of the model/delegate relationship will help you understand the classes and interfaces that accompany each of the Swing components.

As stated by Sun, "With the JTree class, you can display hierarchical data. JTree doesn't actually contain your data; it's simply a view of the data." The challenge comes from associating business objects

with a corresponding Swing object. So how do you represent your current business objects in a JTree without altering your class definition of the business objects?

This article covers two alternatives. In both examples the objectives are to minimize (1) the coupling between the business objects and the JTree, and (2) the amount of effort and code to accomplish the tasks. In both examples I use a class called *Vehicle* that represents any business object and that can be displayed in hierarchical fashion. Since the JTree is designed for displaying data with hierarchical properties, the only requirement is that there be methods within the business object's class definition to implement navigation within a hierarchical structure. In the examples, the Vehicle class has subtypes that provide a hierarchical structure. For example, one instance of the Vehicle class might be called "Motor Vehicle," which could contain subtypes of Car, Truck and Van. These vehicles, while separate, share the commonality of being a motor vehicle.

Throughout this article I use terminology common to the JTree API and its associated classes and interfaces. The terminology is defined for you: in the JTree Terminology Table.

JTree Terminology

Node: Any position within the JTree where data associated with the business object is being represented.

Path: A collection of a contiguous set of nodes. A path can contain one or many nodes. A null path indicates a zero node path or an empty path. The collection of nodes will consist of a strict ancestry line. (If you think of a traditional organizational chart as a tree, then an example of a path would be the line drawn from you to the president or CEO.)

Leaf: A special kind of node. As its name implies, this is the node at the end of a path. There are no more nodes connected to the leaf node. (Using the organizational chart example again, the leaf is the person that has no personnel reporting to him or her.)

Root: A special kind of node. In comparison to a leaf, a root's parent information is never examined. It's the highest point within the hierarchy. A root's parent relationship either does not exist or doesn't need to be displayed.

Parent: Represents a node's relationship with another node. In a parent/child relationship, the parent is analogous to a super class within the realms of object-oriented concepts.

Child: Represents a node's relationship with another node. In a parent/child relationship, the child is analogous to a subclass of its parent. It inherits all the properties associated with its parent. (*Note*: As of JDK 1.2/Swing 1.1, a node only can have one parent.)

User Object: Refers to the business object associated with a node. While not required, all user objects will usually be of the same class type. (In the examples provided, the Vehicle class is used to represent the business object.)

Editor: This is a component (usually an extension of a JComponent) that has the unique role of allowing the user to change the data of a specific node.

Renderer: This is a component (usually an extension of a JComponent) that has the unique role of deciding how a node's data is to be displayed within the context of the JTree when a user isn't editing the data. (*Note*: Using an AWT component as an editor or renderer may generate unwanted results. See articles that relate to mixing lightweight with heavyweight components.)

TreeModelEvents: Swing provides the following three types of tree events:

1. ***Expansion event*** – an event generated when a node is collapsed or expanded.
2. ***Model events*** – there are four types of model events:
   a. *node changed* – generated after a node is changed. This is the only event the TreeModel interface supports with the method valueForPathChanged(TreePath path, Object newValue.) While this method could be implemented to represent any of the four types of model events, typically this represents the node changed event, and the DefaultTreeModel class implements it as such.
   b. *node inserted* – generated when a node is inserted into the JTree.
   c. *node removed* – generated when a node is removed from the JTree.
   d. *structure changed* – a "catchall" event used when something drastic has happened to the structure of the JTree. It's the most expensive event as it may result in a repaint of the entire JTree.
3. ***Selection event*** – an event generated when the selection of a node takes place.

One aspect of the JTree that makes it more complex than other Swing classes is that the associated model in the model-delegate pattern for the JTree isn't where data is maintained. Take the JTextField class, for example. In JTextFields, the view (JTextArea) offers a setText(String) method, and its associated model (called a document) offers an insertString(int, String, AttributeSet). Both methods allow manipulation of the underlying data. In the case of the JTree, neither the JTree class nor its associated model interface – the javax.swing.tree.TreeModel – offers a means of manipulating the underlying data.

Another aspect of the JTree is the add(Component) method, which comes from being a subclass of the Container class. This method, however, is used for performing additions in the sense of containment (i.e., JPanels are commonly used to contain several other components by adding them to the JPanel), not for adding data to the JTree.

In the first example – AddData_ExampleA.java (see Listing 1) – associating a business object's data to the JTree is done using the default helper classes: javax.swing.tree.DefaultTreeModel and javax.swing.tree.DefaultMutableTreeNode. The default model class consists of the methods insertNodeInto(MutableTreeNode, MutableTreeNode, int) and removeNodeFromParent(MutableTreeNode). These methods allow the addition and removal of nodes from the JTree. Since my business object, the Vehicle class, doesn't implement the MutableTreeNode interface, it can't be directly added to the JTree. Therefore, to make it a valid MutableTreeNode without altering the class definition, I "wrap" the Vehicle class using

the DefaultMutableTreeNode.

Here are the steps performed within the code in AddData_ExampleA.java:

1. *Obtain a reference to a user object.* An instance of the user object is created. In this example, the Vehicle class is the user object:

```
Vehicle vObj = new Vehicle("Transportation
Vehicles");
```

2. *Create an instance of TreeNode.* An instance of the DefaultMutableTreeNode class that implements the MutableTreeNode interface, (a subinterface of TreeNode) is created using the instance of the user object created in step 1. (The second argument indicates whether the node will allow children to be added to it. In this example, I want to allow children so I pass in the value *true*.):

```
DefaultMutableTreeNode tRoot = new Default-
MutableTreeNode(vObj, true);
```

3. *Create an instance of TreeModel.* The DefaultTreeModel class implements the TreeModel interface and can be created using the TreeNode object that was created in step 2 as its constructor's argument:

```
i_model = new DefaultTreeModel( tRoot );
```

4. *Create an instance of the JTree.* The JTree is created using the TreeModel object that was created in step 3:

```
i_tree = new JTree( i_model );
```

5. *Create a child TreeNode.* When the user clicks the add button, another instance of the DefaultMutableTreeNode is created using another instance of the Vehicle class with the name of "Car":

```
i_car = new Vehicle( "Car" );
 i_carNode = new DefaultMutableTreeNode(
i_car );
```

6. *Add child node to the JTree.* The method insertNodeInto(MutableTreeNode, MutableTreeNode, int) from the DefaultTreeModel class is invoked on the TreeModel that was created in step 3. There are three arguments. The first argument consists of using the instance of the DefaultMutableTreeNode that was created in step 5. The second argument calls for the parent of the object being inserted, which in this case is the root. To obtain the root, the TreeModel interface provides a getRoot() method. (*Note*: the return type of getRoot() is Object, which requires casting the returned object to the MutableTreeNode class.) The third argument requires an *int* to indicate where within the children (assuming more than one child) the new node should be graphically positioned. Since there are no other children, the value used is 0:

```
i_model.insertNodeInto( i_carNode, (Mutable-
TreeNode)i_model.getRoot(), 0 );
```

To see this example run, compile and execute the AddData_ExampleA.java (see Listing 1) source file. Upon executing the application, the JTree is displayed showing a single node – the root (see Figure 1).

At this point steps 1 and 2 have been completed. Once the user invokes some action that indicates adding a node to the JTree (in this example, clicking the button labeled Add Node: "Car" button) a new node is added to the tree (see Figure 2).

To remove a node from the JTree, the DefaultTreeModel method – removeNodeFromParent(MutableTreeNode) – is called using the object created back in step 3 of the add process:

```
i_model.removeNodeFromParent( i_carNode );
```

To see this happen, click on the button labeled Remove Node: 'Car' button in the AddData_ExampleA.java program. By repeating steps 5 and 6, the program will construct the entire contents of a JTree.

While this add process is simple and easy to program, there are some shortcomings with this approach. First, it demands

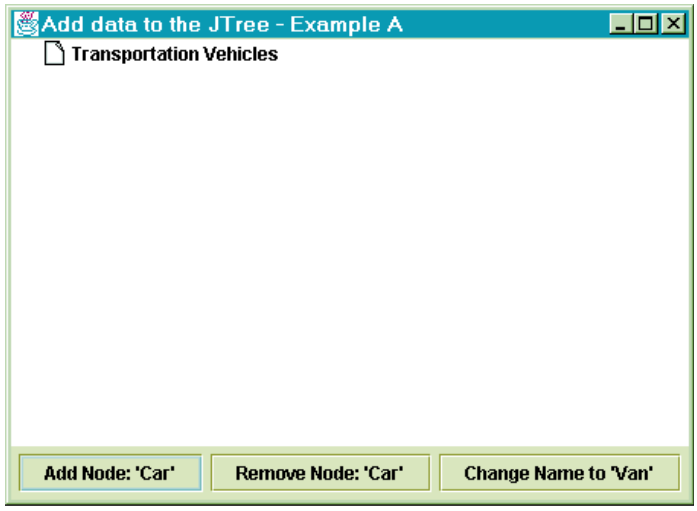


Figure 1: JTree after inserting one child node to the root

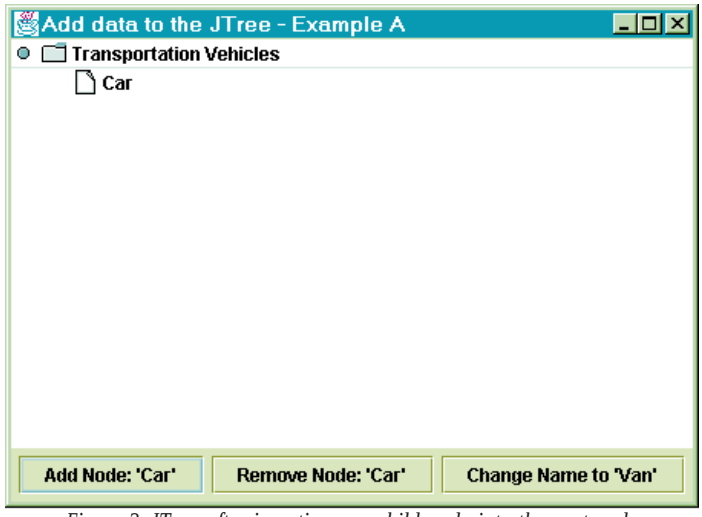


Figure 2: JTree after inserting one child node into the root node

that the application constructing the JTree take full responsibility for constructing and maintaining all the hierarchical relationships between each node. The code to handle this can easily become too large and difficult to debug or maintain.

A second shortcoming is that the responsibility of keeping concurrent data accurate falls back on the application containing the JTree. Running the AddData_ExampleA class explains this. After you create and add the child node "Car," if the button labeled "Change Name to 'Van'" is clicked, the node that previously displayed "Car" will now display "Van." However, for the refresh to occur immediately, the following code is required:

```
i_model.valueForPathChanged( pathToRoot,
i_car );
```

Another method, called valueForPathChanged(TreePath, Object), is provided as part of the TreeModel interface (see Figure 3). However, it again requires knowing which node has changed and the path in which it resides. The reason the update isn't "free" is because Swing is still not aware of attribute changes made to the node's user object.

A third shortcoming is with the use of the default classes that are provided by Swing. While convenient to use, it should be noted that certain limitations and costs exist. In my example, the DefaultMutableTreeNode is not a thread-safe class. Other issues relating to performance may need to be addressed when using the "default" classes in Swing.

It should also be noted that since the methods insertNodeInto() and removeNode() aren't part of the TreeModel interface, calls made to the getModel() method will require casting prior to invoking these methods. This defeats the advantage of using interfaces because if these methods were part of the TreeModel interface, then the cast to DefaultTreeModel after getModel() wouldn't be necessary.

The second example, displaying a business object's data in a JTree, is done by creating a tailored MutableTreeNode class. Writing my own MutableTreeNode class gives me a "bridge" between the user object class being displayed and the Swing MutableTreeNode interface. I use the term *bridge* to imply that there will be a translation between API calls invoked by one class and the appropriate methods invoked in a corresponding target class (see Figure 4).

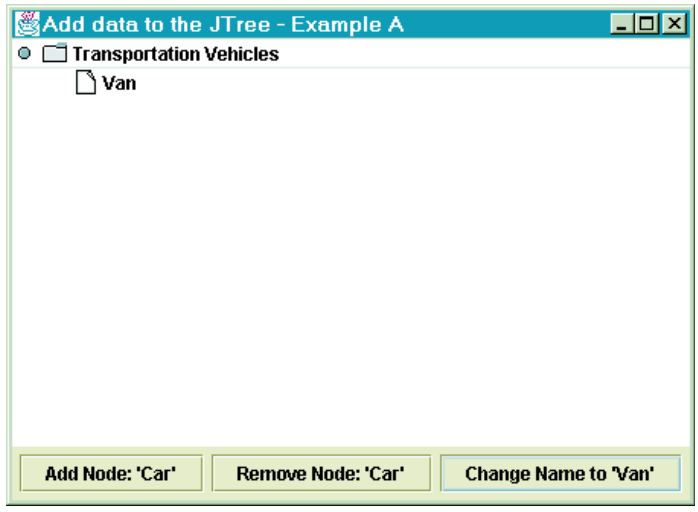


Figure 3: Result of calling the valueForPathChanged (TreePath, Object) on the TreeModel

This allows the target class (the user object class) to be free from knowing the functionality of the calling class, and vice versa. Therefore, the Vehicle class definition (see Listing 2) isn't influenced by how Swing is implemented.

Note: It's good practice to implement the toString() method in your objects to provide a meaningful String representation of the class. The JTree uses the toString() method to determine what text to display in the TreeNodes.

Accomplishing this requires completion of the two steps listed below:

1. Create a MutableTreeNode (VehicleTreeNode) class (see Listing 3) for the Vehicle class. This class will implement the MutableTreeNode interface by invoking methods defined in the Vehicle class.
2. Update the JTree to reflect changes made to the user object.

When changes are made to the underlying object the JTree won't update its view to reflect the new value until it's prompted to do so. As discussed earlier, the method valueForPathChanged(TreePath, Object) on the TreeModel will update the JTree view. However, to invoke the method requires a reference to the TreeModel that can't be obtained from a TreeNode. Therefore, I chose to implement an event mechanism as a logical means of communicating updates made by the MutableTreeNodes to their associated user objects. This required creating two interfaces (UpdateEventSource, UpdateEventListener) (see Listings 4 and 5) and one class (UpdateEvent) (see Listing 6) for the event.

Now to bring it all together! Following is the sequence of steps that occurs when executing the AddData_ExampleB (see Listing 7) application class:

1. Create an instance of the root Vehicle class:

```
Vehicle wrkVehicle = new Vehicle( "Vehicle" );
wrkVehicle.setType( "Motor" );
wrkVehicle.setDescription( "Classification
for motor vehicles" );
```

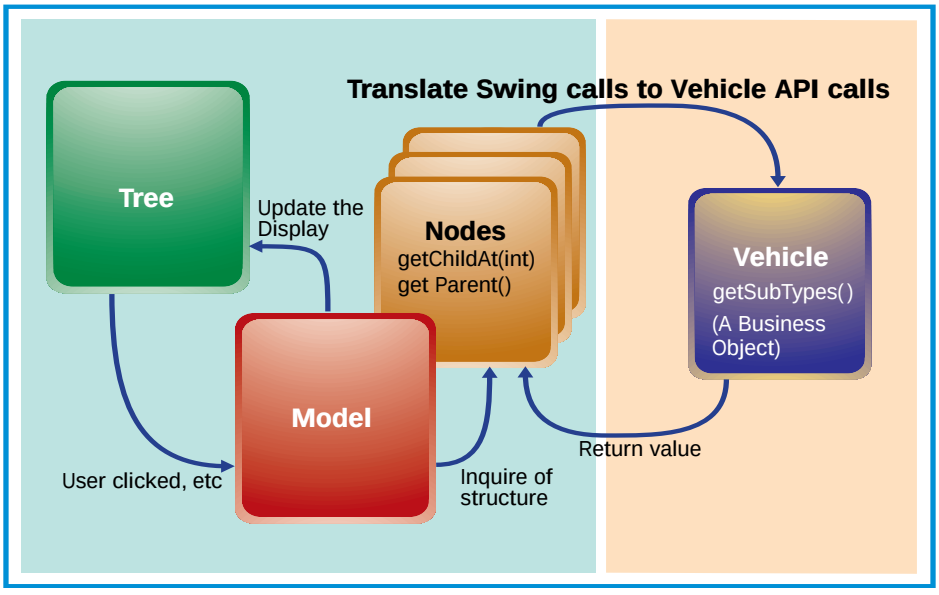


Figure 4: Result of creating a MutableTreeNode class that interfaces with the Vehicle class

2. Create an instance of the VehicleTreeNode class. The argument used in its constructor is the Vehicle class that was created in step 1. The second argument is a Boolean that indicates if the node being created will allow children. In this example, the nodes will have children so the value of true is used:

```
i_root = new VehicleTreeNode( wrkVehicle,
true );
```

3. Create an instance of a TreeModel class. By using the DefaultTreeModel class, the constructor is passed in the root node created in step 1 and true is passed in to indicate that children are allowed:

```
i_model = new DefaultTreeModel(i_root, true);
```

4. Create an instance of the JTree. The JTree is constructed using an instance of the DefaultTreeModel class, which is constructed using the VehicleTreeNode object that was created in step 3:

```
i_tree = new JTree( i_model );
```

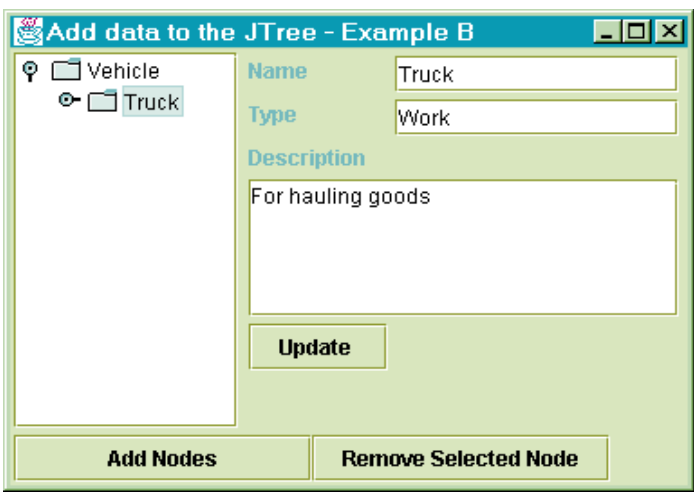


Figure 5: Translating Swing method calls to business object method calls

At this point the work is finished and the magic of the JTree begins (see Figure 5).

Here's how this works: subsequent calls are made by the JTree to the TreeNode (in my example, it's the VehicleTreeNode), asking if it allows children (allowsChildren). If so, it obtains a child count (getChildCount), iterates through the list of children and sets the current node as the parent (setParent) on the child node. This will repeat for each node until the leaf node is reached (getChildCount returns 0). Actually, the default behavior is to perform these steps in a lazy fashion. Rather than take a performance hit by obtaining the entire structure of the JTree right away, nodes are displayed in a collapsed state and wait until they're expanded before completing construction of the JTree.

Adding nodes to the JTree is accomplished by invoking methods similar to those in the first example. By invoking the DefaultTreeModel's method, insertNodeInto(MutableTreeNode, MutableTreeNode, int) with a VehicleTreeNode as the required MutableTreeNode, any subtypes associated with the Vehicle will also immediately appear on the JTree. This differs from the first example in that subsequent calls to the insertNodeInto() method would be required to add the subtypes of the Vehicle to the JTree. This can be seen by running the AddData_ExampleB.java application. The Truck/Vehicle is added to the root Vehicle prior to being added to the JTree. So when the root Vehicle is added to the JTree, the child node Truck is also added to the JTree without requiring the additional call to insertNodeInto().

It's worth noting that while the MutableTreeNode interface offers methods like insert(), remove() and removeFromParent(), invoking these methods directly to alter the parent /child relationships circumvents the TreeModel. Since the TreeModel maintains the view, changes made directly to the MutableTreeNodes won't be reflected until a forced repaint occurs (resizing the window, etc.).

Summary

The first example demonstrated how data could be added to a JTree simply and easily. In the example, the handling of the details of the JTree was delegated to the Swing default classes. It's a good solution if the JTree is going to be used to display predominately static or read-only data. However, if the JTree is to be used heavily, such as in an administration application, then the second alternative – creating a specific MutableTreeNode class to handle the translation between the graphical and data classes – may be more appropriate. It minimizes the resources necessary to perform the construction of the tree, and the code that needs to be written is kept at a minimum.

I hope this article assists developers who are new to Java, or to the JFC and Swing to quickly become acclimated to the power of using a JTree to graphically represent and administer their data objects.

Resources

1. http://java.sun.com/docs/books/tutorial/uiswing/components/tree.html
2. http://java.sun.com/docs/books/tutorial/uiswing/overview/swingFeatures.html#model

CODE LISTING

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

About the Author

A graduate of Taylor Univ. with a degree in computer science/artificial intelligence, Mark Steenbarger is a senior software engineer at Tivoli Systems, Inc. He develops Enterprise Service Management software as a member of the Applications Team and has been developing ESM software since 1996. Mark can be reached at ark.steenbarger@tivoli.com.

mark.steenbarger@tivoli.com

Building a Tree Viewer

Viewing hierarchical relationships graphically in Java without using Swing JTree

by **Daniel Dee**

A Tree for All Occasions

The Java Foundation Class, also known as Swing, in addition to augmenting, enhancing and generally implementing platform-independent replacements of AWT components, added the JTree class to its repertoire of new GUI components. Swing's JTree supports a Windows Explorer-style (outliner-style) tree that makes it easy to graphically render data with hierarchical relationships. A limited number of graphical attributes as supplied by the default look-and-feel provided with Swing – e.g., the icon representing the nodes – is also readily configurable.

This article describes an implementation similar to JTree that can be used in Java 1.0.2 or when using Swing may not be an option. This implementation also adds features not found in the default JTree. These include the option of rendering a tree vertically or horizontally, and of aligning it left, center or right. In addition, subtrees can be interactively moved from one branch to another using drag-and-drop. I also describe how some of these extra features can be migrated to Swing when it becomes more popular in the future. The source code in this article requires the Callbackable, CallbackList, Widget, PositionableGridConstraints and PositionableGridLayout classes introduced in previous issues of ***JDJ***.

Creating a Tree – The TreeNode Class

A tree is essentially a set of hierarchically related nodes. In this implementation, each node is an instance of the TreeNode class. A user object may be stored in a TreeNode, including any AWT or composite AWT Component. While TreeNode doesn't contain any rendering code, the TreeViewer class stores the icon for each node in its representative TreeNode instance. Thus each TreeNode can be represented with a unique icon if necessary.

An instance of the TreeNode class stores references to its parent and children. A null reference for the parent means that this node is the root of the entire tree. A null reference for its children means that the node is a leaf. Listing 1 shows the implementation of a TreeNode class. The following code fragment shows how to create the tree shown in Figure 1:

```
TreeNode root = new TreeNode( null, null,
null, "Root", null );
      new TreeNode( root, null, null,
       "Level 1.1", null );
      new TreeNode ( root, null, null,
       "Level 1.2", null );
      new TreeNode ( root, null, null,
       "Level 1.3", null );
      new TreeNode ( root.getChild(1),
       null, null, "Level 2.1", null );
      new TreeNode ( root.getChild(1),
       null, null, "Level 2.2", null );
      new TreeNode ( root.getChild(1),
       null, null, "Level 2.3", null );
```

The first line creates the root of the tree; note the null passed in the first parameter, which indicates no parent. Lines 2 to 4 create the second level; note that root is passed as the first parameter in this case. Lines 5 to 7 create the third level; note that the child of the root with index 1 is use for the first parameter.

Walking the Tree – The TreeWalker Class

While a collection of hierarchically linked TreeNode instances implements the structure of a tree, the TreeWalker implements an abstract class for traversing the tree. In other words, given a tree, an extension of the TreeWalker will traverse it in either a breadth or depth first approach, performing an action on each node as it goes. Listing 2 shows the code of the TreeWalker class. When creating an instance of a TreeNode, an instance of a TreeWalker extension is usually passed as the last parameter of the TreeNode constructor. Listing 3 shows the TreePrinter class, which is an extension of TreeWalker for printing each node of a tree.

A sample printout of the tree in Figure 1 using the TreePrinter will yield the console output seen in Listing 4.

Rendering the Tree – The TreeViewer Class

TreeViewer is an extension of the TreeWalker class specifically for providing an interactive graphical front-end to a tree. The TreeViewer class takes a tree, traverses it and renders each node on any AWT display surface – e.g., Canvas or Panel. As noted before, the AWT Component stored in each TreeNode instance is used to render the node. For example, if a Button is to be displayed for a specific node in a tree, then an instance of the Button should be added to the TreeViewer, giving its location in the tree with respect to a parent node. The TreeViewer also manages the relocation of subtrees when the user initiates a drag-and-drop action.

TreeViewer contains the TreeViewerPanel class that uses the custom LayoutManager introduced in the article "Implementing a Grid LayoutManager with Positionable Components" (***JDJ*** Vol. 3, Issue 12) to lay out the tree. The depth of a node in a tree hierarchy determines its vertical position in the grid. To determine its width, the TreeViewer recursively walks the tree (using the walkDepth method) to determine the grid position of each node relative to its neighbor on the left. After adjusting for alignment and orientation to determine its exact location on the grid, the node, as well as the lines leading to it from its parent, is drawn.

Actions on the nodes are left to the AWT Components representing the nodes themselves to handle. However, Tree relies on the Widget class introduced in the Widget-izing AWT Components to implement drop-and-drag. Because some AWT Components (notably the Button) recognize only the action event in some implementation of Java 1.0.2, the drag-and-drop capability is also implemented in the TreeViewerPanel. In general, a node

or a subtree may be dragged by holding the mouse button just outside of the Button representing the node or the root of the subtree, and then moving the mouse. If the particular AWT implementation supports dragging inside the Component (e.g., Button), dragging may be accomplished by holding the mouse button down inside the Component itself and then moving the mouse. When the user moves a subtree, the callback determines the drop site, removes the subtree from the original location and attaches it to the new parent (MouseDragCallback). The complete TreeViewer implementation is shown in Listing 5.

Using the Tree Class to Display a Tree

Creating a visual representation of a tree using the TreeViewer class parallels that of creating a tree structure using the TreeNode class. Listing 6 shows an application that creates an interactive tree, which is center aligned and vertically oriented. The application requires two command line parameters: the first to specify whether the tree is to be displayed "horizontally" or "vertically"; the second to determine whether it is to be displayed "left," "center" or "right" align. The root will be created by default. Click on a button to create a child for it. To move a node, hold down the mouse button just outside the Button representing the node and drag it to the node that will become the new parent of the subtree. An entire subtree may also be moved this way.

TreeViewer and JTree

TreeViewer and TreeNode are analogs of Swing's JTree and TreeNode. Both TreeViewer and JTree are the respective renderer and event handler in each implementation.

By default, JTree provides only a Windows Explorer-style tree. In reality, JTree delegates most of its look-and-feel decision to an instance of the BasicUI, so it is possible to customize the renderer by extending the supplied BasicTreeUI class. As Swing gradually becomes the GUI toolkit of choice for Java, the code use in this implementation of a general tree viewer may be migrated.

Conclusion

In this article we introduced a TreeViewer implementation with extra features that are not in Java's Swing-supplied implementation. In general, TreeViewer could be a powerful tool for visualizing hierarchically organized structures. A future article will explore the use of TreeViewer in viewing the organization of an AWT graphical user interface design – a step that will take us one step closer to implementing a GUI layout editor.

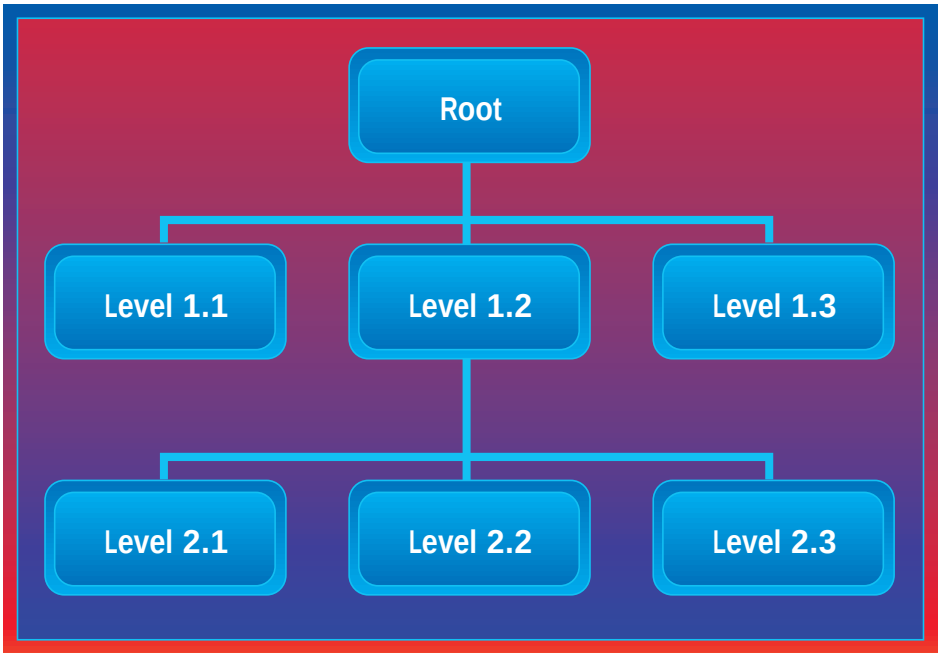


Figure 1: A simple tree

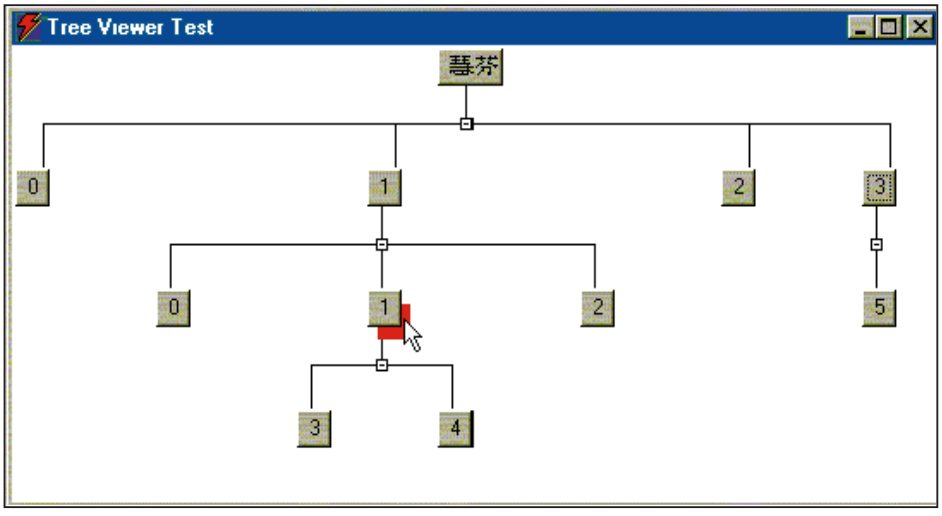


Figure 2: This figure shows a vertically oriented, center-aligned tree using widgetized Buttons as nodes. Also shown is a red rectangle indicating that the "1" Button is being dragged. Dragging is accomplished by holding down the mouse button – represented here by the white cursor – near the Button to be dragged. The red rectangle will trace out the drag path while the mouse button is continued to be held down. Releasing the mouse button on top of any other Buttons moves the entire subtree rooted at "1" to the new location.

Download Source Code

The program in this article requires the callback and widget code from the Implementing Callback and Widget-izing AWT articles published in the April and June 1998 issues of ***JDJ*** (Vol. 3, Issues 4 and 5). It also requires the positionable grid layout code from the December issue of ***JDJ*** (Vol. 3, Issue 12). Full source code for this article (including a Java 1.1 version) can be downloaded free from www.wigitek.com. A version that also supports the collapsing of subtrees for Java 1.0 and 1.1 is also available from Wigitek Corporation at the same Web site.

CODE LISTING

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

About the Author

Daniel Dee has two MS degrees and is currently president of Wigitek Corporation. He has more than 10 years' working experience in the development of GUI software toolkits, using X versions 10 and 11 and then Java since its inception. Daniel can be reached at daniel@excaliber.wigitek.com.

daniel@excaliber.wigitek.com

Developing Cross-Platform Applications in Java

Making "write once, run anywhere" a reality

by Steven Gould

One of the primary design goals for Java, and still one of the primary reasons for using it, is the idea of "write once, run anywhere." This allows software developers to develop an application using their preferred operating system and to deploy it to a wide range of other platforms. Java is therefore an ideal language choice when faced with the challenge of developing a platform-independent application.

Many articles have been written on applet development and the use of Java for bringing more interactivity to pages on the Web. While this is a valid use of Java, it's equally well suited for developing more traditional standalone or client/server applications that can run across multiple platforms. When Java is used to develop each of the tiers of an *n*-tier client/server application, its cross-platform nature offers additional benefits, including scalability (independent of hardware platform), flexibility and vendor independence.

Which JDK?

One advantage of application development over applet development is that you have much more control over which version of the Java Development Kit will be used to run your application. If you're developing a small application for deployment in the next couple of months, you should probably use JDK 1.1. However, you may want to consider using JDK 1.2 for slightly longer-term projects. It offers many new features and improved performance.

Whichever version you decide to use for your application, be sure to bundle the appropriate Java Runtime Environment when you distribute your application.

Application development, with its more relaxed security model, allows lower-level access to the operating system. This low-level access, however, can result in your application's becoming platform-specific. In this article we'll look at five primary areas you should pay special attention to when developing cross-platform Java applications:

1. Adopting good programming practices
2. OS differences and limitations
3. Reading and writing files
4. Designing your graphical user interface
5. Other issues

Adopting Good Programming Practices

Depend Only on the Core APIs

The Java core API forms a standard foundation of classes that all Java Virtual Machines must implement to be considered "standard Java." These are the only classes you can reasonably expect to be available on the end user's machine.

If you use any third-party class libraries, you'll need to distribute their runtime versions (which may require a separate license). It's also recommended that you verify that these third-party libraries make use of the core Java API only and don't use any native calls. Otherwise the code you write may be cross-platform. However, since your code depends on a third-party product that's not cross-platform, your resulting application won't be cross-platform either.

Enable All Compiler Warnings

By default, the Java compiler generates warnings for code it considers ambiguous, platform-dependent or unclear. Although you can disable these warnings using the javac -nowarn option, it isn't recommended. Good programming practice suggests enabling all warnings – to their maximum level, if appropriate – in your compiler, then changing your code to eliminate all warnings produced.

Avoid Deprecated Methods

In Java application development it shouldn't be necessary to use deprecated methods. While they may work in the current release of the JDK, they're no longer the preferred method. Furthermore, since the plan is to remove them from the JDK, on some platforms they may already have been removed.

Avoid "Undocumented Features"

Almost any implementation of the core Java API will include supporting classes and/or packages that aren't themselves part of the core API. Such classes/packages are generally undocumented – although they may provide some quick-and-dirty functionality, they should be avoided since there's a good chance they won't be available on other platforms.

Similarly, your applications shouldn't depend on the implementation details of any one particular Java implementation. For example, use of the AWT component peer interfaces is documented as being "For use by AWT implementers." Peer classes are highly platform-specific. As a portable application makes use of the AWT rather than implementing it, you should avoid using peer classes in your application.

Follow Any Applicable API Protocols

Certain methods in the core API must be called or implemented in a certain pattern. By skipping over some methods or calling them out of sequence, you may write syntactically correct code, which is highly nonportable.

For example, the 1.1 JDK introduced a new AWT event model. According to the new documentation, the results may be unpredictable if you mix the new event model with the older 1.0 event model. (Refer to "Updating 1.0 source files to 1.1" in the JDK documentation download bundle for details.)

Several of the AWT methods (Component.Paint and Component.Update, in particular) accept a Graphics object as one of their arguments. While you can use and make changes to this object within the method to which it was passed, you shouldn't store a reference to it for later use. Since the AWT implementation is allowed to invalidate any Graphics object after the method returns, storing and later using a reference to it may work on one platform but isn't guaranteed to work on another. This type of problem is particularly difficult to debug.

If any of your classes override the Object.equals method, they should also

override the Object.hashCode method so that for any two objects *x* and *y*:

```
 x.equals(y) implies that
a.hashCode()==b.hashCode()
```

See the JDK java.lang.Object.hashCode documentation for details.

Use System.exit Method Sparingly

You should use this only for abnormal termination of your application. Generally, you should terminate your application by stopping all nondaemon threads. This is often as simple as returning from the main method in your application.

Since the System.exit method forces termination of all threads in the JVM, it may, for example, destroy windows containing user data without giving users a chance to save their work.

Adhere to Good Practices with Thread Scheduling and Synchronization

As with any multithreaded language, don't rely on priorities or luck to synchronize threads. Thread scheduling may differ on different platforms, and even between different machines running the same platform.

Before using Java's multithreaded capabilities, read and understand the JDK documentation for the java.lang.Thread class and the java.lang.Runnable interface. Although Java does simplify multithreaded application development, it's still somewhat of an advanced topic and many of the thread synchronization issues still exist. Doug Lea's book, *Concurrent Programming in Java*, covers this subject in depth.

OS Differences and Limitations

Avoid Native Methods

Occasionally it may appear tempting to use native code to implement certain functionality. However, the native code is by its very nature platform-dependent. Although the rest of your Java application may be completely platform-independent, the fact that you're using native code means that your entire application is also platform-dependent.

Before using native code, consider the following alternatives:

- Define a simple server that provides the necessary functionality, then write your Java application as a client of that server.
- Implement the functionality of the native method(s) in Java.

You may think that confining the native methods to a single class and providing an implementation of that class for every Java platform is a sufficient workaround. This is actually a poor "solution" since the number of Java platforms is ever-increasing. In addition, some Java platforms have no ability to execute native code.

Exercise Caution When Using Runtime.exec

Although the java.lang.Runtime.exec methods provide a means to execute other applications on the system in a seemingly platform-independent way, you should use the Runtime.exec methods with caution. While these methods are part of the core API, the contents of the string arguments passed to these methods are generally platform-specific.

If you intend to use the Runtime.exec methods in your application, then you should provide a way in which the user can specify the exact command strings. Ideally this would be done through a GUI. However, if the command(s) rarely need changing, you can prompt the user to enter them during installation, then store them in a properties file for later use.

Do Not Use Hardwired Platform-Specific Constants

The core Java API provides several ways to help you write a platform-independent application. For example, instead of printing strings with embedded carriage returns and/or line feeds, use the System.println method to print a string followed by an end-of-line character. Alternatively, use System.getProperty("line.separator") to retrieve the line separator for the current platform.

The AWT also provides ways to help you create a platform-independent GUI. Details of these features are described in a separate section below.

Issues Particular to Command-Line Programs and Command-Line Processing

Not all Java platforms support the notion of standard input or standard output streams. Command line programs that use System.in, System.out or System.err may not run under all platforms. Consider implementing a GUI to provide the same functionality.

Even with those platforms that support command line programs, command line processing isn't consistent across these platforms. Although the most portable solution is not to use the command line, this may not be acceptable for programs that need to be executed from within a script. In this case consider using the POSIX convention in which command line options are indicated with a leading dash. As an alternative, you also may want to consider reading the options from a properties file and/or providing a GUI.

Do Not Assume Support for Rendering Unicode Characters

Since not all platforms can display all Unicode characters, use only ASCII characters for the default text for messages, menus, buttons and labels. It's acceptable to use non-ASCII characters in localization resources and in text entered by the user.

Make No Assumptions About the Hostname Format

The java.net.InetAddress.getHostName method returns the fully qualified host name for the InetAddress object. The format of the String returned by getHostName is, however, platform-dependent. On some platforms it'll contain a fully qualified domain name, yet in others it'll contain only the host part of that name.

If your application is merely displaying the resulting hostname, this is unlikely to cause a problem. However, when the name is passed to other systems or applications, it may be best to provide the IP address in addition to the hostname. Note that the IP address is available using the InetAddress.getAddress or InetAddress.getHostAddress methods.

Reading and Writing Files

Do Not Hard-Code File Paths

Hard-coded filenames, and especially hard-coded file paths (a directory name followed by the filename), frequently present portability problems in any language. You can address this problem in Java by using java.lang.System.getProperties("file.separator") to retrieve the file/path separator or using a java.awt.FileDialog to prompt the user for a filename.

Alternatively, to dynamically construct a path and filename in a platform-independent way, use one of the java.io.File constructors. Be aware that even the format of an absolute path differs between platforms. For example, on DOS and Windows-based platforms, an absolute path typically begins with a letter followed by a colon. Under UNIX, absolute paths begin with a forward slash: "/".

Do Not Hard-Code Line Termination Characters

A common problem when transferring text files between different operating systems occurs because of the different ways the platforms represent an end-of-line sequence. Some platforms, most notably DOS and Windows 3.1/95/98/NT, use the character sequence "\r\n" (a carriage return followed by a linefeed), whereas other systems – namely, most varieties of UNIX – use a single linefeed ("\n"). Others may use a single carriage return character ("\r").

Additionally, though Java internally uses Unicode characters for text, different platforms have different internal representations.

JDK 1.1 provides several methods to help address both of these problems. To output text in a platform-independent manner, use any of Java's println methods. These methods output the end-of-line character sequence appropriate for the platform on which your Java application is running. If you really need access to the appropriate end-of-line sequence for the current platform, use the value returned by java.lang.System.getProperty("line.separator").

Similarly, to input text files, use the java.io.BufferedReader.readLine method. This reads a line of text terminated by a line feed ("\n"), a carriage return ("\r") or a carriage return followed immediately by a linefeed, and returns the contents of the line, excluding any line-termination characters.

Be Aware of the Maximum Length of Filenames

All platforms have some maximum length on valid filenames. This may be more of an issue during installation of your application, especially when using the inner classes of JDK 1.1 (which concatenate the class and inner class names to come up with the resulting filename). However, it can also cause problems at runtime in that the JVM may have trouble locating some of the required classes.

One workaround to this is to package your classes into a Java archive (JAR). Alternatively, a ZIP file will work if you're developing for JDK 1.0.

Observe Strict Case Distinctions on All Platforms

Some platforms – most notably DOS and the Microsoft Windows platforms – ignore case when comparing filenames. However, don't let this affect your development efforts. Always use the correct case for class names and filenames throughout your code. That way, your Java application will run as intended on a wider range of platforms.

Combining all your classes into a JAR or ZIP file, as discussed above, also helps preserve the case of filenames.

Avoid "Special" Filenames

Some filenames have a "special" meaning on certain platforms. For example, in DOS and Microsoft Windows platforms, "LPT1:" refers to the first printer port and "CON:" refers to the console. Similarly, under UNIX, /dev/null is the name of a special device that simply absorbs all output directed to it. In other words, don't try saving anything that you may later want to retrieve to a file with this name.

Be aware of these and other "special" filenames on the different Java platforms.

Designing Your GUI

Don't Mix Event Models

Don't mix the newer JDK 1.1 event model with the older JDK 1.0 event model – the results may be unpredictable if you do. In particular, you may achieve the intended results on a single platform with a particular version of the JVM, but this same code may give different – or unexpected – results on another platform or with a different JVM.

The JDK 1.1 documentation download bundle contains details about upgrading in the section titled "Updating 1.0 source files to 1.1." Although it may be tempting when upgrading your application from JDK 1.0 to 1.1 to do this piece by piece, resist this temptation. Bite the bullet and do the entire upgrade in one step.

Use Layout Managers for Sizing Elements

Don't hard-code the sizes or positions of any GUI components. Their exact size is almost guaranteed to differ between platforms, as will the size of the screen and default windows.

Break the habit of laying out components by size and position, and learn how to use Java's layout managers effectively to achieve the desired results.

Blend Your GUI with the Desktop

The size of the screen and the number of colors available are likely to be different between platforms – even between users. Additionally, though you can use your own color scheme with particular RGB values, displays can vary considerably as to the exact color rendered by a given RGB value.

To make your colors blend in with the user's desktop, you can use colors from the java.awt.SystemColor class. Alternatively, you may want to provide the user some way of customizing the appearance – particularly the colors and fonts – of your application, either through an Options dialog, a properties file or both.

Rarely should you need to obtain the screen resolution, but if you really have to, you can use the java.awt.Toolkit.getScreenResolution method.

Don't Assume Existence of Nonstandard Fonts

The availability and size of fonts varies from platform to platform and even from machine to machine depending on installation.

Don't hard-code font sizes. Let text components take on their default size using an appropriate layout manager, and use java.awt.FontMetrics.stringWidth if you really need to determine the actual displayed width of a string.

Before selecting a nonstandard font, be sure to test for its availability and provide a suitable default font in the event that it doesn't exist. The default fonts supported by JDK 1.1 are: "Serif" (usually Times Roman), "SansSerif" (usually Helvetica) and "Monospaced" (usually Courier).

Other Issues

Consider International Issues

The JDK 1.1 provides extensive localization and internationalization features. It's worthwhile familiarizing yourself with them and using them where applicable in your Java application. When you first write your application, you may only expect users from one country, speaking one language, to use it. However, it's much easier to provide support for multiple locales at this point than trying to retrofit these changes at a later date.

For more details on the internationalization and localization features of Java, refer to the section titled "Internationalization" in the JDK 1.1 document download bundle.

Choosing a Distributed Framework

When you begin developing larger applications, you may soon realize the need and/or benefit of distributed objects, or at least the need to separate out parts of the application in more of a client/server role. There are various ways to do this, including writing your own application-specific protocol. The most widely supported distributed object frameworks include RMI, CORBA and DCOM.

CORBA, the most generic of these frameworks, supports not only a wide variety of platforms but also a variety of object development languages. This allows, for example, a Java client to communicate with a C++ CORBA server on a different machine, knowing only the interface published by that object.

RMI is specific to Java and makes communication between distributed Java objects fast and easy. It has a lot in common with CORBA, and there have been talks about possibly merging the RMI specification into CORBA sometime in the future.

Finally, DCOM is Microsoft's Distributed Common Object Model. Like CORBA, it's also language-independent but isn't platform-independent, being supported only on the newer Microsoft Windows operating systems. Furthermore, DCOM isn't supported in the standard Java API. This isn't recommended if you're aiming to develop a truly cross-platform solution.

Loading JDBC Drivers

While Java Database Connectivity has many uses in standalone applications, it's most likely to be used when developing middleware that communicates client requests to a back-end database. It provides a way to allow flexibility in driver loading without having to recompile your code each time you want to change drivers. The java.sql.DriverManager class is responsible for providing this flexibility. The two ways in which a driver can be made available to the DriverManager class are through the jdbc.drivers system property or through a call to java.lang.Class.forName, which causes the driver to be explicitly loaded.

Using JDBC may restrict the portability of your application if not implemented with some forethought. For example, some JDBC drivers may be available only on certain platforms, thereby limiting your application.

To keep your application as portable as possible, provide a means for the user to specify the JDBC driver name. This can be done through either a GUI, the jdbc.drivers system property or a similar properties file.

Developing Your Installation Program

Having developed a platform-independent application, you also may want to consider providing a platform-independent installation program for your application. help with this endeavor, you may want to consider using one of the Java installation programs available. They don't come with the JDK but are available from various third-party vendors. Two of the major players in this market are the InstallShield Java Edition 2 (www.installshield.com/java/) and InstallAnywhere (www.zerog.com) from ZeroG Software.

If you're developing an *n*-tier distributed application, you may also want to consider automating updates to your classes. This can be made easier through the use of an application server, which in many cases may remove the need for a separate installation program.

Certifying Your Application as 100% Pure Java

If you follow the above guidelines for platform-independent application development, you should already have a fairly "pure" application. To test just how pure and to look for possible problem areas you may have missed, check out several free testing tools available from the 100% Pure Java home page at www.javasoft.com/100percent/.

If your application is to be a commercial product or even a high-visibility internal product, you may also want to consider getting it certified as 100% Pure Java and obtaining the logo. Details of this application process are beyond the scope of this article but are available from the 100% Pure Java home page.

Resources:

Updating 1.0 Source Files to 1.1:
http://java.sun.com/products/jdk/1.1/docs/relnotes/update.html

Internationalization support in JDK 1.1:
http://java.sun.com/products/jdk/1.1/docs/guide/intl/

100% Pure Java home page:
www.javasoft.com/100percent/

InstallAnywhere by ZeroG Software:
www.zerog.com/

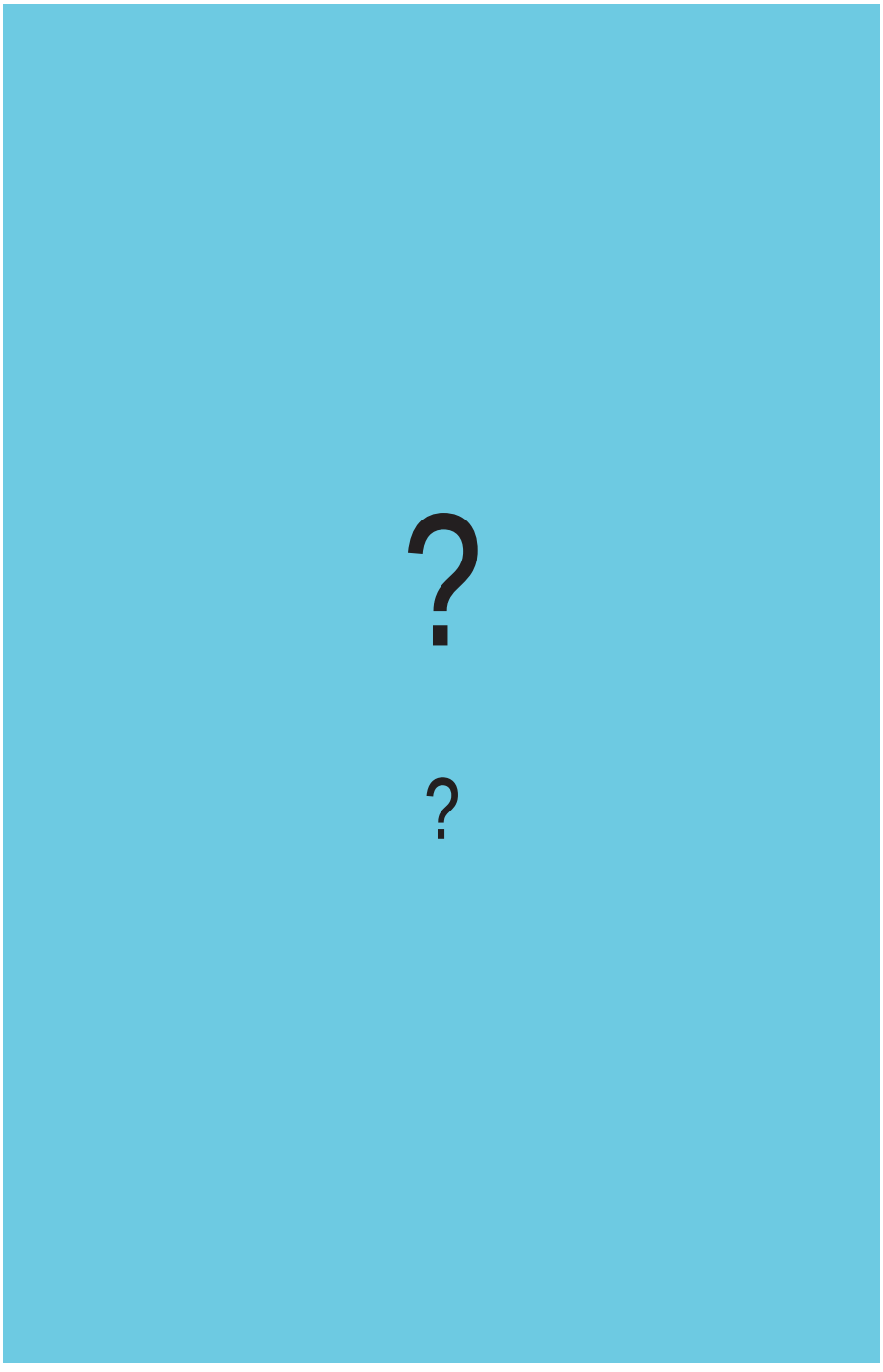
InstallShield Java Edition 2 by InstallShield:
www.installshield.com/java/

The Java Developer Connection (JDC):
http://developer.javasoft.com/

Doug Lea, Concurrent Programming in Java: Design Principles and Patterns, Addison Wesley.

About the Author

Steven Gould, a consultant for Deloitte & Touche Consulting Group/DRT Systems (www.drtsystems.com) in Dallas, develops primarily in C++ and Java under Windows NT and various Unix platforms. He is a Sun Certified Java developer and a Microsoft Certified Solution Developer. Steven can be reached at 73774.2356@compuserve.com.



JSpinner

The strength of these widgets lies in their customizability – and in the lessons learned from effective design

by **Claude Duguay**

It's ironic how sometimes the simplest ideas can turn out to be the most development-intensive. This month's Widget Factory participant is the seemingly modest JSpinner control, which lets you constrain user interface selections by using arrow buttons or up/down keystrokes to increment or decrement values, typically in a field. JSpinner comes with a whole family of siblings to handle numbers, currency, percentage, date, time, lists and custom values. It supports multiple field elements, custom renderers and a compound model to make it all possible.

The table at right shows the various spinner controls we'll be implementing. JSpinner and JSpinnerField are the basic classes. The others stand as good examples of what can be done with a well-designed premise. You'll rarely tend to use JSpinner directly. You'll usually reach for JSpinnerField or one of its subclasses to do specific work. Figure 1 shows the JSpinner family at work.

It's worth noting that this installment of the Widget Factory has a large number of listings, but most of the classes involve only a small amount of code. The main classes, like JSpinnerField, DefaultSpinModel, DefaultSpinRangeModel and DefaultRenderer, do most of the work. The SpinTimeModel and SpinDateModel are relatively uncomplicated, for example, as are various JSpinnerField extensions. Several of the listings are merely interfaces that maintain flexibility in our design, supporting reuse and extensibility.

Architecture

The JSpinner architecture uses several interfaces to maximize flexibility. The SpinModel interface defines the methods required to access the model, which represents values in the JSpinner architecture. The SpinModel contains one or more instances of a class that implements the SpinRangeModel. Listing 1 shows the SpinModel interface. The SpinModel contains ranges that can be accessed by a field ID. These identifiers map directly onto the field identifiers provided by the text Format classes in the Java API. The SpinModel always has an active field and can get and set a list of field IDs. This is important when you need to switch between locales because the subfield order is not always the same. We also support the ChangeListener interface so that views can be updated when the model changes.

The SpinRangeModel is very much like the BoundedRangeModel provided by the Swing API, but it supports the use of decimal values. Listing 2 shows how a SpinRangeModel allows you to set and get the currently selected value, a minimum and maximum value and an increment (extent), and whether the model wraps or not when it hits a boundary. In contrast, the BoundedRangeModel doesn't permit the maximum value ever to be selected and is restricted to integer values. It also knows nothing about wrapping values, so it was necessary to invent a new model for the JSpinner controls. The BoundedRangeModel also supports the ChangeListener interface, which is used by the SpinModel to detect changes.

The SpinRenderer interface is designed to support custom renderers in JSpinner controls. The only required method is called getSpinCellRendererComponent and returns the rendering component. Listing 3 shows the SpinRenderer interface. We expect a reference to a JSpinnerField, the current value object, a flag indicating whether we have the focus, a Format instance and the field identifier for the cur-

The JSpinner Family of Classes

JSpinner
Provides a pair of up- and down-arrow buttons and operates on a SpinModel to increment or decrement values. It handles keyboard events as well. It's up to other components to watch the model and update their views when they receive a ChangeEvent.

JSpinnerField
A basic numerical spinner that uses a DefaultSpinRenderer and DefaultSpinModel to manage a single range of values. A SpinModel may contain more than one SpinRangeModel, but the JSpinnerField requires only one. This is the base class for all the other family members. It takes responsibility for certain mouse events, listening for model changes and focus handling.

JSpinnerPercent
The percentage spinner uses the NumberFormat.getPercentInstance to format a locale-dependent percentage field.

JSpinnerList
The string list spinner uses a ChoiceFormat to format a list selection. This is a simple field, useful for handling small lists of selected options.

JSpinnerCurrency
The currency spinner uses the NumberFormat.getCurrencyInstance to format a locale-dependent currency field. This is a compound field that supports incrementing and decrementing the integer and decimal values independently of each other.

JSpinnerTime
The time spinner uses the DateFormat.getTimeInstance to format a locale-dependent time field. This implementation uses a SpinTimeModel to map the Calendar object onto a SpinModel. It is a compound field with three elements.

JSpinnerDate
The date spinner uses the DateFormat.getDateInstance to format a locale-dependent date field. This implementation uses a SpinDateModel to map the Calendar object onto a SpinModel. It is a compound field with three elements.

JSpinnerColor
This is a color selection spinner example of using a custom SpinRenderer.

rently selected field. We'll cover this in more detail when we implement the DefaultSpinRenderer.

Figure 2 shows the basic relationship between JSpinner, JSpinnerField and the simplest model configuration.

The JSpinnerField class is the parent of all the other widgets implemented in this article. The JSpinner class handles the buttons and up/down activity. It operates directly on the model and can be used for other purposes requiring up/down activities. That's why it was given the big "J" prefix. You can create an arbitrary SpinModel if you like, regardless of whether you use a view to watch the results. Notice that change events from the SpinRangeModel are sent to the SpinModel. This happens automatically and you can watch for the SpinModel events, comfortable in the certainty that you'll never miss any other change events.

Modeling

The SpinModel and SpinRangeModel interfaces need concrete implementations to provide the functionality they expose. The DefaultSpinModel and DefaultSpinRangeModel provide a generic set of capabilities that most of the controls in this article use. Listing 4 shows the DefaultSpinModel class. The internal list of SpinRangeModel instances is maintained by a hashtable that is accessed by a fieldID Integer. A separate, ordered list of fieldIDs is held in a Vector object, as are the registered change listeners. We also keep an activeField value to indicate the currently active SpinRangeModel.

To make life easier, we expose three constructors. The first simply creates an empty model. The second assumes we will use a single SpinRangeModel and takes the same set of arguments, creating the SpinRangeModel automatically. The third constructor assumes that we plan to use two SpinRangeModels and does the same thing, automatically creating both for us. More than two subfields would make the constructor too complicated, so we assume it's just as easy to add fields outside the constructor.

Listing 5 shows the DefaultSpinRangeModel class. The basics are pretty simple, with the constructor accepting each of the arguments and get/set accessors provided for each of the attributes. Worth noting, however, is that the stateChange event is not fired unless the setValueIsAdjusting method is called with a false argument. This is intended to defer the change event to avoid inconsistent states. My implementation is less robust than the Swing BoundedRangeModel, but it works well enough in practice.

The JSpinner class is provided in Listing 6. The constructor expects a SpinModel instance and creates the up and down buttons using the Swing BasicArrowButton class. We register JSpinner as both an ActionListener and a KeyListener to handle increment and decrement operations on the model. Most of the code that acts on the model is in the increment and decrement methods that handle boundary conditions, deciding whether or not to wrap. The JSpinner class also handles right- and left-arrow keystrokes and changes the active field in the SpinModel. To make this work, you have to register JSpinner as a KeyListener form elsewhere, since the buttons never really get the focus.

JSpinnerField

The JSpinnerField (see Listing 7) is the simplest instance of the JSpinner family of controls, but because it's the parent of all the other family members, it's designed to handle general circumstances. As such, it contains more code than most of the other classes in this article. There are three constructors to let us create an empty JSpinnerField, one with a single SpinRangeModel, or one with an arbitrary model, renderer and field Format class. Because we need to refresh the view after construction, we delegate subcomponent creation to an init method. This allows subclasses to control the call to refreshSpinView, which tends to be specific to selected implementations.

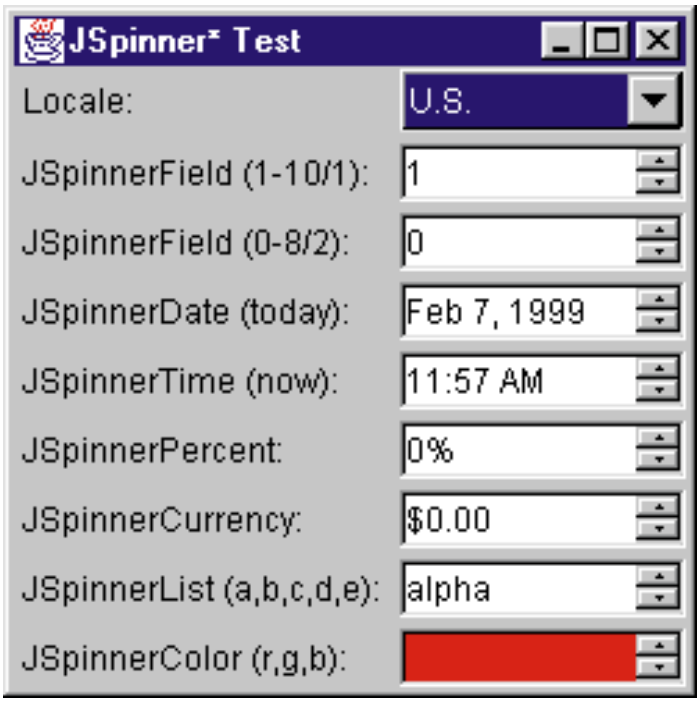


Figure 1: JSpinnerTest with U.S. locale selected

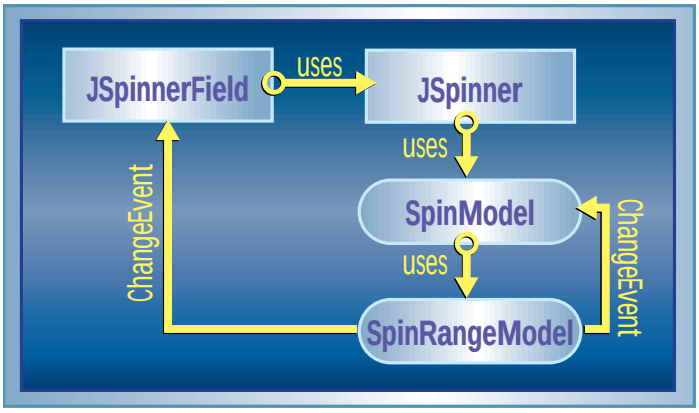


Figure 2: JSpinner, JSpinnerField and model relationships

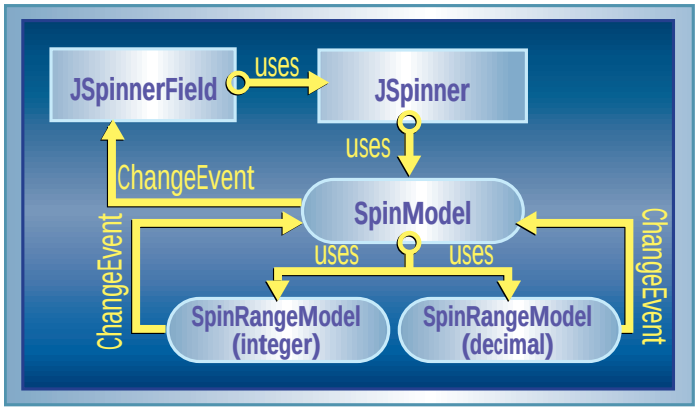
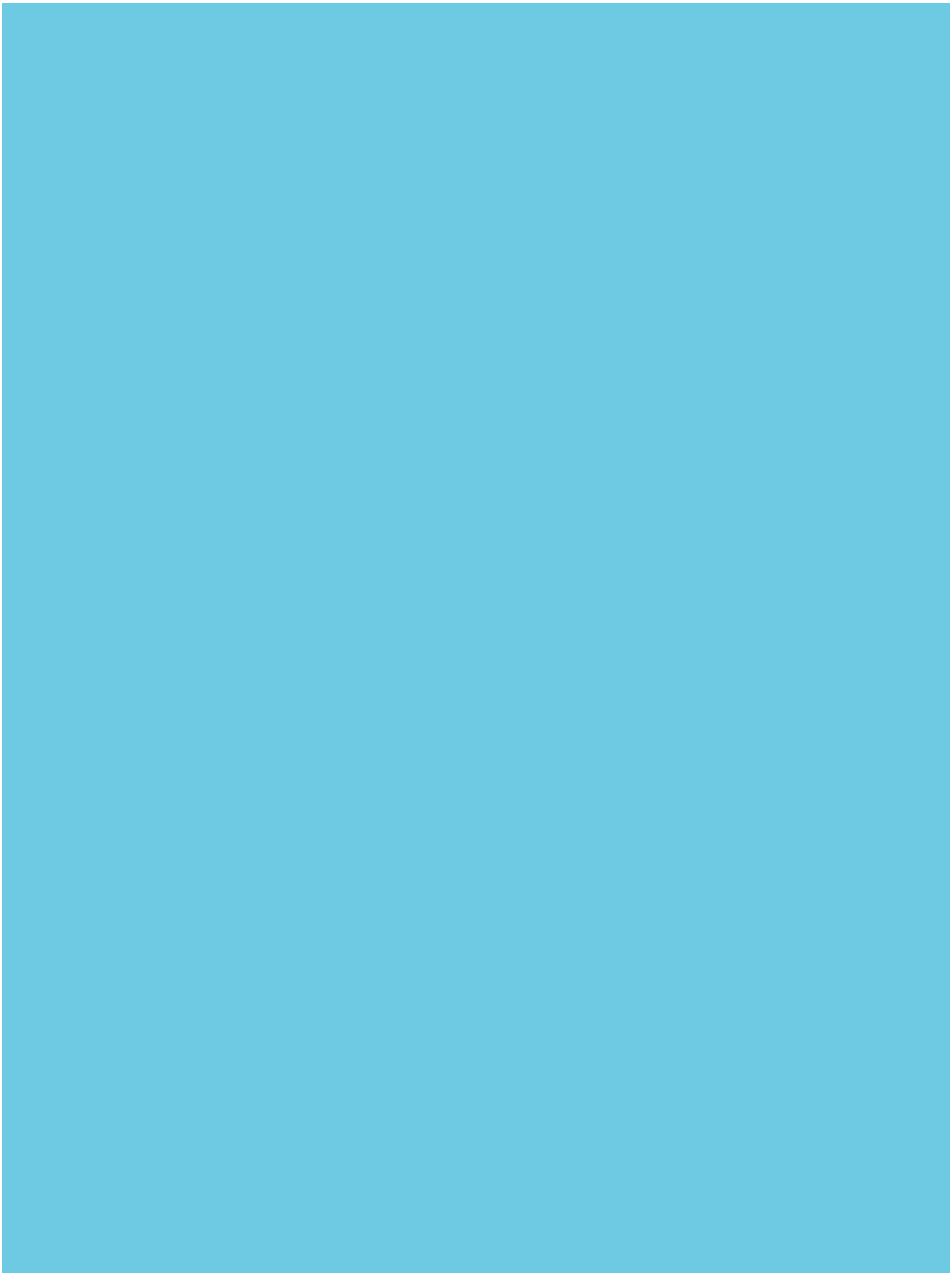


Figure 3: JSpinner, JSpinnerCurrency and model relationships

The setLocale method sets a localized instance of the Format class associated with the control. This is also overridden by child implementations. The updateFieldOrder method is required to determine what the correct field order is for a given locale. This is handled in a separate utility



class called LocaleUtil (see Listing 8), which effectively sorts the fields based on their starting location in the Format object. It also implements a findMouseInField method that lets us determine which field is active when the mouse is clicked on a JSpinnerField.

The JSpinnerField does the rendering through a SpinField class that expects a reference to the JSpinnerField object. As you can see in Listing 9, this class extends JPanel and uses a Swing CellRendererPane to do the actual rendering. The paintComponent call gets the current SpinRenderer and calls its getSpinCellRendererComponent method. We also override the getPreferredSize and getMinimumSize to return the renderer's preferred and minimum sizes.

Listing 10 shows the DefaultSpinRenderer, which extends JTextField and sets the editable flag to false. The getSpinCellRendererComponent method returns the current instance after calling setText with the current value object formatted by the Format instance. It also uses the LocaleUtil.getFieldPosition method to determine what the current selection range is for display if we have the focus.

Simple Extensions

Having just covered the DefaultSpinRenderer, let's take a look at the JSpinnerColor class, which uses a custom renderer to spin through a short set of colors. The ColorSpinRender, shown in Listing 11, is pretty uncomplicated; it ignores most of the getSpinCellRendererComponent arguments and expects to see a Color object as the value. We use a white border to indicate the focus.

Listing 12 shows how simple the JSpinnerColor class is to implement. We extend the JSpinnerField class with a custom spin renderer and a simple model that ranges from zero to the length of our list. We store our list of Color objects in a Vector for convenience. To avoid formatting issues, we declare an empty updateFieldOrder method. We override the refreshSpinView to update the view when the selection changes. All we have to do is get the active model field and range, and set the current list selection based on the active model value. Calling setValue on the SpinField gives the renderer access to the active object.

The JSpinnerList class (see Listing 13) does the same kind of thing without the extra complication of a custom renderer, given that it handles a string list. It overrides the setLocale method because explicit strings are not localizable. The JSpinnerList and JSpinnerColor widgets demonstrate how easy it is to subclass JSpinnerField to create customized behavior. There is no restriction in the data you choose to represent, since both the model and renderer are under your control. Of course, these implementations are not internationalizable unless you use resource bundles or account for it directly in your code.

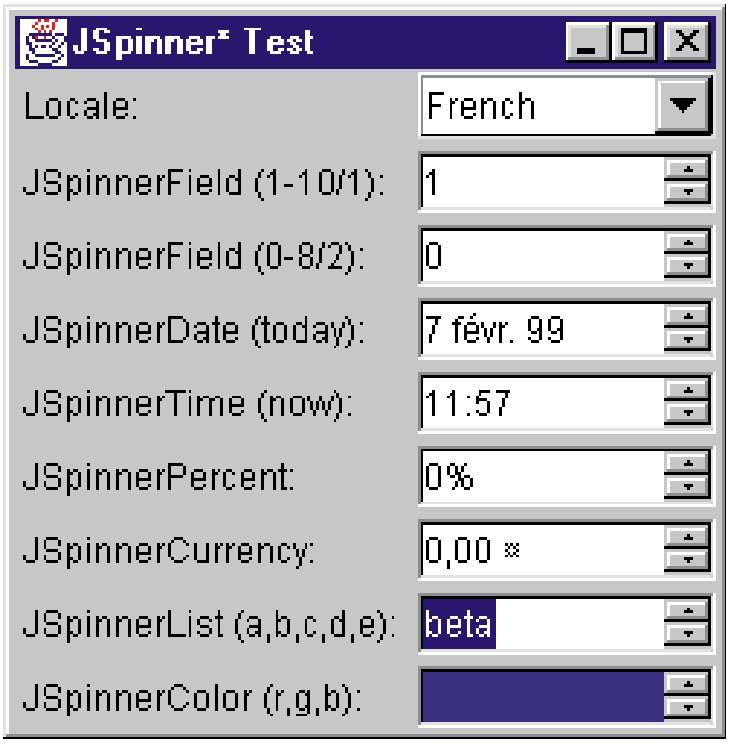


Figure 4: JSpinnerTest with the French locale

Internationalizable Spinners

The last four variations on our theme capitalize on the JSpinnerField infrastructure to handle internationalization through the Java Format class. Listing 14 shows the JSpinnerPercent class, which extends JSpinnerField and implements very little code. The constructor creates a model that uses 0.01 as an increment and sets NumberFormat.getPercentInstance() as the format class. We override setLocale to reset the Format class if necessary.

Listing 15 shows the JSpinnerCurrency control, which is similar but extends the model to use two ranges. One handles the integer portion of our currency field and the other handles the decimal value. We set the NumberFormat.getCurrencyInstance() format in both the constructor and the setLocale methods. The only other thing we need to do is override the refreshSpinView method to properly set the value from the two model ranges. Figure 3 shows the relationship between classes in the JSpinnerCurrency control.

The last two widget variations are only slightly more complicated because they use custom models. Listings 16 and 17 show the TimeSpinModel and DateSpinModel, respectively. Both extend the DefaultSpinModel but override the setRange and getRange methods to control where the values come from. They map the Calendar class values onto the range models as they are being retrieved and manage the Calendar instance that represents the time or date with a couple of accessor methods. You can do something similar to manage your own variations on a SpinModel.

The JSpinnerTime and JSpinnerDate classes are in Listings 18 and 19, respectively. They both override the constructor, setLocale and refreshSpinView methods, but are otherwise unencumbered. Listing 20 shows the JSpinnerTest harness used to test the controls. Figure 4 shows the way it looks when the French locale is selected.

Summary

As you've seen, despite all the listings, JSpinner controls are actually quite simple to use. By providing customizable model and renderer interfaces, the variations you can implement are wide open, as they should be with any open architecture. All the internationalization issues are essentially transparent, thanks to the Format classes provided in the Java API. It would have been easy to write this article around a simpler implementation, but the strength of these widgets is largely in their customizability and in the lessons learned from effective design. I hope you'll agree that even this simple widget turned out to be instructive and worth the investment.

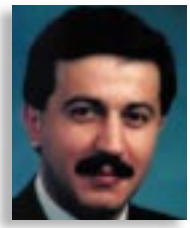
CODE LISTING

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 reach him at Claude@atrieva.com.

claude@atrieva.com



HOW Pro Edition 2.0

by Riverton Software

Utilize reusable business objects and technology components

by **Ed Zebrowski**

Your company has grown beyond anyone's expectations. Internet orders have skyrocketed and management is ecstatic. Everyone is as happy as a bug in a rug. Everyone, that is, except the system administrator. This unexpected explosive growth has caused many new headaches. The server can no longer handle the huge number of hits every day. Raw materials must now be purchased, assembled into products and shipped out faster than the current system can keep track of inventory and purchases. New departments have been added, and new employees in these departments need various levels of access to various areas of the system. All in all, it's a nightmare!

The system administrator can't undertake a project like this on his own. A large development team must be created to tackle such a job. Because of its Web-friendly architecture, Java will be the cornerstone of the new system. What's needed now is a tool that will permit the organization and planning of such a large-scale project. This tool needs to be able to reuse old components, blending them with new ones. The team isn't worried, however, because they know…HOW.

HOW Professional Edition 2.0 for Java is an integrated set of tools that supports object-oriented development of business applications. It works with your favorite tools to enable quick and efficient design and development. It does this by utilizing reusable business objects and technology components. HOW helps you design and build the objects your applications need. These reusable components are then snapped together into applications that can be scaled to meet your needs.

Riverton Software Corporation, the creators of HOW, have boiled down every business application into three elements.

1. ***Database:*** where all the information is stored
2. ***Presentation, or the "front end":*** how clients and staff gain access to the information (e.g., clients placing orders, staff retrieving sales figures)
3. ***Business logic:*** the set of "rules" that govern the flow of data (e.g., minimum orders for clients, employee access to certain records)

In many cases, logic is embedded too deeply into the presentation code. Although this type of scenario is fine for small organizations, it causes problems in larger ones. Difficulty in maintenance and reuse of components, and limitations in both performance and scalability, are but a few of the challenges posed by such a system. Any change in the business rules may result in the need to rewrite the front-end code, which would keep the system down.

HOW presents a better alternative by keeping these logical layers separate. In this "partitioned" architecture, the code for each layer is independent of the other. This presents several advantages:

- ***Flexibility of applications:*** Change of a business rule shouldn't require a change in any of the other tiers.
- ***Reuse of application components:*** The same set of business rules can be applied to several tiers.
- ***Enhancement of team development efforts:*** The code for a user interface, for example, can be changed without affecting the database or other tiers.

HOW enables you to build a foundation of business objects and technology components while completing projects on time and within budget. With HOW you define projects and establish shared libraries to contain new or existing components. HOW brings together a number of tools (see below) to help you achieve this goal. You can then define the additional pieces you need, "snapping" them together as you go.

- ***The repository and its explorer views*** enable the storing and retrieving of information about objects and projects.
- ***The Use Case Builder and Use Case View Builder*** define and summarize how systems are used.
- ***The Business Rule Builder*** defines and classifies business rules.
- ***The Workflow Builder*** provides for modeling business workflows and relating them to the applications that execute portions of the workflow.
- ***The Domain Builder*** enables the creation of class objects and their interrelationships.
- ***The Interaction Builder*** defines how object instances interrelate.

Installing HOW

There are basically three types of environments in which HOW can operate:

HOW Professional Edition 2.0 for Java
Riverton Software Corporation
One Kendall Square, Building 200
Cambridge, MA 02139
Phone: 617 588-0500
Fax: 617 588-0412
E-mail: info@riverton.com
Web: www.riverton.com

HOW Client

HOW Client can be used by application developers to model, generate and run Java applications on a local system. To install HOW on a client machine, the following system requirements apply:

- Pentium 100 MHz processor
- 32 MB of RAM
- VGA 800x600 or better color display
- Two-button mouse or equivalent pointing device
- CD-ROM drive
- 80 MB or more disk space

In the client environment, HOW is designed to accompany other development tools. The following third-party software is necessary:

- Windows 95, version B or higher, or Windows NT, version 4.0 or higher with Service Pack 3
- JDK 1.1
- Any Java IDE or tool
- One of the following data modeling tools: Logic Works' ER win /ERX version 3.0 or 3.5, or Sybase's PowerDesigner version 6.0 or 6.1
- Microsoft Access version 7 (Windows 95) or 8 (Windows 97), for producing reports based on the information in the HOW repository
- Microsoft Word 7.0 (Microsoft Office 95) or 97 (Microsoft Office 97), for producing documents that include objects from the HOW repository
- Microsoft Visual Source Safe, version 5.0, for use with HOW's configuration management features (optional)

HOW Team Repository

HOW can be installed on a network-accessible machine. This will allow multiple developers to access a common repository without affecting each other's work. The hardware requirements for installing HOW Team Development Server software are the same as those for the client, except that only 10 MB of free drive space is required. The only software requirement is that the machine be equipped

with Windows NT 4.0 or higher with Service Pack 3.

HOW Application Server

These are machines that run HOW-generated applications; they require NT 4.0 with Service Pack 3 and any third-party software that the application requires. Windows 95 machines cannot act as application servers, but NT 4.0 machines can be both client and server.

Choose the environment that best suits your needs and desires. Make sure your machine meets the necessary requirements and you'll find installation no more difficult than any other CD-ROM application.

HOW and Java

HOW is used to generate Java classes that are the basis for business applications or applets. These components are 100% Pure Java and can be executed on remote servers by using either DCOM or CORBA. They can then be loaded into your favorite Java development environment for further enhancement.

HOW's Java support comes in the form of a "Java Cartridge." It allows for the following functions:

- Generation of design objects into Java classes, including JavaBeans
- Generation of queries into classes that implement JDBC-embedded SQL statements
- Generation of class methods that allow convenient traversal of associations

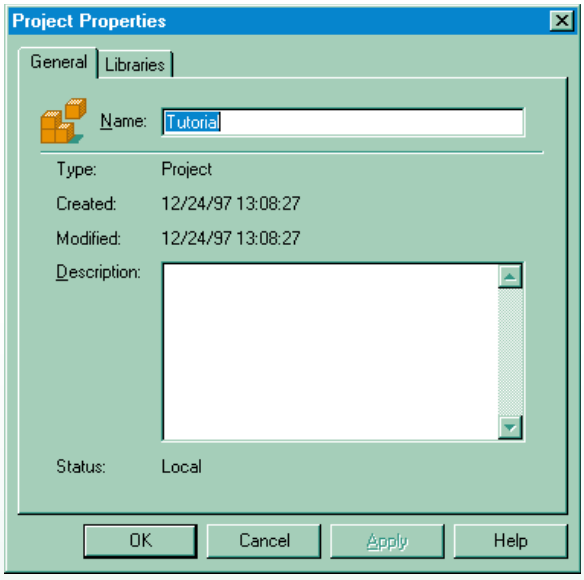


Figure 1: The Projects Properties dialog box is the starting point for creating new projects.

Using HOW: A Brief Example

When HOW is first opened, it displays the Repository explorer. This contains all the objects you define with HOW.

By clicking on File New Project you do just that: create a new project. This will display the Projects Properties dialog box, including two tab pages (see Figure 1). One is Object Info, which provides general information about the project, and the other is Libraries, which is a list of all the libraries used in the project.

In the Name field, type the name of your new project, then choose OK. It's now necessary to create a library that will hold the objects you create as part of the project. To create a new library, perform the following steps:

1. Click on the Libraries tab of the Repository; in the Library pane of the Repository window right-click on the Library1 name.
2. Select Properties from the popup menu; HOW displays the Library Properties dialog box, which displays general information about a library.
3. Click on the name of the library to make it current, then click "Set as Default." When a library is the default, newly created objects will be stored there. Selecting File Import will allow you to import objects into the library.

HOW represents a major breakthrough for developing and organizing business development projects, both large and small. It's relatively easy to use, and works well with your favorite IDE. If you find yourself in the market for such a tool, by all means give it a try.

About the Author

Edward Zebrowski is a technical writer based in the Orlando, Florida, area. Ed runs his own Web development company, ZebraWeb, and can be reached by e-mail at zebra@rock-n-roll.com.

zebra@rock-n-roll.com

VisualAge for Java 2.0

by IBM

The ideal IDE for both beginners and experts

by **Niraj Jetly**

Java is growing by leaps and bounds. New extensions for enterprise development, 2D graphics, servlets, speech, etc., are being added to it. This tremendous growth is making it tough for beginners as well as experts to keep themselves up to date.

With the growing popularity of Java for providing enterprise-wide e-business and e-commerce applications, a sudden requirement for a robust and user-friendly IDE has been felt by developers throughout the Java community.

Whether you're leading a team or working as a developer or an independent consultant, your needs are always the same. Everybody wants to release the project as scheduled, with fewer bugs and good documentation, and without spending hours remembering various APIs.

Until now this was just a dream, but with the release of VisualAge 2.0 (Enterprise Edition) it has become a reality. VisualAge helps you – right from learning Java to building complex applications – with updated documentation. You can test new syntax without compiling the complete project and you can reduce the turnaround time for any bugs by using the powerful debugger. There's no need to remember syntax or use coding help available until the instance variable of a class.

VisualAge has a concept of repository, meaning a single file that stores all other files within itself. If you import a new Java or class file, VisualAge will add that file to its repository. The repository has a built-in source code control system in which all the classes and methods can be versioned.

Feature Set Available with VisualAge 2.0 (Enterprise Edition)

VisualAge for Java comes with many powerful features and utilities to create a complete environment for building large, complex applications. I just looked at the "readme.txt" file and was amazed by the feature list. I feel that all you need is an operating system and VisualAge for Java and you're all set to make even the most complex Java applications. Here's a brief list of features that come with this IDE:

- An integrated development environment with visual programming support for creating Java applets and Swing beans
- Support for a team of programmers to share and maintain source code in a single repository
- Support for JavaDoc output
- Integration with VisualAge TeamConnection, ClearCase and PVCS
- Enterprise Toolkits for AS/400 and Workstation, including high performance compilers for Java and a remote Java debugger
- Enterprise Access Builder for SAP R/3, JDBC, C++, RMI

Installation

I installed VisualAge on a Pentium 166 MHz with 64 MB RAM and a Windows NT 4.0 (service pack 3). The installation CD guided me through the various steps and it was a cakewalk. During the installation, you can select the features to be installed. You can also specify the repository location – either local or on a different server. I installed the repository on the local machine.

Feeling Comfortable

It took me only a couple of minutes to get adjusted to the WorkBench (the main window). The controls are laid out in a very user-friendly manner. Different panes show the Projects, Packages, Classes, Interfaces, Management and Problems. Within five minutes I was ready to create an application using Visual Composition. The creation process was fun – just "drag and drop" and connect, and the application is ready. Within minutes, I created my first complete application without coding.

I love coding, but I hate to cram APIs. So I tested the coding help that the IDE provides. I was amazed that the IDE helped me until I saved the code. Save your code and the code is compiled automatically! No more using the JDK's javac command. It was wonderful, and if you make a mistake while coding, it will save the method with a cross on top of it.

VisualAge's Strengths

The main strengths of VisualAge lie in its coding, debugging, multiuser development and its advanced features.

Coding

- *Coding help* – Type a class, press ctrl+space and you have access to all the available public methods and class variables (see Figure 1). When you select the desired method, you will get the input parameters required for the method. We no longer have to remember all methods in a class.

VisualAge for Java 2.0
IBM Corporation
Armonk, NY
Phone: 800 426-4968
www.software.ibm.com/ad/vajava

Minimum System Requirements:
Windows 95/NT 4.0 with service pack 3, TCP/IP communication protocol, Pentium processor, SVGA 800x600 display, 64 MB RAM, Frames-capable Web browser, Netscape Navigator 4.04, or, Microsoft Internet Explorer 4.01, Java Development Kit (JDK) 1.1 for deploying applications or JDK 1.1.2 for deploying applications using Swing components.

- *Repository and code versioning* – You can maintain multiple versions of classes and methods. Don't worry about messing up with code because you can always revert to any version you want.
- *Scrapbook* – If you need to try out new syntax but don't want to compile the complete project, then use scrapbook. Just type the line of code you want to try with complete context and execute it. You can type the following code and execute :

```
java.util.Vector vec = new java.util.Vec-
tor();
vec.addElement("Item1");
vec.addElement("Item2");
vec.addElement("Item3");
System.out.println("Size of Vector is : " +
vec.size());
```

- *Bookmarks* – If you're working on a huge project with hundreds of classes or need to jump to different classes for source code modifications, you can save time by using bookmarks to scroll between different classes or methods.
- *Incremental Compilation* – This comes in handy no matter what size your project is. It tells you the source of any problem as you save your code.
- *Java docs generation* – Documentation has always been a problem. With the growing popularity of HTML documents, developers have been forced to learn and write documentation in HTML. VisualAge comes to the rescue again. Just type in your comments for a class or a method in plain text and you get all the HTML documentation ready for release.

Debugging

The debugger has been one of the compelling reasons for my shifting my entire development to

VisualAge. I've noticed that the bug turnaround time is significantly reduced if my team uses the VisualAge debugging facility. We can easily step through the code, have a look at all the variables and disable all the breakpoints (so there's no need at all to remove any breakpoint). The strength of the debugger is a real help when you're writing servlets. Run your servlet from the HTML page in the browser and debug the Java code in VisualAge.

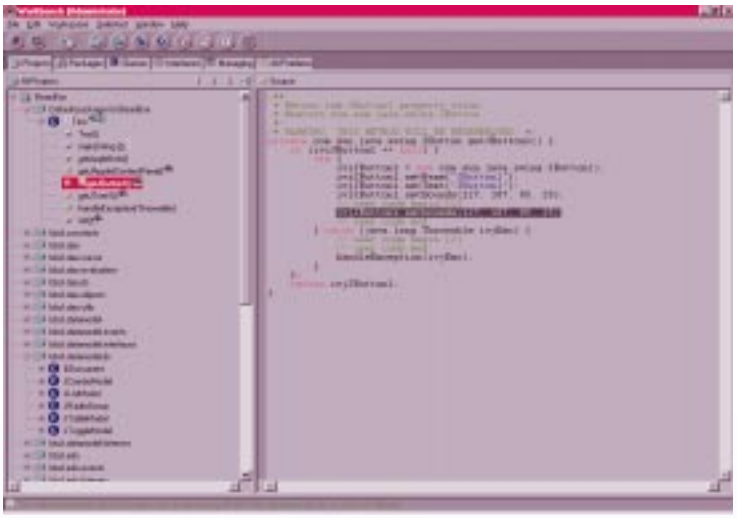


Figure 1: Coding help on typing ctrl+s[ace after a class name

Multiuser Development

VisualAge also supports a multiuser development environment. We can install the repository on one server and the entire development team can act as clients to the central repository.

Advanced Features

VisualAge has built-in support for many features, such as RMI, CORBA, Domino AgentRunner and San Francisco wizard.

Areas of Improvement

I have been addicted to VisualAge ever since version 1.0. While new and powerful features have been added in version 2.0, some of the earlier features are somehow missing.

Design Time Debugging

I do a lot of JavaBeans development, so I have to make lots of customizers for JavaBeans. In version 1.0 I could put a breakpoint in the customizer code and run it in a debugger, but in 2.0 this feature was turned off. This makes the process of debugging a customizer a little more time consuming. I hope that VisualAge 3.0 will include this feature again.

Online Help

This is another area where you may face some problems. I installed VisualAge on Windows NT 4.0 and the online help always worked fine. But if I install the same IDE on Windows 95, the possibility of installing it properly is greatly reduced.

Conclusion

I've been working on Java for more than a year. I tried many IDEs, but all of them had something missing. When I used VisualAge, despite the small problems I mentioned above, I found it is an excellent IDE. The problems I mentioned are very specific and arise only if you're doing highly specialized work (and they were discovered only after more than a year). I'm sticking to VisualAge because its other features far outweigh any problems I've come across. This tool is ideal for both beginners and expert Java developers. It is the definite choice of any project leader or manager who wants to reduce the bug fix turnaround time. VisualAge can be exceptionally useful whether you are working in a single-user or multi-user development.

About the Author

Niraj Jetly is the project leader for the award-winning product, ROAD:BeanBox. He has been working extensively on Java and can be contacted at njetly@specializedSoftware.com.

njetly@specializedSoftware.com

SYS-CON RADIO INTERVIEW

Recorded live at SYS-CON Radio, host Chad Sitler and JDJ Editor-in-Chief, Sean Rhody spoke with Lee Garrison and Alan Armstrong of KL Group

Lee Garrison
Java Product Marketing Manager ***KL Group***

Alan Armstrong
JProbe Product Marketing Manager ***KL Group***

JDJ: *Alan, give us a little overview of JProbe 2.0. What does it offer?*

Armstrong: JProbe is a tool to help developers search and destroy performance bottlenecks and memory hogs in their applications. It's a way to basically profile an application and then afterwards – and even during – take a look at the memory usage characteristics of the application and also what parts of the application are taking the most amount of time….It helps you quickly focus on the parts of your application that you really need to tune, from a memory perspective and also from a performance perspective.

JDJ: *Some new features in 2.0 aren't available in the previous releases. Can you give us an idea about those?*

Armstrong: The whole area of memory debugging has really been improved in version 2.0, and real-time memory debugging wasn't available in the previous release. You know, in a lot of ways people think that memory in Java is not a problem, but that's not actually the case. When you start developing with Java, you find that because of the number of objects that are created and also because of the way that Java handles memory, things can be left allocated and applications can continue to consume memory when you wouldn't think they would because of garbage collection. So what JProbe memory debugger allows you to do is to find those objects that are no longer being used but are still in memory and then zoom in on that area of the application, change the source code and reduce the memory requirements of your application.

Rhody: *A lot of us are used to just running our Java code and seeing the results, and I don't think too many of us think about it. I think you mentioned something that is a viable place to look for these memory leaks. But I guess the biggest question in my mind would be, once you can identify them, what can you do about them? How much of a help can you provide for reducing those kinds of problems?*

Armstrong: Let me give you an example. Because Java is such an object-oriented language, a lot of people tend to use objects basically willy-nilly, without thinking about the penalty that you can pay because of overuse of objects. For example, if you're manipulating simple kinds of data, it may be very much more effective to just use native types such as [in] or double. In fact, in our own JClass product line, we had been using an object for our live table in order to maintain the coordinates of a particular cell in a grid; by using JProbe on our own JClass components, we were able to find out that that was actually a major bottleneck in the component. And we have very experienced object-oriented developers on our staff. So it is in the area of changing the algorithm but also finding out the best ways to use classes in Java.

Rhody: *That sounds pretty good. What are some of the kinds of things you can identify with this? What level of detail do you go into and do you hook into the virtual machine?*

Armstrong: Sure. It's fairly tunable, actually. You can zoom into a very fine granularity of detail or you can back off and just say I am sort of interested in a core screen of information. It depends on what kind of information you're looking for. For example, you may not have any idea at first what part of the application is taking so much time, why this thing is so slow, and so you look at things from a very high level. But then when you narrow it down, you can turn on some of our unique technology to really get down to the line of source code that's causing the problem.

Rhody: *What do you provide in the way of statistics? I mean, what are some of the gross things that you can get on a level like it? Do you tell them how much memory is in use over a given period or that kind of thing?*

Armstrong: It depends on whether you're talking about the memory side or the performance side. On the performance side, there are nine different measurements that you can make on code including the cumulative time spent in a method, the sort of exclusive time spent in the method, the number of times a method is called – and I could go on. JProbe collects all of that information and then allows you to sort things and paint the call graph display, which is a graphical thing that highlights where the time is spent. JProbe will sort things, build the call graph and paint it depending on what kind of criteria you're after.

Rhody: *Can you tie in JProbe to an IDE like JBuilder?*

Armstrong: Absolutely. JBuilder and Semantic Café and also Powersoft and PowerJ – all have opening PIs that we work with.

Rhody: *So you could compile your code and then run it using your VM and then get information and still do everything from their API.*

Armstrong: Absolutely. The interesting thing is we have an excellent relationship with all of the IDE vendors. Right from the word go, JClass components were embedded in the IDEs. You'll see them in Semantic Visual Café, Inprise J-Builder, Sybase PowerJ and IBM Visual Aides. We have very close working relationships with all of those folks, so on the Profiler side we were able to really…to work together well on that. It's something that is very important to us.

Rhody: *What about the present with JClass? What do we have to look forward to?*

Armstrong: For [an answer to] that I'll pass it over to Lee Garrison. He's the JClass product manager.

Garrison: We were talking about a couple of interesting things today with respect to the JClass announcements. The current release of JClass, the 3.6 version, is out and provides common API support across all of the major JDK releases including Java 2.0 with Swing 1.1. That's a real benefit for developers who want to have the flexibility of migrating their code across multiple JDK versions without having to rewrite anything. They can use the JClass 3.6 release and simply recompile with the different versions in JClass 3.6. But we're also working on a new generation of JClass, and the next major release of JClass will be specifically structured for Java 2.0. It will really optimize for the Java 2.0 features, pluggable looking field, drag-and-drop. Components like JClass chart will make use of the 2D API. So it will really be a benefit for developers who want to start new projects and build specifically for Java 2.

Another interesting question that we often get asked and that I might just point out to you is what the impact of Swing has on the other JClass components, and clearly Swing is a good thing. It's providing base-level functionality in the core classes, and things like JTable, the grid in Swing, is great for very simple display of tabular data. But we found that there are quickly limitations to that, particularly in mission-critical or scalable environments. JTable doesn't perform very well beyond a couple hundred rows, and it's not very customizable if you want to change cell appearance on a cell-by-cell basis. That's where things like JClass Live Table as a premium component come into play. It can handle millions of rows with real-time scrolling performance. It can be completely customized on a cell-by-cell basis. It automatically data-binds to JDBC objects as well as native data sources in Visual Café or JBuilder.

Rhody: *If I can dive a little bit into that, what about tying something like that into some more abstract data source like a result set or something else that would be transferable or in some sort of three-tiered fashion? I am doing a lot of distributed work with EJB servers and we typically have the database on the back tier and the EJB in the middle and want to transfer data.*

Garrison: We have a couple of research projects looking into things like that. We haven't stepped into the EJB space yet, but we are certainly aware of it and looking at it. And there is a natural evolution of something like the JClass Data Source Bean, which is a nonvisual hierarchical data object working in those kinds of environments.

Armstrong: If I could just make a comment. The extent that those middle-tier servers are based upon the standard result sets, there should be no reason why JClass wouldn't work with them.

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 Mike Merritt of Sybase and JDJ managing editor Brian Christensen spoke with Don Roedner of Riverton Software.

Mike Merritt
Senior Director ***Sybase***

JDJ: *How is Sybase implementing the Java technology?*

Merritt: Actually, Sybase is implementing the Java technology in all three tiers. We have Java support in the database, in the middle tier and in the development and client site. I started the back-end move, back toward the client, and in the adaptive server family we actually support Java in the database.

We support Java store procedures, which are becoming more prominent across the database vendors. But we also support Java as a data type within tables within the database. This allows you to do powerful things.

Coming into the database, Sybase has award-winning JDBC activities, a type-4 pure Java JDBC implementation that's very well accepted and provides the type-4, meaning that you don't have to install anything. And since it's pure Java, you can download it. You can run it from anywhere.

In the middle tier, Sybase has the Sybase Enterprise Application Server, which we actually started some time before the whole Enterprise JavaBean effort began. We worked with JavaSoft on the Enterprise JavaBeans since the beginning and actually implemented our first production release over a year ago based on a nonpublic version of the EJB spec because the timing was a little bit off. But it was implementing the same level of abstraction and the same intent and purpose of Enterprise JavaBeans that you see today with the JB 1.0, and obviously we'll soon be EJB 1.0 compliant.

To pull that all together, we also have very strong support in the development tool arena with the PowerJ development tool. PowerJ provides the rapid application development environment that people have become used to, and it's tightly integrated with the deployment server with Enterprise Application Servers. You can browse components from within the development environment that are already deployed in the middle tier. You can create new components and deploy them directly into the middle tier with the click of a button, and you can do distributed debugging from the PowerJ development environment.

We're really excited about this integration because over the last few years productivity in application development has grown quite a bit, and when you start moving to distributed architectures, it drops that productivity down.

JDJ: *What can Sybase offer mainstream development that other companies can't?*

Merritt: In addition to the Java support, one strength our platform has is openness in terms of component models. We support more than Java. If you have C++ code, you can't just throw out everything you have; you have to be able to integrate the applications you already have and interface with them so that you can move forward.

To move existing customers and existing applications forward, there are three things people try to leverage. One is their code. One is their skill sets and the third is their data. Now Enterprise Application Server, our adaptor server family, allows them to migrate data. But in terms of moving applications, what we're really talking about is moving the code and the skill sets forward. So by supporting multiple component models and, for example, Enterprise Application Server, it allows people to leverage not only their code – at least to the extent that it's reusable in the new environment – but perhaps more important, the skill set they already have.

We also bring to the table the fact that we're an Enterprise customer. We're used to dealing with the Enterprise capabilities of applications. In fact, we've been doing distributed Enterprise applications for eight years, so there are many Enterprise production applications in Wall Street and all over that are based on Sybase's distributed architectures. This is just the next evolution.

We also have a huge strength in application development with all of our development tools, PowerJ, Power Builder, etc. We not only have the knowledge of what it takes to build and maintain applications in the Enterprise, but the whole paradigm of development.ment with all of our development tools, PowerJ, Power Builder, etc. So we not only have the knowledge of what it takes to build and maintain applications in the Enterprise, but the whole paradigm of development.

JDJ: *Do you have a Web site where our listeners can go to see what Sybase has been up to?*

Merritt: That would be www.sybase.com.

Don Roedner
Director of Marketing
Riverton Software

JDJ: *Can you tell us a little bit about the award-winning product, HOW 2.0?*

Roedner: HOW is a modeling tool, and it won the modeling tools category. We like to say that HOW is a modeling tool that extends the concept of modeling beyond just drawing pictures. In fact, the product's tag line is "Modeling is more than pictures." It's a modeling tool that's focused on business application developers. These are people who aren't necessarily going to have all the skill sets and all the wherewithal to build serious applications with Java. The whole idea behind HOW is to augment their skills to provide some architecture out of the box and to make it easier for mainstream developers to develop large-scale business apps.

JDJ: *And you also concentrate on the Enterprise JavaBeans?*

Roedner: In the next release of HOW for Java [released this past January], we start generating EJB. We start generating session and EJ Beans out of business models that are built in-house. Our direction is to become very focused on Enterprise JavaBeans and EJB app servers and so on. If I had a dream, it would be that in a year we'd be perceived as the de facto standard modeling tool and deployment framework for building EJB-based app servers in the Java space.

JDJ: *What are some other outstanding features of your product?*

Roedner: We like to talk in terms of four really outstanding features, and I've mentioned most of them. One is its ease of use. Because our focus is on the business application developer, we've given a lot of thought to how a modeling tool ought to behave in that kind of context, and that shows up in a lot of ways, such as in the approach we take to present information to developers. It shows up in the documentation of the product and so on.

There are a variety of ways in which this is a very easy-to-use product as compared to other modeling tools. We built it on top of a real repository. This is a clear differentiator. Most of the modeling tools on the market don't have a repository. What this gives us is the ability to strongly support team development. Our first release in the PowerBuilder market has proven to be a real strong selling point for us. In fact, we genuinely support team development, and things like sharing of application components down at the object and component level, which most people can't do.

Another thing that makes us stand out, as I suggested before, is a tight integration. In the case of Java, we work with virtually all the development environments. But that tight integration allows us to generate components into a development environment whether it's PowerBuilder or Visual Basic or Java. It's very smart of those environments. For example, our first-generation product – the PowerBuilder product – is very smart about how PowerBuilder works. We generate nonvisual objects to the middle tier. We're very aware of PowerBuilder foundation class and so on. You can say this is comparable to the BB and Java products.

I think the real differentiator for us is that we ship a framework with the product that allows us to fully integrate individual components from the pictures that developers build in HOW. This framework represents an architecture out of the box, and it's the glue that ties together all the application tiers that developers design and then generate. As a result, people using HOW generate working applications. They push a button and out comes an application that automatically operates in a distributive fashion – it's ready to go.

A Screaming Advertisement?

"Give Some Print Space to Other Products!"

Was this issue (*JDJ* Vol. 4, Issue 3) intended to be a screaming advertisement for SilverStream? No less than three major articles were devoted to the product (two articles and one editorial which was written by a SilverStream employee). That's 10 pages out of less than 70.

If equal time had been devoted to other products, I don't think I'd be so disappointed. SilverStream has lots of competition in the marketplace. Looking at this issue, and at the *JDJ* Web site, one wouldn't know that.

For instance, the most recent review that *JDJ* has done for NetDynamics was for version 3.0. Version 4.0 has been available for over a year and has a greatly improved event model. Using your site as a reference, one would be left to compare the glowing recommendations of the most recent version of SilverStream to a more than two-year-old release of NetDynamics.

Another application server that deserves some mention is WebSphere. It's currently ranked #1 in the application server category of *JDJ*'s Readers Poll, yet I've seen no mention of it.

In short, give some print space to other products. Try not to feature the same product in several articles unless the entire issue is clearly devoted to a single product. It would be far more helpful to those of us who would actually like to see a comparison of products rather than a promotion of a single product.

Grace Frederick
grace@fredtek.com

JDJ FEEDBACK

Thank you for your feedback, but we'd like you to know that we have no vested interest in any particular tool. We have not, and will not, anticipate receiving advertising dollars or any other compensation to run this type of feature.

Our decision was based solely on their unexpectedly impressive performance at our recent "Readers' Choice Polls."

–Editor

Love Those JMaskField Classes!

I love your JMaskField classes, especially since I need to restrict myself to Swing classes. I took it one step further by adding a property called Mask, which is available during design time in Visual Café, and made a new constructor with no parameters. In the setMask method, I executed some of the code from the other constructor. It works beautifully. This way it's also possible to change the mask during runtime.

Joshua Glickman
JGlickman@globaldirectmail.com


E-safety pin man

4 young entrepreneur working from the Scottish sticks has turned-on Silicon Valley with his server technology. He calls it the internet's safety pin.

32 ALAN WILLIAMSON

Diary

Equal Opportunity?

We've been evaluating several products in preparation for embarking on a Java development project. We have yet to find any tools we could, in good conscience, recommend as being the "best" in any category. But, I was wondering if you guys are interested in some "avoid like the plague" votes?

We've got a WHOLE bunch of those!

Terry Grossman
Terry_Grossman@spe.sony.com

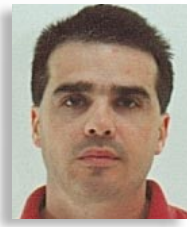
Telling It Like It Is

Concerning Alan Williamson's February "Straight Talking" column (JDJ Vol. 4, Issue 2): best unbiased article I have read...ever...thanks!

James Boice
jboice@williard.com

We're very pleased to have Alan on our roster of columnists for JDJ. We may not always understand his Scottish idioms, but we more than get the gist! Business Age, *the U.K's answer to* Forbes Magazine, *just published an interview with him – kilt and all – about his server technology – what he calls the Internet's "safety pin."*

-Editor



Designing a Web Browser

Using Swing classes to create a better Web browser

by **Pascal Ledru**

A Web browser is often considered a very complex application. In this article I will go over the design and implementation of a simple browser offering a somewhat similar look to Navigator and Internet Explorer. I'll use an existing HTML renderer and several classes of the Swing API. This will give us the opportunity to review some classes introduced in the Swing API, such as JPanel, JButton, JLabel, JOptionPane and JProgressBar. Most of the classes I will use have AWT equivalent classes, but they all provide additional capabilities. For example, JPanel provides the option of displaying a border around the panel. In this article I focus on some of the Swing classes and how to use the additional features they provide as compared to the AWT classes.

A browser such as Navigator is composed of four panels: a panel including buttons controlling the browser, a panel showing the current location and allowing the user to change this location, an HTML renderer and finally, a panel showing whether the browser is loading a page or if the cursor is over a link. I will first describe how to use the HTML renderer and then go over the three remaining panels and present how to set up the underlying look and feel.

HTML Renderer

The tasks of an HTML renderer are basically to parse and render HTML documents loaded from the local file system or over HTTP connections. To implement this simple browser, I have chosen to use the ICE renderer, which is available at www.icesoft.no. Other renderers, such as HotJava and HTMLEditorKit (which is part of the Swing components) and Clue (www.coolclue.com) could also have been used.

The primary class of interest that handles HTML documents is the Browser class, which extends java.awt.Panel through the Document class. I will use the following methods from the Browser class to implement our simple browser:

- gotoLocation(java.lang.String loc) – load a new HTML document
- goBack() – go to the previous document in the history list
- goForward() – go to the next document in the history list
- reload() – reload the current document
- getBackHistory() – get a list of previous documents in the history list
- getForwardHistory() – get a list of next documents in the history list
- getParsingProgress() – get the parsing progress in the current document

In addition, a document fires a PropertyChangeEvent when important status changes occur in the document. PropertyChangeEvents can be monitored by adding a PropertyChangeListener to the Document. The following properties are used:

- statusString – The status property is set by the Document when the mouse is over a link, and by Applets.
- documentBase – This property indicates the URL of a document.
- documentTitle – This property indicates the title of the document.
- parsingProgress – This property returns a string of the form: "frameName charsRead totalChars", where frameName is the name of the current frame, charsRead is the number of characters read and totalChars is the size of the html document (if available), or -1 if it's unknown. Parsing is complete when charsRead is equal to totalChars.

Control Panel

The control panel includes a button to open a dialog for file or URL selection, buttons to move forward, backward and reload the current page, and a button to exit from the browser. Each button contains an icon and text. The button's border is only visible if the cursor rolls over the button. The icon must change whether the button is enabled, disabled or if the cursor rolls over it. In addition, the tool tip text should appear when the cursor stays idle on a button for several seconds. These buttons should be grouped on the left of the control panel, while an animated icon should appear on the right of the panel. Using the JButton class of the Swing API, implementing such buttons is straightforward. Since all the buttons must have this common behavior, I start first by extending the JButton class by the ControlButton class with a constructor that removes the border and focus, and then aligns the text under the icon:

```
class ControlButton extends JButton {
  ControlButton() {
    setBorderPainted(false);
    setFocusPainted(false);
    setHorizontalTextPosition
      (SwingConstants.CENTER);
    setVerticalTextPosition(SwingConstants.BOT-
TOM);
  }
}
```

A mouse listener is used to detect whether the mouse enters or exits a button, and paints the border if necessary. The set of icons used is set by the setIcon, setRolloverIcon and setDisabledIcon methods. The tool tip text is set by the ToolTipText method. Whenever, it's appropriate, the tool tip texts of the Back and Forward buttons are updated with the URL of the previous or next page, respectively.

As noted above, the control panel also contains an icon reflecting whether or not the browser is loading a document. To include an icon in a panel, I use the JLabel class and its setIcon method. In addition, it's also possible to specify the border of a label with the setBorder method. Subsequent calls to the setIcon method are used with either a static or an animated GIF icon. Animated GIF images are handled automatically and no additional coding is required.

To select a new URL, the dialog is implemented using the static method showInputDialog of the JOptionPanel class. JOptionPane is designed to make it easy to pop up standard dialog boxes. It includes several showXXXDialog methods to ask a confirming question, prompt for some input or to inform the user. Only a one-line call is necessary to use these methods:

```
String result;
result = JOptionPane.showInputDialog("Enter
the URL ");
if (result != null) {
  iceBrowser.gotoLocation(result);
}
```

To complete the implementation of the control panel, it's necessary to lay the components down correctly. Two internal panels are used, both of them using FlowLayout. The panel containing the buttons uses a LEFT alignment, while the panel containing the icon uses a RIGHT alignment. These two panels are then included in another panel using GridBagLayout. The first panel uses a WEST alignment, while the second panel uses an EAST alignment. In addition, a non-null weightx value is specified to allow components to grow within the panel.

Location Panel

The location panel includes a label containing an icon, text and a textfield. The label is implemented as is described above, except the default alignment is used. The textfield is implemented using the JTextField class and a KeyListener is used to detect characters typed by the user. Once the ENTER key is pressed, a new document is loaded. Again, GridBagLayout is used to lay down the components within the panel. A non-null weightx value is specified to allow the textfield to grow within the panel.

Link Panel

The link panel includes a label specifying whether or not the browser is loading a document, or if the mouse is over a link. A progress bar is used to indicate how much of a document has actually been loaded. The JLabel class is used to display the label. This label is an update function of the state of the browser. The progress bar is implemented by the JProgressBar class. Three methods of the JProgressBar class are used: setMinimum, setMaximum and setValue. They respectively set the progress bar's minimum, maximum and current values. The minimum value is set to zero while the maximum and current values are updated when the parsingProgress property is received. Once again, GridBagLayout is used to allow the label to grow within the panel.

Putting the Pieces Together

I will now present how properties are handled and how the control, location and link panels are updated. It's first necessary to add a PropertyChangeListener to the browser:

```
iceBrowser = new Browser();
iceBrowser.addPropertyChangeListener(this);

public void propertyChange(Property-
ChangeEvent ev) {
  String prop = ev.getPropertyName();
  if (prop.equals("statusString")) {
    ...
  } else if (prop.equals("documentBase")) {
    ...
  } else if (prop.equals("documentTitle")) {
    ...
  } else if (prop.equals("parsingProgress"))
{
    ...
  }
}
```

A propertyChange method receives updates from the HTML renderer. If the property is statusString, the cursor is updated to a default cursor or to a hand cursor, whether the property's value is an empty string or not. If the property is documentBase, the textfield of the location panel is updated with the new URL. If the property is documentTitle, the browser's title is updated with the new title, the animated icon is displayed, the progress bar is reset and the tool tip texts of the Back and Forward buttons are updated. If the property is parsingProgress, the progress bar and the Link panel's label are updated. When all the document is loaded, the animated icon is switched back to a static icon.

To lay down all the panels, BorderLayout is used, but first an internal panel is used to group the control and location panels. This panel, as well as the remaining panel, are then added to the frame.

Look And Feel

As Swing components do not rely on the GUI manager of the underlying operating system, as AWT does, it is possible to provide a look and feel different than the one of the native windowing system. This can be used by an application to provide a common look and feel on different windowing systems. For example, an application can set up a motif look and feel by the following code:

```
lnf = "com.sun.java.swing.plaf.motif.Moti-
fLookAndFeel";
UIManager.setLookAndFeel(lnfName);
```

Figure 1: Metal look and feel

The simple browser using the metal look and feel is shown in Figure 1.

Conclusion

Using the Swing API, it is possible to painlessly implement a user interface that only requires a major effort using the AWT API. I have only used a few of the classes provided by Swing, but Swing also includes advanced GUI components such as trees and tables. Of course, Swing does not eliminate AWT; if some Swing components can be used instead of their equivalent AWT components, a good knowledge of the layout managers is still needed to implement complex user interfaces. As I mentioned above, I've used the ICE HTML renderer. To allow the capability of switching from one look and feel to another, and to ensure a consistent look and feel between the scrollbar of the HTML renderer and the other components, I have used the version of the renderer implemented with the Swing API. This version is still under development but will parse most simple Web pages. The set of icons was developed by Dean Jones and they are available at www.javalobby.org/jfa/projects/icons. I have borrowed the animated and static icons to show if a document is being loaded from the HotJava browser. Finally, it should be straightforward to extend this simple browser with capabilities such as keeping a list of favorite URLs, and setting up the home page.

About the Author

Pascal Ledru is a software engineer specializing in networking applications at Netran Interactive Commerce. He is also working on his Ph.D. in computer science at the University of Alabama in Huntsville. Pascal may be reached at pledru@worldnet.att.net.

pledru@worldnet.att.net


Clarify Your Code in the Functional Style

Employing your functional code directly in Java

by **Gene Callahan & Robert Dodson**

In creating the HotScheme interpreter (***JDJ*** Vol. 4, Issue 1), we decided to employ functional programming concepts to Java, our implementation language, whenever it was practical. Functional programming has a number of advantages over more traditional procedural code, which we will enumerate below. The common thread uniting these advantages is an attempt to create code that's conceptually transparent. Employing this functional style directly in Java allowed us to define many Scheme functions in Java the same way they would be written in Scheme itself. We wouldn't recommend attempting to program Java in a completely functional style -- the resulting Java code would run poorly and appear bizarre and convoluted to most Java programmers. But we do feel that many of the benefits of functional programming can be brought to Java, if some discernment is used in when to apply the style.

What Is Functional Programming?

The functional programming style is characterized by an extensive (sometimes even exclusive) use of "pure" functions. Input to pure functions comes only from their arguments, not on state information stored in local or global variables. (Of course, Java does not have global variables per se, but public static class variables play essentially the same role.) Pure functions produce no side effects, such as setting a counter or a flag, and are called only for their return value. In the idiom of functional programming, procedures that don't meet these strict criteria are called pseudofunctions. The theory of functional programming has a firm foundation in the theory of mathematical notation, where centuries of experience have confirmed the value of a concise, transparent notation for expressing abstract ideas.

There are six key properties of code written in the functional style, which also hold for mathematical statements in their traditional form. (This list of properties is based on Bruce J. MacLennan's *Functional Programming*, Addison Wesley, 1990.)

1. Value is independent of the evaluation order.

In evaluating a pure function, it doesn't matter whether the arguments are evaluated from left to right, from right to left or even from the middle out. Since each expression depends only on inputs and doesn't create any side effects, no order dependencies can ever arise.

2. Expressions can be evaluated in parallel.

The first property implies that multiple expressions can even be evaluated at the same time, on different processors. Since the final result doesn't depend on evaluation order, neither the programmer nor the compiler need to worry about the sort of complicated dependencies between different procedural statements that make parallel computation so notoriously difficult.

3. Referential transparency.

This means that any expression is independent of its context. If you see f(7, 12) in two different places, you can be sure f will return the same value both times.

4. Absence of side effects.

Fundamentally, this entails avoiding assignments. Assignments are the "go-tos" of data. They create order dependence in code execution, hidden state and nonlocal relationships between pieces of code.

5. Inputs to an operation are obvious from the written form.

Because pure functions don't keep state or use global variables, all of their input appears right there in the function call.

6. Effects of an operation are obvious from the written form.

Since a pure function doesn't set any state or produce any other side effects, to use the return value makes it the direct argument to another function. For example, g(f(7, 12), h(1)). It's clear in this code snippet that f is called to supply g with its first argument, and h is called to supply it with its second.

Why Employ It?

Functional programming can aid in the creation of extremely concise code, which, once you are familiar with the style, is easier to reason with than procedural code. Someone unfamiliar with the style might, when viewing a functional program for the first time, suspect that the programmer has been deliberately cryptic. With a great deal of meaning packed into each line of code, the density can be intimidating. However, 2,500 years of mathematical history show that once the language of mathematical expressions is comprehended, mathematicians can glance at a few symbols and grasp what those untutored won't understand in an hour of verbose explanation. Mastery of the functional style having been achieved, economy of expression, which is inseparable from the properties listed above, will facilitate the comprehension of complex programs tremendously.

This economy of expression means that algorithms are expressed more directly in the functional style than in procedural code. Because the code isn't busy maintaining state and creating side effects, each expression tends to model directly a step in the algorithm itself.

C.S. Peirce pointed out that the form of mathematical expressions is iconic in that the arrangement of terms in an expression is a picture, or icon, of their relationship in our minds (Philosophical Writings of Peirce, Dover, 1940. (reprinted)). Mathematical expressions are the original "visual programming," for these expressions are diagrams of the logical relationship of their terms. So it is with functional programming: the return of a pure function depends only on what you see when looking at its written form, namely, its arguments. The use of its return value is clearly visible from its context, where it will appear as an argument to some other function. In short, a pure function has manifest interfaces. Two examples will illustrate the difference between a pure function and a pseudofunction in this regard.

The following is a pure function. The

return depends only on the value of the parameter *x*.

```
int succ (int x) { return x + 1; }
```

The following is a pseudofunction. The return depends on value of x and the "hidden" value of a.

```
int plusA(int x) { return x + a; }
```

Feeding the function 7 as an argument will not always return the same thing. For you to understand what will happen when plusA() is called, it's not enough to look at the local contexts of the call and the function definition. You must also hunt through the code to where "a" is defined, and then determine where and how its value is set. If the value of "a" depends on some other global or state variable, then you can soon find yourself reading an entire program in order to understand a single statement.

The manifest interfaces of pure functions allow easier proof of correctness. You can imagine how having to follow the state of persistent variables around hundreds of lines of code adds tremendous complexity to a formal proof and pressure for the program to correctly implement its requirements.

This can be expressed more formally by saying that the syntax graph and the dataflow graph for a functional language have identical, treelike structures. Subexpressions that communicate data to each other are always found to be adjacent in the syntax tree. This is not true with procedural code, as assignments allow nonlocal communication – an assignment in one module of a program can have an effect in an entirely different module when the variable is finally used. Note that encapsulation does not make this any less true; just because you're accessing the variable through a function rather than directly doesn't change the degree of nonlocality.

> ***"We do feel that many of the benefits of functional programming can be brought to Java, if some discernment is used in when to apply the style."***

The absence of state makes functional programming inherently "thread safe." There are no persistent variables to worry about locking, and therefore no critical code portions that can't be entered simultaneously by different instances of the same function. Since no pure function maintains state or creates side effects, it is, by definition, safe to execute as many of them in parallel as the environment cares to run.

Java has many features that reduce bug count significantly. It forces you to have accurate arguments and returns, catches or throws for all thrown exceptions and so forth. The absence of pointers eliminates the source of many bugs in C and C++ programs. There is also runtime trapping (array bounds, etc.) so bugs that slip by the compiler are often found in the first test run of the program, rather than lurking silently until a real user hits some peculiar condition and the program blows up.

Even so, functional programming offers an entire other level of "bug protection."

The lack of assignments (in a pure functional program) removes the need to worry about state, and the possibility of bugs arising from failing to account for all possible pseudofunction states disappears. The inherent "thread safety" also gets rid of many of the difficulties inherent in debugging threaded programs that are written in a procedural language.

How to Do It in Java

As we mentioned earlier, we don't view functional programming as a panacea, nor do we recommend its exclusive use, especially when working in a nonfunctional language such as Java. But for those times when you do want to use the functional style in Java, we can recommend the following measures:

- ***Have functional primitives available early on, then use them in higher-level functions designed later.***

For instance, very early in the project we implemented Length(), first(), second() and restl(), all of which operate on a list argument, and return the length, the first item, the second item and a list of all items but the first of that argument, respectively. We also defined a cons() function to build a new list from an object and a list.

Once these functions were done, we would use them frequently. See Listing 1 for an example of how we used each of these primitives several times in a fairly small class.

- ***Avoid assignments.***

Instead of writing:

```
int f4(a, b, c, d) {
  int x = f1(a, b);
  int y = f2(c, d);
  int z = f3(x, y);
  return z;
}
```

Write:

```
int f4(a, b, c, d) {
      return f3(
          f1(a, b),
          f2(c, d));
  }
```

You may find some who claim you are being obscurantist by writing the second version. However, note that it precisely illustrates the use of all the arguments and subcalls in one statement, while the first version has four times as many lines and three new, unnecessary variables.

- ***Use auxiliary functions as another substitute for assignments.***

See Listing 2 for an example.

- ***Use recursion instead of looping:***

```
public final long Length() throws SchemeEx-
ception
  {
// the last object will be #f -- see Length
there!
    return 1L + cdr.Length();
```

Listing 1:

```
// this is the class that
// implements the Scheme 'map' command
class Map extends BuiltIn
{
    public SchemeObject Apply(SchemeObject args, Environment env)
    {
        return map_aux(args.first(), args.second(),
            (args.Length() > 2) ? (args.restl()).restl() : null,
            env);
    }
    private SchemeObject map_aux(SchemeObject f,
        SchemeObject ls, SchemeObject more, Environment env)
    {
        if(more == null) return map_aux1(f, ls, env);
        else              return map_aux_more(f, ls, more, env);
    }
    private SchemeObject map_aux1(SchemeObject f,
        SchemeObject ls, Environment env)
    {
        if(ls.Nullp()) return SchemeObject.False;
        else
            return SchemeObject.cons(
                f.Apply(SchemeObject.cons(ls.first(),
                SchemeObject.False),
                env),
                map_aux1(f, ls.restl(), env));
    }
    private SchemeObject map_aux_more(SchemeObject f,
        SchemeObject ls, SchemeObject more, Environment env)
    {
        if(ls.Nullp()) return SchemeObject.False;
        else
            return SchemeObject.cons(
                f.Apply(
                    SchemeObject.cons(
                        ls.first(),
                        map_aux(First.car, more, null, env)
                    ),
                    env),
                map_aux_more(f, ls.restl(),
                    map_aux(Rest.cdr, more, null, env),
                    env));
    }
}
```

Listing 2:

```
// This code implements the Scheme "begin" syntax, which
//  executes a number of expressions in sequence.
//  Syntax: (begin exp1 …)
public final SchemeObject Apply(SchemeObject args,
    Environment env)
  throws SchemeException
{
  return begin_aux(args, args.Length(), env);
}
private final SchemeObject begin_aux(SchemeObject args, int len,
    Environment env)
  throws SchemeException
{
  switch(len)
  {
    case 0: return False;
    case 1: return args.first().Eval(env);
    default:
      args.first().Eval(env);
// we evaluate the first statement, then call begin_aux
// recursively on the rest of the statements
      return begin_aux(args.restl(), len - 1, env);
  }
}
```

CODE LISTING

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

```
}
```

Can we do functional programming in Java and still take advantage of its object-oriented features? We'll examine this question in terms of polymorphism, inheritance, encapsulation, and the use of class libraries.

Polymorphism is a great aid in writing functional Java. Many of the core functions in the functional style (those for forming and pulling apart lists, for mapping functions over lists and for searching lists) can accept many data types as arguments and may return an object of a different type, depending on what is passed to them. Polymorphism obviates the need to create versions of these functions for all the different permutations of argument and return type.

Inheritance doesn't lose any of its applicability in "functional Java." You can still subclass and override methods in subclasses as long as the new methods are pure functions; then you haven't lost any purity of functional style.

Encapsulation becomes less important in functional programming, since there are fewer state variables and less to encapsulate. However, when you break the functional mold and do employ state variables for performance or code simplicity, encapsulation plays an important role in hiding and localizing the nonfunctional parts of the program.

The standard Java library is quite extensive and is partly responsible for the language's success. It and other third-party libraries can be used in one of two ways in a (mostly) functional Java program. The first possibility is to encapsulate them in a functional layer, then use that layer for the rest of the program. The other is simply to accept them as being among the nonfunctional parts of the program wherever they need to be used.

Results and Trade-offs

Using the functional style may result in some performance loss in some situations, resulting from the increased number of function calls and recursion. In other cases performance may increase because of fewer temporary assignments.

Code may be hard to understand for those unfamiliar with style, a difficulty that is exacerbated by the fact that Java wasn't designed as a functional language. In a language like Scheme, the notation:

```
(print (eval (make term START global_env)))
```

is easier to decipher than the OOP version:

```
SchemeObject.make(term, SchemeObject.START,
  global_env).Eval(global_env).Print();
```

However, we were generally satisfied using the functional style in the HotScheme interpreter. We were able to implement new Scheme functions and even syntactic constructs with remarkably few lines of code, and they were almost always correct the first time we wrote them. Also, to code in this style, we had to think like a Scheme interpreter, so there was a unity of conceptualization between the application and the code that implemented it.

About the Authors

Gene Callahan is the president of St. George Technologies, where he designs and implements Internet projects. He has written for several national and international industry publications. He can be reached at gcallah@erols.com

Robert Dodson is a software developer who writes options trading software in Java and C++ for OTA Limited Partnership. Previous projects include weather analysis software, tactical programs for Navy submarines, and code for electronic shelf labels. He can be reached at rad@ox.com.

gcallah@erols.com rad@ox.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.

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.

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.

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

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.

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.

Sun Certifies Architects To Build Mission-critical Applications Using Java

(Palo Alto, CA) – With the use of the Java programming language on the rise for mission-critical and business-critical applications, Sun is announcing the release of the Java Technology Architecture Planning and Design courseware to train developers to use Java to plan, design and implement solutions for the enterprise. Sun is also offering an exam to certify the architects.

The new certification validates the knowledge architects need to advise clients on the use of Java applications, to identify major architectural issues, and recommend techniques to increase security and performance of their Java applications.

For more information visit http://suned.sun.com.

KL Group Appoints New Director of Business Development

(Somewhere, CA) – Lee Garrison has been promoted to the position of Director of Business Development for KL Group.

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.

Web Application and the Supply Chain

THE GRIND

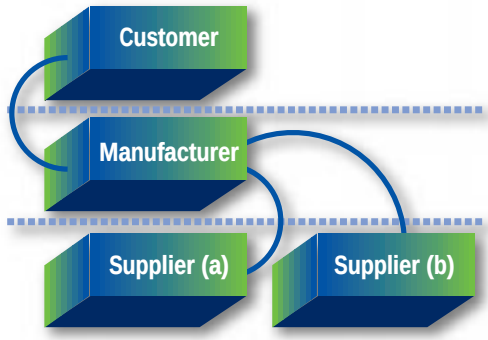
by **Java George**

Most of us think of Web applications as end-user oriented systems, bridging the gap between a data user and data. Indeed, many of the currencies of Web application development are oriented towards this goal.

The Supply Chain as seen through a Web interface

One important evolutionary phase in a new paradigm shift is its use across companies in a way that allows cross-company usage of data and workflow. The Web application and Java paradigm shift is no exception.

In the above example, information flows from suppliers (independent of each other) to a manufacturer, and vice-versa. The same holds true between the manufacturer and the customer.



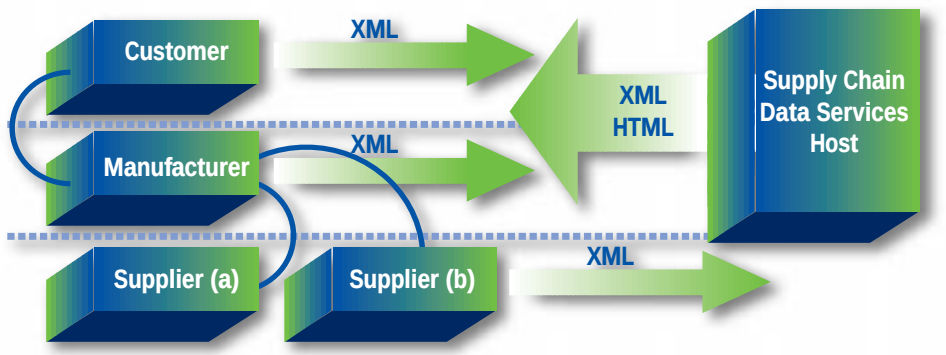
Of course, in the real world a manufacturer will have multiple customers and multiple suppliers so this picture can get very crowded. The supply chain can also become much more challenging to manage and very deep as suppliers have sub-suppliers and sub-manufacturers. Thus the supplier is itself a customer in another chain!

Now what if we put a Java Application Server at each of the supply chain nodes and attached it to the information and workflow sources? What if these Application Servers provided supply chain information across secure and regular Web connections? Many interesting possibilities can emerge.

The Java and open platform technology behind such a scheme is relatively involved. Beyond having a Java Application Server at each distinct supply chain node, the server needs to to deal with the necessary firewalls separating it from the next node in the chain. Remember, these companies are not affiliated or part of a conglomerate. They are simply doing business with each other, and security is very important.

Two distinct forms of interface are needed for transferring data between the chain – a human interface and a computer interface. The human interface is best represented via HTML or Java client, while the computer interface is best represented via XML.

One efficient way of achieving such a view is to transfer all necessary data via XML to a central "host" and bounce back that information as needed in either XML or viewable form.



In this scheme, each of the suppliers, the manufacturer or the customer can get important supply chain information back from the so-called "supply chain data host." The customer can get data on the backlog of the manufacturer on a particular product, the part availability for part of a product being manufactured, etc.

Java and Java Open Appservers in combination with HTML, SSL and XML combine to make this possible across the Web as well as private networks.

???????

??????????

????????

????????

????????

???????

??????

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