HIGHEST PAYING JAVA JOBS

NEW! Java Career Opportunities p.65

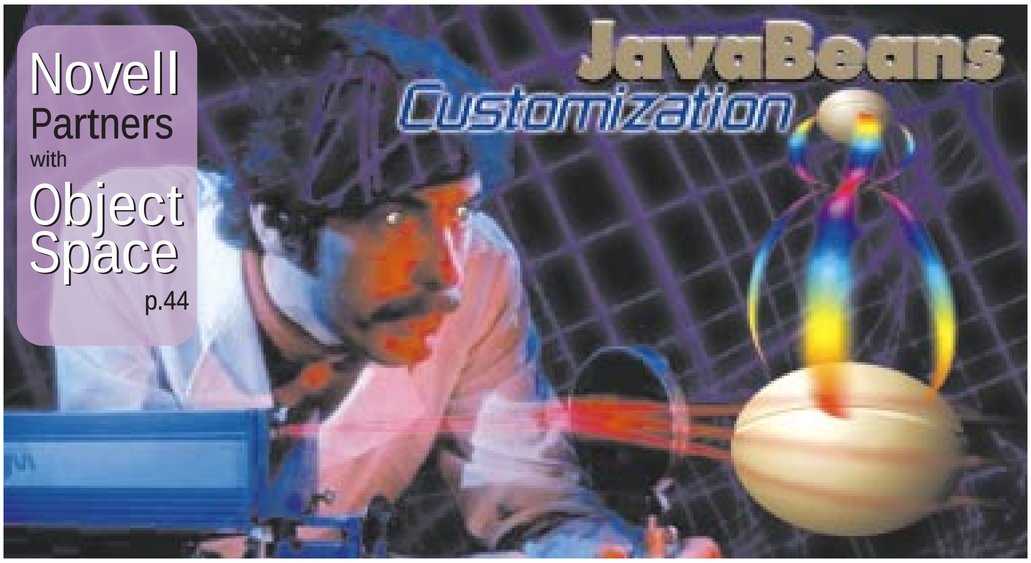
JAVA™ DEVELOPER'S JOURNAL

The World's Leading Java Resource

Volume: 4 Issue: 5, May, 1999

JAVADEVELOPERSJOURNAL.COM

SYS-CON PUBLICATIONS



BEA

weblogic.beasys.com

Protoview

www.protoview.com

Inprise

www.inprise.com/appserver

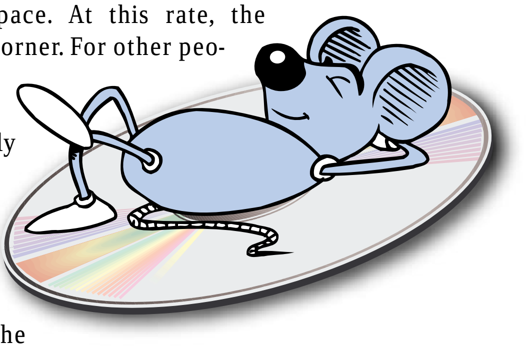
JAVA DEVELOPER'S JOURNAL (ISSN#1087-6944) is published monthly (12 times a year) for $49.00 by SYS-CON Publications, Inc., 39 E. Central Ave., Pearl River, NY 10965-2306. Application to mail at Periodicals Postage rates is pending at Pearl River, NY 10965 *and additional mailing offices.* **POSTMASTER:**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

Sean Rhody, Editor-in-Chief

New Technology

The saying goes, "Build a better mousetrap and the world will beat a path to your door." The world rewards innovation and improvement. It likes new things. This month's focus is on new Java technology. Given the rapid pace of development in our area, that's not quite the oxymoron that it appears. New specifications, new releases, new products come out almost daily. Last year SUN released the 1.0 specification for Enterprise JavaBeans. I can name a dozen products that implement the 1.0 specification at this point, and that number is likely to grow before it shrinks. More recently, the Java 2.0 release became available to the general public. Depending on your viewpoint, this is either old news to you, or too bleeding edge to consider yet.

One of the most invigorating things about Java (for me, at least) is the way the technology advances at such a rapid pace. At this rate, the improvement you need is just around the corner. For other people, this is a difficult issue. Maintenance of code through various releases is a problematic area. As a consultant I frequently see shops that are three, even four releases behind the current release of their primary development tool. Migrating code to the current version is often an arduous process. And yet the pace of the industry requires making the investment in updates. Often, only the current and next-to-last versions of a development package are supported by the vendor (thus ensuring a continuing revenue stream for updates, but somebody has to pay for all the improvements). So far, Java has completely deprecated an existing event model, done major surgery on its windowing toolkit and released a flurry of new specifications. Not bad for a 2.0 product.



Back to EJB for a second. One of the press releases I received recently discussed the use of EJB to provide for the registration of thousands of college students at a major East Coast university. This is one of the first large-scale success stories to date concerning EJB. The OMG group (the CORBA people) has been very active working with SUN and the various EJB vendors to marry CORBA and EJB. This marriage helps secure EJB as the emerging standard for distributed computing.

One of the newest APIs to emerge from SUN is the JTA specification. JTA is Java Transaction API, and is a higher level specification that will supersede Java Transaction Service (JTS), which will remain as a low-level protocol. JTA is tied to the EJB specification and is really an attempt to improve the basic flaw of JTS – no two-phase commit. JTA will include support for the XA protocol, which is the industry standard for heterogeneous transactions.

I don't expect the pace of development to slacken anytime soon. Although the Java language itself is somewhat more mature with Java 2 than it was in its infancy, other tools and products have still to emerge. Testing Java applications is still in the startup phase, with products from SUN emerging to compete with existing products that are scrambling to adapt to Java testing. Source code control is also expanding. In short, we're watching the birth of the Java third-party market.

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.*

Cerebellum

www.cerebellumsoft.com

GUEST EDITORIAL

Alan Williamson

I Have Seen the Future

Last month I came to you as a developer as opposed to a CEO. Well, this time I'm moving up the social ladder and I'm writing in the capacity of a user. I'd like to tell you a little story that scared the living daylights out of me. Continuing on the "Ally McBeal" theme from last time, I have seen a glimpse of the future, and all I can say is, "I am troubled."

I was out walking in the middle of London. It was a beautiful, sunny day. Ever since I upgraded my old analog mobile phone to a nice, shiny Nokia digital phone, I've found myself using it more often. Couple this with the fact that I have one of those headsets for the phone that allows me to stay on it for hours without the fear of blowing my head off with the fallout from the nuclear reactor held to my ear.

I love my Nokia. This is the popular one as seen in the movie *Armageddon* and the one Mulder, from "The X-Files," uses to ask Scully various useless things. It's filled with features. Lots of different features, and a great display. It's everything I never knew I wanted from a phone.

Back to London. I had just finished a call to the office, which inspired another call to be made. I took my phone in hand and moved the cursor up and down the address book with all my contacts in it (which was previously beamed from my PC straight into my phone). However, the whole phone froze. Completely hung on me. The display was still alive, but all my buttons where dead and – even more frightening – my on/off switch was rendered useless.

I had to pop the battery off and back on again to return the machine to life. It then burst back as if nothing had happened. So what went wrong that required me to reboot my phone?

Short answer: Who knows? Chances are, even Nokia doesn't know.

As more of our appliances become more reliant on software, we run the risk of adding more features than beta testers can realistically debug effectively. Are we going to have video recorders hanging in the middle of recording, or toasters crashing as they brown bread? I hope not.

But why is it that, as the features get richer, the bugs exponentially increase. Surely we should be spending more time getting what we have working before adding new bells and whistles. Or is the pressure of continually evolving products so great that companies are taking the risk of bugs in favor of the overall dazzling product?

As you all probably know, Java was not destined for the environment it's running in now. It was originally conceived for the very appliances we're experiencing problems with. Maybe it's about time Java came full circle and completed the job it started out to do. As these devices become far more flexible and functional, the risk of them falling over is also increased.

We don't wish to get to the level of stability that some...mentioning no names...desktop operating systems offer us today. How different the world would be if every time we went to make a cup of coffee we had to debug the kettle!

I just hope we never get to the stage where we have to download the latest patch for our washing machine from the manufacturer's Web site so we can wash our whites brighter than white. Can you imagine how complex shopping for household appliances would become when we're comparing version numbers of vacuum cleaners? A sorry state of affairs. How about virus checkers for appliances? Imagine having the RED day virus, which on a given date washes all your clothes at the highest possible temperature. It may sound far-fetched and silly now, but in five or ten years' time, let's see how silly it is then.

The future is based on software. The hardware is probably going to be the most stable part of the system, but let down by poorly written software. Some say writing bug-free code is possible; some say it isn't. I'm in the camp where you can write software only as good as the environment will allow.

We need a tool that will help us, work with us – and hopefully catch a lot of the silly errors before the end user even gets near it.

Is Java that tool?

About the Author

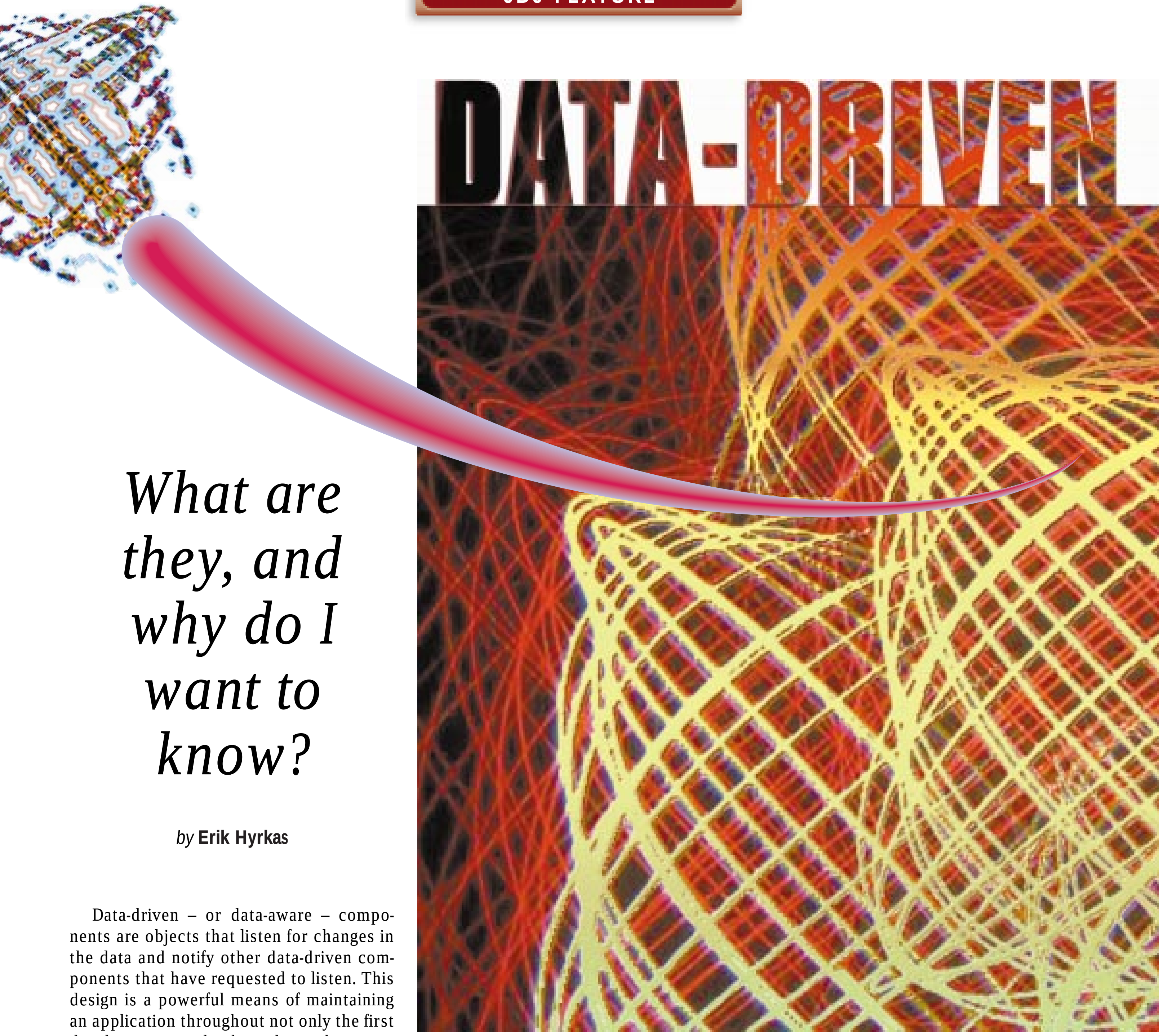
*Alan Williamson, **JDJ**'s "Straight Talking" columnist and a member of the **JDJ** Editorial Board, is CEO of n-ary ltd, a Java consultancy company with offices in Scotland, England and Australia that specializes solely in Java at the server side. Alan is the author of two Java Servlet books and has contributed to the 2.1 Servlet API. He can be*

One Realm

www.

onerealm.com/

jdj



DATA-DRIVEN

What are they, and why do I want to know?

by **Erik Hyrkas**

Data-driven – or data-aware – components are objects that listen for changes in the data and notify other data-driven components that have requested to listen. This design is a powerful means of maintaining an application throughout not only the first development cycle, but also subsequent cycles as your product becomes more robust and refined.

A component can be either a "behind-the-scenes working object" that loads and manipulates data directly or a "user interface tool" that you can click on or type in. Classes that need to send or receive data change notifications should be made into data-driven components.

Some basic overhead is involved in accomplishing this, including superclassing specific core Swing data models that are needed for your application. The alternatives often lack the flexibility to easily maintain your application and the robustness to create a professional program.

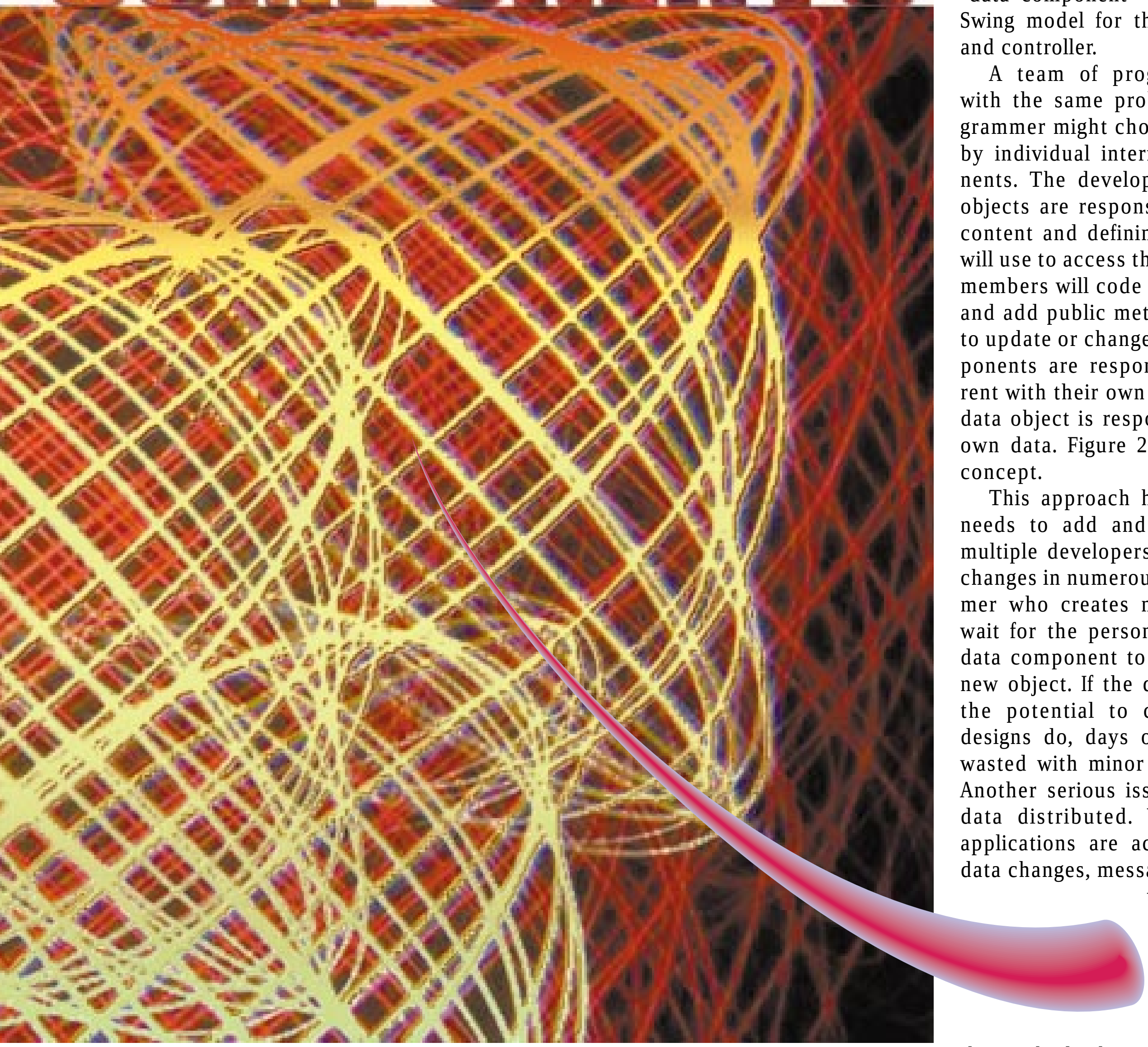
In this article, I'll discuss Swing data models along with a custom data model to handle our business objects and interact with our middleware. Because of the scope of this topic and the infinite requirements possible, a specific code-complete implementation won't be offered. However, I will attempt to describe the design process in-depth enough to empower you with working tools. The overall architecture used in this article is three-tier. I won't cover the use of CORBA, RMI or other middleware implementations for your business logic, but the design is intended to handle each of these multitier designs.

I would like to note that this article isn't directed toward a person unfamiliar with Java or Swing, but rather a developer inexperienced with business application design in Java and Swing.

From Problem to Solution

Every business application requires data to be connected to the user interface. Before you write any code, you have to determine what is appropriate for the task

COMPONENTS



and the team. A solution that can be programmed fairly quickly and offers good performance might be inflexible and time consuming to maintain. I'm not implying that a robust, maintainable solution has to be difficult to design and develop, but it needs to be thought out carefully. One of the most challenging aspects of data-driven components is proper design. Poor implementation can lead to elusive bugs and poor performance.

A programmer who's developing an application alone might choose to create a handful of methods and event handlers in the body of the application that are responsible for populating the user interface and delegating data. Figure 1 demonstrates this simple design. As the application grows, these methods become unwieldy with business logic and presentation preparation. Eventually, the programmer will be faced with the challenges of an inflexible design.

Java is object-oriented, and offers a solution – move data and user interface elements of the application into their own classes. Each user interface component is responsible for presenting its own data properly. The data component is responsible for offering the means for other components to readily access the data. In Swing, the components are a bit more sophisticated and offer us an easy way to separate the data aspect of the visual components. The "data component" of the interface is the Swing model for the specific Swing view and controller.

A team of programmers confronted with the same problem as the lone programmer might choose to divide the work by individual interface and data components. The developers creating the data objects are responsible for managing the content and defining the methods others will use to access the data. The other team members will code the visual components and add public methods for other classes to update or change the content. The components are responsible for staying current with their own data object, and every data object is responsible for tracking its own data. Figure 2 illustrates the design concept.

This approach has a flaw. If the team needs to add and remove components, multiple developers are required to make changes in numerous places. The programmer who creates new components must wait for the person responsible for their data component to add knowledge of the new object. If the design is fluid and has the potential to change, as nearly all designs do, days or even weeks can be wasted with minor component swapping. Another serious issue lies in keeping the data distributed. When multiple client applications are accessing data and the data changes, messages need to be sent to the graphical interface to show the new information. These messages in the above approach would have to travel through the methods that are hard-coded into the data objects.

You need a common means by which the components can communicate. The data objects need to have a common inherited ancestor, along with the base class that gives them their core functionality. Java doesn't support multiple inheritance, like C++, but there is a solution. An interface can be written that puts in methods you know will be present in all your data-aware components. These methods will do the core work of getting and setting information in a way that's compatible with the specific object that contains it.

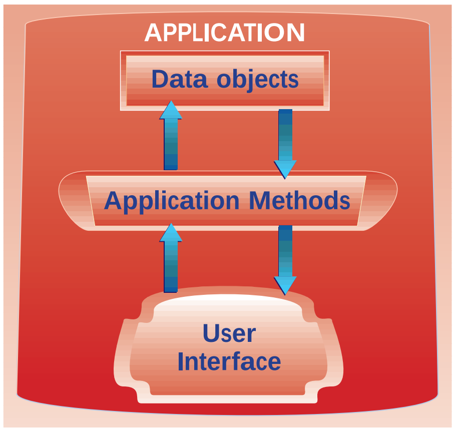


Figure 1: A simple data model

You now wield a common means for creating each data component, but you still need to reduce the work of swapping components in and out, and build a highway for messages to travel. A custom event can trigger listeners to call methods in the event's source. You can use the event to transfer notifications between the data-aware components.

Every object that receives data should be both a listener and an event notifier to its own listeners. The components can inherit an interface to be an event listener and hold an instance of a component to fire events to listening objects.

Your graphical interface is composed of sets of visual components. Swing components have three pieces: model, view and controller. The model is what we're interested in at this point. We can override the core models used by the Swing components in our application so they are data listeners to our custom data model. The custom data model is responsible for creating and storing business objects gathered from the middleware (which might be built around CORBA, RMI, custom socket servers, etc.) and reflecting data messages to the Swing models. An application-specific challenge of overriding the Swing data models involves giving them the constructors and methods that allow you to point at select business objects from the custom data model and choose the proper criteria to narrow that selection.

Business objects are also inherited from a common ancestor that allows our custom data model to manipulate, transfer and inspect them in a consistent means. These data objects are responsible for maintaining relationships with other business classes and displaying information through their own business logic. They are key to our design, since they allow our Swing models to be constructed with specific business constructs in mind.

Now you have the basic concepts of the data model.

Getting Technical

Data-driven components allow for a more flexible and robust means of creating visual components, as well as data objects, without prior knowledge of what the end combination of objects will be. By creating a generic data event model capable of firing events to any component the developer chooses to tie in, the application design becomes more flexible and, over the development cycle, more robust. Both the custom data model and the overridden Swing data models are considered data components. A data component fires events to update, destroy, reload or filter information while the receiving data component can communicate back through the data component interface methods. If you have only a few graphical components and one data object, you never need to change the design, and the data never changes unless the component tells it to; component-driven data is acceptable. However, in a distributed environment where data resides in a central network location and is manipulated by multiple users at multiple locations, the data model must be capable of notifying all the components tied to it. Figure 3 depicts a typical environment for real-time Java applications. If a client application sends an event that updates the data, the middleware must signal the other clients' custom data models to update. CORBA developers refer to this as a callback. In an application not using real-time updating, the responsibility for data synchronization sits on the client side, where it must periodically poll the database. The balance is between concurrency – the number of users who can access the data at one time – and consistency – the accuracy of the data at any moment in time.

Every component that deals with data implements two interfaces that contain the common methods required to communicate. The first interface needs to give us a way of identifying our components as "data-driven" and the common methods to communicate. The definition might begin:

```
public interface dataComponent extends
dataListener {
  public void addDataListener(dataComponent
l);
  public Vector getData();
  public void setData( Vector data );
```

The needs of each application will differ, but the basic ability to get and set data is essential for both our Swing models and our custom data model. You can add methods to accomplish any functionality that other dataComponents might want to perform on each other. For example, you might add methods to reset the component to its initial state or clear all data. The method for addDataListener will be described later, but for now you must accept that you need the ability to add dataListeners.

The other interface your components will require is an event listener. Listing 1 shows an interface that requires your components to define methods for data destruction, updates, loads or filters. The exact events that you need will vary, but a few examples are provided to make it easier to understand. The dataComponent extends the dataListener, since we want to ensure that all components are also listeners.

All of the dataListener methods receive a dataEvent. Listing 2 shows an example definition. The event is a simple object that tells the listener what change has occurred and where. The source is a very useful object since it allows us to invoke methods from the caller. Listing 3 has the code for the dataNotifier. Notice that it won't fire the event if the source doesn't belong to a dataComponent. Thus, you have methods at your disposal from the source object that will exist because only a dataComponent can create a dataEvent. Since Java does not support multiple inheritance, the code for firing events was encapsulated in an object

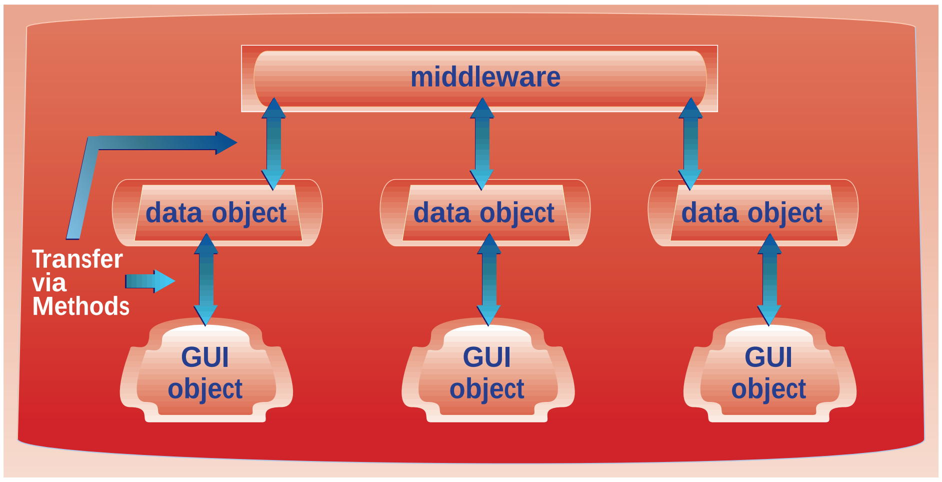


Figure 2: Component-driven data

EnterpriseSoft

www.enterprisesoft.com

that I named a dataNotifier, which can be instantiated in every dataComponent. Rewriting the code in all of the data-aware components would be tedious and error-prone if it needed to be changed later – and just plain time consuming to type in.

The constructors of our Swing data models are an important aspect of our design. We need the ability to choose the type of business objects we're interested in, and what criteria they must meet. Our custom data model is responsible for being a data proxy and distributor of all the business objects that we've already loaded. It also retrieves and distributes new business objects from the middleware.

When building your business objects, take into account that they will eventually need to be turned into displayable information, and it may help to have standard methods that all of your business classes contain. The Swing TableModel is an easy case to remember, but the TreeModel might give you some serious headaches. I recommend adding methods that can retrieve a value at a specified column as well as the column name, and also return the type of that column – very similar to the getColumnClass(), getValueAt(), and getColumnName() methods of the TableModel. If you plan on tackling the TreeModel, plan carefully because this is one of the most challenging of the Swing models.

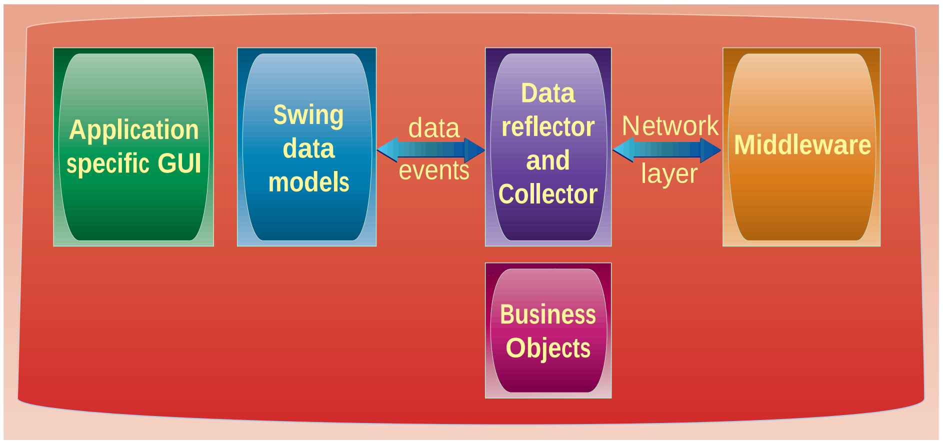


Figure 3: Java applications deployed on a network

Subscribe Today and receive the "JBDJ Digital Edition" FREE at www.JBUILDERJOURNAL.com

GET YOUR OWN!

1 800-513-7111
or subscribe online for faster service
subscribe@sys-con.com

Finally, we need to wire the components together. Every dataComponent requires the method addDataListener. In one central location, where the models are instantiated, you can call addDataListener with another dataComponent – your custom data model, which is a superclass of a dataListener. Your Swing GUI components can instantiate data objects with parameters that specify the business objects we are interested in and the criteria they must meet. You can create very complex designs where custom data objects connect to each other, and multiple overridden Swing models; however, that is beyond the scope of this article.

Basic Design Considerations

When you tackle the design of your application, be certain to review it closely. Often bugs are created because events reflect back from the data object to the event caller, which the programmer didn't anticipate. Check to make sure only those components that need to be listening are connected, since it's possible to generate a considerable amount of traffic with multiple data objects and hundreds of user interface components. Be wary of creating infinite event loops: dataComponent A triggers an event that dataComponent B receives and sends back to dataComponent A, and dataComponent A sends the same event to dataComponent B again. In most cases, you need only one overridden Swing model of a given type, since it's designed to handle data from a generic dataComponent. Your constructors should be doing the work of picking data constraints and converting the business objects to be displayed for your specific needs.

Java is object-oriented. Use it. Design modularity into your application and this will keep bugs from replicating themselves. You might laugh, but I've seen the same bug-ridden code cut and pasted everywhere in an application. Certainly, there is a performance cost to calling methods, but there are severe development costs to creating spaghetti code.

One major performance consideration is object creation. Java is very slow at instantiating new objects. I have seen static declarations of the different events in each component so they don't need to be reallocated with every change. I have not benchmarked this design myself, but I can guess that in very busy components that fire many events, allocating the event objects once might save you significant processing time.

Last Words

Solving the issues brought about by the need for rapid development and creating maintainable code is difficult. "Hard-coding" knowledge of specific components throughout your application is a dangerous task, since it's possible the components might not survive the duration of the project before they're swapped out in favor of better or different objects. Data-driven components handle this by creating a highway for events to travel back and forth without regard to the specified objects tied in. This generic interface allows developers freedom from delays while specific interfaces are built for their individual components.

Finally, data-driven components give us the power to update information from the middleware all the way to the user's screen in a seamless and elegant fashion.

CODE LISTING

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

About the Author

Erik Hyrkas is a full-time consultant in St. Paul, Minnesota, and has been developing Java intranet applications and applets for the business community since the release of the language. He can be reached at ehyrkas@pclink.com.

ehyrkas@pclink.com

Interbase

www.interbase.com/products/demojdj.html

On JavaBeans Customization

Overcome the limitations of the property sheet by building a special-purpose customizer

by **Lawrence Rodrigues**

JavaBeans, now in its third year, is proving to be a powerful component model. Whether it's the Java e-commerce framework or the Java platform for the enterprise, JavaBeans is at the heart of many new and exciting technologies. The JavaBeans model provides a framework to build, customize, run and deploy Java software components. While there are numerous books and articles on JavaBeans, not many of them deal with the customization aspect of JavaBeans.

An object that conforms to the JavaBeans specs is called a JavaBean or, more generally, a bean, which at runtime is like any other object. What distinguishes it from other Java objects is that it can be manipulated with visual builder tools. This process includes customization and connection, which means that you can visually customize beans and connect them to assemble an application.

Let's discuss customization using the Pie Chart bean (part of the visualization bean suite developed for my book, *The Awesome Power of JavaBeans)* shown in Figure 1. If you want to use this bean, you'll probably customize it to suit your application, which may include changing the size and position of the pie as well as the location of the legends. You can customize the bean visually by using a bean builder tool. Once you customize it, you may want to save the configuration for future use by using the Java serialization feature. Sun's BeanBox, which comes with the BDK, is an example of a visual builder tool that allows you to customize, serialize and connect beans. Most of the Java IDEs provide similar tools.

Property Sheets

To customize a bean visually, you need a tool to view and edit its properties – builder tools typically provide property sheets for this purpose. Figure 2 shows the property sheet generated by the BeanBox for the Pie Chart bean. Every time you insert a bean or click on the existing bean in the BeanBox frame, the BeanBox generates a new property sheet. To generate a property sheet, the builder tool obtains the property names and their values from the bean using a mechanism called *introspection*.

Introspection

According to the JavaBean specs, three basic features describe a bean: properties, methods and events. The introspection mechanism analyzes the bean classes to obtain these features. A bean author can help the introspection process by providing a design-time class, BeanInfo, that is specific to a runtime bean class. For example, PieChartBeanInfo is the BeanInfo class for PieChart. Bean authors can provide descriptions of bean features in the BeanInfo class. In the case of a property, its description may include display name, access methods and short text.

How does introspection work? The process first checks the BeanInfo class to obtain a bean's feature (property, method and event) descriptions. If the bean author doesn't provide any information about properties (or other features) in the BeanInfo class, the introspection process uses the low-level reflection API (java.lang.reflect and java.lang.Class) to obtain it. Using these APIs, it probes the bean class to obtain its fields and methods, then deduces a bean's feature descriptions by matching the method signatures with the naming conventions defined by the JavaBeans specs.

The naming conventions for properties define the signatures for setter and getter methods. The former sets the property and the latter reads it. Here is a naming convention defined for a simple property:

- The setter method:
```
public void set<PropertyName>(PropertyType t)
```
- The getter method:
```
public <Property Type> get<PropertyName) {}
```

The following example shows the setter and getter methods for the Graph Color property:

```
public void setGraphColor(Color color){//code }
public Color getGraphColor(){// code}
```

The other types of properties, which include indexed and boolean, have similar naming conventions.

Editing Properties

Once the builder tool obtains the exposable properties, it creates a suitable edit field for each one. For example, the edit field for the plotTitle property in the Pie Chart bean is a text field (TextField or JTextField). Text fields are adequate as long as a property is simple, that is, of Java primitive type (short, int, byte, char, etc.). When a property is of enumerated type, the property sheets typically provide combo boxes (Choice in AWT; JComboBox in JFC).

To edit a property, each property type needs to be associated with a property editor. For simple properties, a property value can be expressed as a text string. The property editor converts the property value obtained from the bean to a text string so as to display it in the text field. When the user inputs a value in the text field, it converts the inputted text string to the corresponding primitive type.

When you start writing real-world beans, you'll realize that Java primitive types aren't always adequate to represent different types of bean attributes. Take the background and foreground colors in the Pie Chart bean, for example. In this case the property type is Color, which is a class in the AWT. You can't enter a color name in the edit field because the text type property editor can't convert a color name to a Color object. What you need is a special property editor that does this conversion. Builder tools often provide custom-written property editors for commonly used properties such as Font and Color. If the property type is a class defined by the bean author, then she/he needs to write a property editor for that class.

Property Sheet Limitations

While the property sheet concept is a vast improvement over manual customization (through programming), it's inadequate to customize complex beans. Here are some limitations:

- Bean authors can't choose the GUI components they like for editing properties. Consider the Pie Chart bean example again. Let's say you want to move the pie to the right. You can do so by entering a value in the Pie Center X Increment field. It's hard to guess the right increment the first time. Before you get the pie position right, you typically enter the increment values a few times. For this kind of operation you'd probably prefer a different component to move the pie.
- Property sheets are static. Edit fields for properties can't be added to or removed from the sheet depending on the value entered in another edit field.
- Notice that the property sheet for the Pie Chart bean displays its properties in a random order. Before you start editing the properties, you may have to mentally group the properties along the functional lines. This problem gets worse when there are a large number of properties. Imagine a bean that has more than a hundred properties – not uncommon in complex beans. You'd have to scroll the property sheet many times to get an overall picture. Even with the Pie Chart bean, which has fewer properties, customizing is easier when properties are grouped together.
- It's not always easy to capture the semantics of the customization process just by looking at the property sheet. Sometimes the process may require properties to be entered in a certain order.
- You often have to configure a bean at runtime. But property sheets are available at design time only because a builder tool is needed to construct the sheet.

Customizers

The JavaBeans model provides an alternative to property sheets. It specifies an interface called Customizer to enable bean authors to implement a bean-specific customizer. Such a customizer can be a simple panel with related properties grouped together or a sophisticated wizard that allows the user to navigate through a hierarchy of screens. Since builder tools recognize any customizer that implements the Customizer interface, it can be launched from any builder tool that has the bean installed.

Customizer Interface

The Customizer interface is simple and has only three methods:

1. ***setObject(Object bean):*** The builder

tool calls this method to pass the target bean instance to the customizer. It is called only once, which is before the builder tool launches the customizer.

2. ***addPropertyChangeListener(PropertyChangeListener listener):*** This method adds a propertyChange event listener. By registering as a listener, the target object(s) (typically the builder tool and property sheet at design time or bean or bean context at runtime) can receive notification and the modified value of the property. When the target object receives a propertyChange event, it retrieves the setter Method object and the property value from the PropertyChangeEvent object. Using this Method object and the property value, the target object then invokes the setter method in the bean to modify the property.
3. ***removePropertyChangeListener (PropertyChangeListener listener):*** This removes a property change event listener.

Builder Tool Interaction

To run the customizer at design time, the builder tool first fetches the customizer class from the BeanInfo class associated with a bean. Using the customizer class, it creates an instance of the customizer object by calling the newInstance() method. The builder tool then embeds the customizer object in a dialog box and launches it.

Before the builder tool launches a customizer, it calls the setObject(Object bean) method in the customizer object and passes the bean instance as the parameter. It also registers for the propertyChange events. The customizer has to fire these events whenever a property is modified. When the customizer fires the event, it sends the new property value encapsulated in the PropertyChangeEvent object.

Designing Customizers

Choosing the right graphical user interface for presenting and editing properties is an important part of the design. Besides being user-friendly and interactive, the GUI should capture the semantics of customization. In other words, it should be intuitive for a user to start using it just by looking at the GUI.

If you intend to provide a customizer for your bean, start the customizer design while you're designing the bean itself. Design your bean's properties to be compatible with the customizer GUI you have in mind. For example, I wanted to have "+" and "-" buttons to increment and decrement the pie size in the Pie Chart bean (see Figure 3). I originally intended to have just one property, Pie Diameter, to represent the pie size. In order to develop a more interactive customizer GUI, I created one more property, Pie Diameter Increment, to represent the increments or decrements relative to Pie Diameter. Thus, when you click the "+" button, the pie diameter is incremented by a certain amount. Likewise, when you click the "-" button, it is decremented by the same amount. With the Pie Position property, I added two more properties: Pie Position X Increment and Pie Position Y Increment to represent the increments (decrements) to the pie position in X direction and Y direction, respectively.

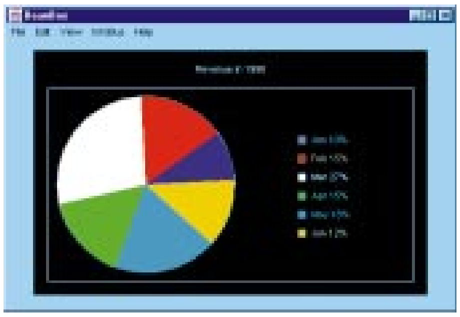
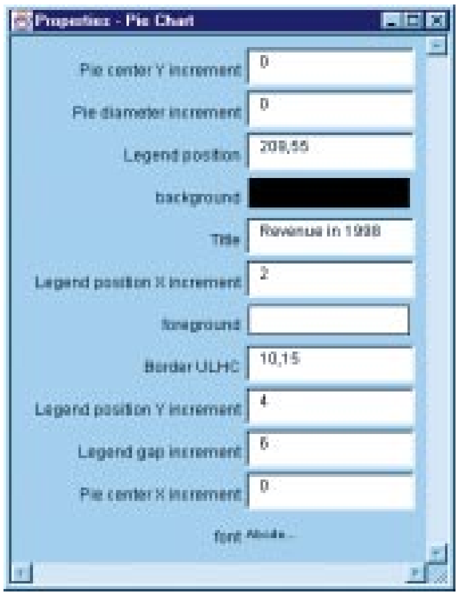
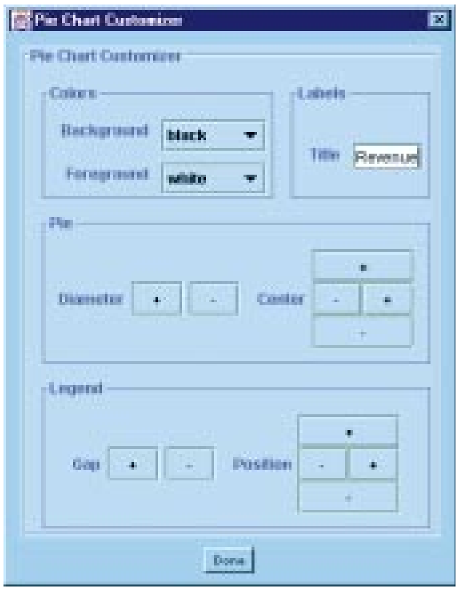


Figure 1: The Pie Chart bean


Properties - Pie Chart

Pie center Y increment	0
Pie diameter increment	0
Legend position	208,55
background	■
Title	Revenue in 1998
Legend position X increment	2
foreground	□
Border ULHC	10,15
Legend position Y increment	4
Legend gap increment	5
Pie center X increment	0
font	Abcde...

Figure 2: The Pie Chart bean property sheet


Pie Chart Customizer

Colors: Background black; Foreground white

Labels: Title Revenue

Figure 3: The Pie Chart bean customizer

Figure 3 shows the Pie Chart bean customizer, which provides a better graphical interface compared to its property sheet. You can adjust the size and position of the pie smoothly by clicking the "+" and "-" buttons. Similarly, you can adjust the position of the legend by clicking the appropriate "+" and "-" buttons.

Implementing a Customizer

Once you design the customizer GUI, implementing the customizer class is just a matter of writing the code for the user interface and reading/writing properties. Building a sophisticated customizer may require a lot of programming, but you can develop it in such a way that most of the code can be reused.

A customizer class has to meet the following requirements to enable builder tools to launch it:

1. ***It must implement the Customizer interface.*** This is to enable the builder tool to recognize the customizer and to obtain the bean instance, which can be passed on to the customizer. Using this bean instance, the customizer can fetch and modify a bean's properties.
2. ***It has to be a subclass (direct or derived) of the java.awt.Component class.*** This is to enable builder tools to embed the customizer in a dialog box.
3. ***It must have a constructor with no arguments.*** This is to enable the builder tool to launch it. If constructors with arguments are allowed, builder tools wouldn't know which constructor to call (in case there are many) or what values to pass as parameters.

The customizer UI will have different types of GUI components depending on your design. The event-handling code for these components should fetch and set property values. Since the customizer object gets the bean instance, it can directly call any public methods in the bean. This means that the customizer can directly invoke the getter and setter method to get and set a property in the bean. While you can build a customizer easily this way, it's not a good solution for the following reasons:

- Property changes in the customizer aren't reflected in the property sheet. In the majority of bean builder tools, when the customizer is launched, the property sheet doesn't go away. Changes made to the property in the customizer have to be reflected in the property sheet as well because the customizer can't access the property sheet directly.
- Code isn't reusable because getter and

KL Group

www.klgroup.com

setter methods are used explicitly.
- Code may require maintenance as you may need to change it in a number of places if a property name or type changes.

An appropriate solution would be to use the reflection API to obtain the getter and setter methods of a property and fire the propertyChange events to set its value. The Pie Chart bean customizer uses this approach.

Pie Chart Customizer

The Pie Chart bean is part of a plotter bean suite. The other beans in this suite include XY Plot, Histogram, Line Chart and Bar Chart beans. The plotter bean customizers have many similarities. To capture the behavior common to all the chart bean customizers, I developed two common classes: CustomizerImpl and CustomizerUtil. The former implements the Customizer interface and the latter has a number of factory methods that create edit fields for different types of properties. The Pie Chart bean customizer class extends the CustomizerImpl class and uses the methods in CustomizerUtil class. The same is true for other plotter beans.

CustomizerImpl Class

As mentioned before, the customizer component is expected to be contained in a dialog box, so a customizer class can't be a subclass of Frame (JFrame) or Dialog (JDialog). Furthermore, the customizer class should be a container because it needs to house a number of components for editing. The logical choice would be to subclass Panel (JPanel). The CustomizerImpl class shown in Listing 1 does just that. It extends JPanel and implements the Customizer interface.

As the listing shows, the CustomizerImpl class has three instance variables:

1. ***bean:*** Holds the target bean instance. Since the setObject() method is called only once, this variable saves the bean instance.
2. ***pcs:*** Holds the PropertyChangeSupport instance. The propertyChange event registration methods call the corresponding methods in pcs.
3. ***title:*** Holds the customizer title. The addNotify() method, which is called only when the component receives a parent, sets the customizer title in the parent.

The instance variables, bean and pcs, are set in the setObject() method. The createGUI() method creates the GUI components and is called only after the target bean instance is received. It is an abstract method, which means that subclasses of CustomizerImpl provide the actual code.

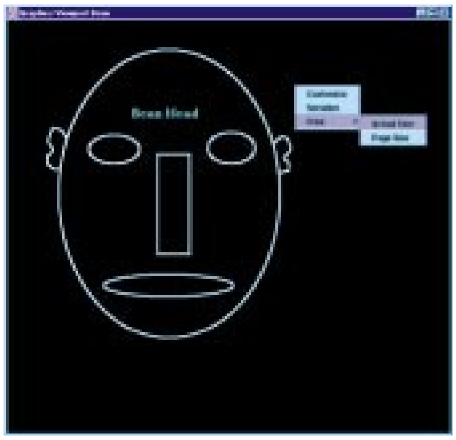


Figure 4: The Graphics Viewport bean

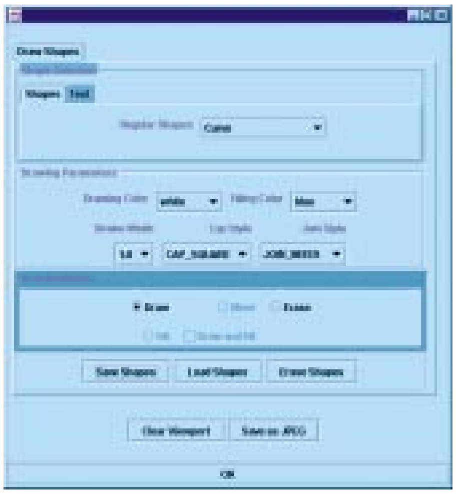


Figure 5: The Graphics Viewport bean customizer

CustomizerUtil Class

The CustomizerUtil class has a number of utility methods that help to create edit field components, which include JTextField and JComboBox. This class uses the reflection API to get and set a property value.

As an example, see Listing 2, which shows the createJTextField() method. This method first constructs a JTextField component. It then obtains the actual property value from the bean by calling the getProperty() method (see Listing 3). Using the property name string, getProperty() method first constructs the Method object for the property's getter method. The Method object executes invoke() to invoke the getter method.

The createJTextField() method also has the code to set the property. When the user types a value and hits Enter, the JTextField fires an action event. To handle this event, createJTextField() registers with the JTextField to receive action events. The event-handling code for the action event is in an anonymous class in which the actionPerformed() method first fetches the text entered by the user from the JTextField and converts the value to an appropriate object. It then calls the firePropertyChangeEvent() method in pcs.

Listing 4 shows the code to create "+" and "-" buttons for incrementing and decrementing a property. This method uses a highly reusable class, IncrButtonAdapter, which is an adapter class for the action event (see Listing 5). An IncrButtonAdapter instance is constructed for each button. The property name and increment intervals are passed as arguments to this constructor. The actionPerformed() method, which is called whenever the increment button is clicked, first fetches the property value from the bean, then adds the increment to this value and calls the firePropertyChangeEvent() in pcs. The IncrButtonAdapter class is used by pie and legend properties in the Pie Chart bean customizer.

PieChartCustomizer Class

Listing 6 shows part of the PieChartCustomizer class, which extends the CustomizerImpl class. It overrides the createGUI() method, which calls createLabelsPanel() and createColorPanel() methods to create labels and color panels. It also calls the createIncrPanel() method to create increment panels for pies and legends. As you can see from the listing, these methods call static methods in the CustomizerUtil class to create components for editing.

Registering the Customizer

The customizer has to be registered for the builder tool to launch it. The BeanDescriptor class is the only class that holds the customizer class. You can set the customizer class by using one of its constructors. The BeanDescriptor class has two constructors:

- **BeanDescriptor(Class beanClass)**
- **BeanDescriptor(Class beanClass, Class customizerClass)**

The second constructor is used to register the customizer. If you use the first constructor, the builder tool assumes that the bean has no customizer. The code to register the customizer goes in the BeanInfo class. The example below shows how the Pie Chart bean customizer is registered. This code is taken from the PieChartBeanInfo class.

```
public BeanDescriptor getBeanDescriptor(){
 BeanDescriptor bd = new
BeanDescriptor(PieChart.class, PieChartCustomiz-
er.class);
 // Other stuff
 return bd;
}
```

A bean's own BeanInfo class is the only place where you can register a customizer. This means that building a BeanInfo class for that bean class is a must. This is because builder tools won't recognize the customizer even if the customizer is regis-

Object Domain

www.objectdomain.com

tered in a bean's superclass BeanInfo.

Customizer at Runtime

Although the availability of customizers at runtime is often desirable, the JavaBeans specification makes no mention of runtime customizers. You can provide this facility, however, by adding a few more methods to your bean. The program that launches the customizer can reside in the bean itself or in the bean context (container).

To launch the customizer, you can use the button, pop-up menu or any suitable component depending on your bean and the application. The Graphics Viewport bean shown in Figure 4 provides a pop-up menu that can be triggered by right-clicking. (You can download this bean from my Web site at www.execpc.com/~larryhr/gvbean.html and use it to draw a variety of shapes and text.) This menu contains the items customize and serialize.

The customizer class itself is a property in the Graphics Viewport bean. This means that the class can be set by the setter method and fetched by the getter method. The bean context or application that uses the Graphics Viewport bean can call these methods to register and launch the customizer.

Listing 7 shows the event-handling code (which is in the Graphics Viewport bean itself) for the Customize menu item. As you can see, the actionPerformed() method first fetches the customizer class and passes it as a parameter to launchCustomizer() to launch the customizer.

Launching the Customizer

Listing 8 shows the launchCustomizer() method, which:

1. Creates an instance of the customizer object
2. Calls setObject() method to pass the bean instance to the customizer
3. Registers for the propertyChange event

In the event-handling method it retrieves the property values from the PropertyChangeEvent object and calls the setProperty() method (see Listing 9). This method uses the Method class to invoke the setter method.

Bean as a Standalone Application

Figure 5 shows the customizer for the Graphics Viewport bean. This customizer is much more comprehensive than the Pie Chart customizer. With it you can set the drawing parameters and also draw shapes and text. Since this customizer can be launched at runtime, you can use the Graphics Viewport bean as a standalone application.

To make a bean run as a standalone application, you need to add the main() method. Starting with Java 2, a JAR file can be made executable. This means that you can run a bean that has the main() method by executing its JAR file. For example, to run the Graphics Viewport bean, just type java -jar grbean.jar.

To make a JAR executable, you need to include an attribute called Main-Class in the JAR manifest file. Following is the example from the grbean.jar manifest file.

```
Main-Class: com.vistech.viewport.GraphicsViewport

Name:
com/vistech/viewport/GraphicsViewport.class
Java-Bean: True
```

This brings up another issue: Should the runtime and design-time customizers be the same? There is nothing to prevent you from having two customizers – one for design time and one for runtime. You can use the same Customizer interface to implement these customizers.

Conclusion

Property sheets are simple and don't require a lot of programming effort. They're inadequate, however, when beans are complex and have a large number of properties. We can overcome most of the limitations of the property sheet by building a special-purpose customizer. The Customizer interface is a simple but powerful API that can be used to develop user-friendly customizers to configure complicated beans. However, writing a good customizer requires a lot of programming effort. Unlike property editors, which are property specific, a customizer is specific to a bean. So if you make changes to the target bean, you may have to modify the customizer as well.

References

1. Rodrigues, L. (1998). *The Awesome Power of JavaBeans*. Manning Publications.
2. Sun Microsystems, JavaBeans API Specification Version 1.01, July 1997.
3. Rodrigues, L. (1997). "Java, The Next Generation: JavaBeans." *Java Developer's Journal*, Vol. 2, Issue 1, January.

About the Author

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

Lawrence Rodrigues, author of The Awesome Power of Java Beans, *is the lead architect at a large industrial automation company. You can reach him through his "Bean man's home page" at www.execpc.com/~larryhr or via e-mail at larryhr@execpc.com.*

larryhr@execpc.com

Flashline

www.flashline.com

Electronic Java

Java – working hand in glove with other computing technologies to offer business solutions in the world of e-commerce

by **Ajit Sagar**

Hello, and welcome to electronic Java! In this column we'll examine the role of Java in the fast-growing world of electronic commerce. We'll also look at how the different components of the Java 2 Platform fit together to create complete enterprise-level e-commerce applications. This column will also focus on how these technologies relate to the world of Java and vice versa.

Before we get into the specifics of how Java contributes to the world of e-commerce, I'd like to set the stage by briefly introducing e-commerce and how it relates to the Internet. Of course, once we get to the Internet, Java's presence is inevitable; as far as Java and the Internet are concerned, when one is present, the role of the other is more or less taken for granted.

E-Commerce and Internet Commerce

E-commerce may be defined as a method of doing business "electronically." This isn't a new concept. For example, EDI has been the main means of conducting business electronically for several years now. What has made a difference in the past few years is the phenomenal growth of the Internet, leading to a revolution in the way business can be conducted electronically. Internet commerce is the new, next-generation, "revolutionize the world as we see it today," whizbang phenomenon that is currently in the process of taking the world by storm. Business surveys estimate that e-commerce–based business will grow to well over a trillion dollars by the year 2002.

In the context of business transactions, Internet commerce is a part of e-commerce that refers to the use of the global Internet for purchase and sale of goods and services. It can be categorized as two types of businesses. The first allows the end customer (the buyer or purchaser) to interact in the transaction by directly buying/ordering goods on the Internet. This necessitates a highly interactive user interface on the client-side architecture used to implement the features of Internet commerce. In the second type, businesses conduct transactions without the involvement of the end users. One business plays the role of purchaser; the other, the role of the seller. Please note that when I mention e-commerce in the discussions offered in this column, I refer to Internet commerce, since Java is primarily applied in the areas of Internet commerce. The development efforts in the e-commerce market today are focused mainly on two fronts:

- ***E-commerce application development:*** Parties involved in this kind of development are using current technologies and frameworks to develop and deploy e-commerce alternatives to current businesses.
- ***Development of e-commerce–enabling frameworks:*** Parties involved in this kind of development are using compatible enabling technologies to develop frameworks that can be used to build e-commerce applications.

Putting together enterprise-level frameworks or applications for enabling commerce on the Internet involves design and

Java Technology	Description	Role in E-Commerce
Java Applets	Programs written in the Java programming language that can be included in an HTML page. The applet's code is downloaded to the client and executed by the browser.	Adds dynamism and higher interactivity to Web pages for facilitating online purchasing.
Java Wallet	A client-side architecture designed to bring together pluggable commerce components to enable complex and secure online transactions in a platform-independent manner.	Offers a framework for performing purchasing, banking, and finance-related transactions.
Enterprise JavaBeans (EJBs)	Provides access to a core set of system services for developing enterprise-level multitier application systems for high-volume business transactions	Encapsulates transactional and business logic for e-commerce transactions.
Java Database Connectivity (JDBC)	Provides connectivity to existing relational databases.	Enables database-independent access to data sources.
Remote Method Invocation (RMI)	Enables the creation of distributed Java-to-Java applications in which the methods of remote Java objects can be invoked from other Java Virtual Machines, possibly on different hosts.	Enables invocation of remote e-commerce related services across different tiers of the architecture.
Java Servlets and Java Server Pages (JSPs)	Allows server-side Java programs to run with any major Web server and thus supports the middleware layers of the enterprise.	Allows access to server-side operations and services without compromising security constraints. A more efficient alternative to CGI-based services.
Java Security	A base infrastructure that ensures the privacy and consistency of data across the tiers of a Java-based architecture.	Ensures security in e-commerce transactions.

Table 1: Technologies that can be applied to e-commerce

Object International

www.oi.com

integration of various technologies that play specific roles in a distributed computing environment. A distributed topology is a prerequisite for building such applications since the Internet is inherently distributed in nature. A distributed architecture, especially in the realm of electronic commerce, places higher demands on connectivity, security, reliability, robustness and scalability than do applications in other areas of computing.

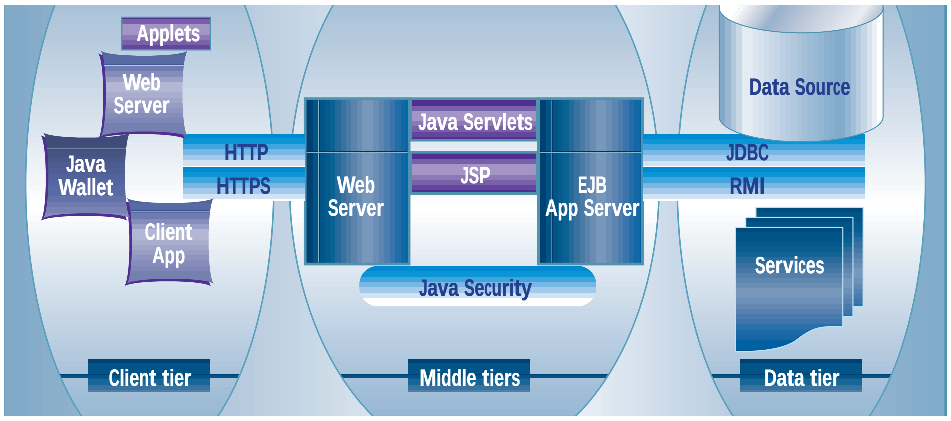


Figure 1: The role of Java technologies in a distributed n-tier architecture

Java and E-Commerce

So what role can Java play in the world of electronic commerce? Before answering that question, let's refresh our minds with the original goals of the Java Platform. The authors of Java used the following buzzwords to define the features offered by Java – *simple, object-oriented, distributed, robust, secure, architecture-neutral, portable, interpreted, high performance, multithreaded, dynamic*. To date Java has proved most of these descriptors accurate. Also, Sun Microsystems and other industry partners that influenced the evolution of the Java Platform are working hard to make sure that Java meets the expectations of the industry regarding features that are not yet satisfactory (e.g., performance).

Of the 11 buzzwords, the ones that have a direct bearing on the world of Internet commerce are:

- ***Distributed:*** As mentioned earlier, Internet commerce implies a distributed nature. Transactions on the Internet may take place across several tiers of a distributed architecture. The Java Platform and its APIs provide a high degree of support for distributed architectures.
- ***Robust:*** Buying and selling goods or services on the Internet requires robust services. Lack of trust in transactions can make the difference between people accepting e-commerce channels over traditional commerce channels.
- ***Secure:*** Security is perhaps the first thing that comes to mind when parties conduct transactions over the Web since the data transferred over the Internet is over public channels. Java provides an extremely secure base infrastructure that ensures security in Internet transactions.
- ***Architecture-neutral:*** The fact that software developed in Java is architecture-neutral is the most important reason for its being particularly well-suited to networked environments, since networks (especially the Internet) typically interconnect heterogeneous platforms that must work together despite their underlying differences.

Java also provides a rich set of APIs that facilitate building networked applications, particularly for the World Wide Web. This is why Java has become the enabling technology for building interoperable, distributed Web applications. Commerce, by definition, involves relationships between multiple participants, as opposed to transactions between individuals. As such, when business is conducted over the Web, it's to the advantage of all parties involved to overcome underlying differences in their respective computing platforms, to concentrate on buying and selling, not on whether they're using compatible operating systems. For this reason, Java is well-suited to being an enabling context for Web-based commerce, paving the way for business on the Internet. At the same time, for the end user, Java-based commerce-enabled Web pages can bring a greater degree of interactivity and responsiveness to a buyer's shopping experience.

Java Technologies for E-Commerce

The Java Platform offers several technologies that can be applied in the world of electronic commerce. Some are listed in Table 1. Figure 1 illustrates how each of these technologies fits into an *n*-tier e-commerce application. Note that the different technologies occupy specific roles in each tier of the architecture.

Other Technologies

The Java Platform offers many components that will contribute to the successful conduct of business on the Internet. However, it won't provide all the pieces necessary to make e-commerce a success. Other technologies are competing with the solutions offered by Java, trying to ensure their place in the world of e-commerce. At the same time, Java needs to integrate with supplementary technologies to provide viable solutions in the e-commerce market. Providing the technological answers solves only part of the problem. Integrating with existing and legacy technologies is imperative for providing feasible solutions for the market. Some existing technologies include payment and ordering systems, EDI, modules implemented in legacy programming languages and so on. Prominent among emerging technologies are new security paradigms, non-Java commerce frameworks and XML (Extended Markup Language). We'll discuss these technologies and considerations in future columns.

About the Author

Ajit Sagar is a member of the technical staff at i2 Technologies in Dallas, Texas. He holds a BS in electrical engineering from BITS Pilani, India, and an MS in computer science from Mississippi State University. He is a Java certified programmer with eight years of programming experience, including two in Java.

Ajit_Sagar@i2.com

Titestone

www.tidestone.com

Forget Stress – Life is a Gas...

Unless you let things get out of hand

by **Alan Williamson**

Is it me, or are the months flying past? It seems like only last week I was sitting down writing this column, hoping to bring a little happiness into your lives. This month, fortunately, I have a lot to tell you about the wonderful world of Java as seen through the eyes of a European CEO. But let's not get into the heavy stuff just yet. This column is like getting to work in the mornings: you don't just rush straight into it; you go and make a cup of coffee, you relax, you allow your heart rate to return to normal after the road-war you had to go through to get here. Hoping you don't meet that wee old lady who insisted on driving 25 mph in front of you the whole way to work. Hell, you deserve a rest. So if this column is coming to you on a Monday morning or some other stressful morning, then let's ease into it together, gently.

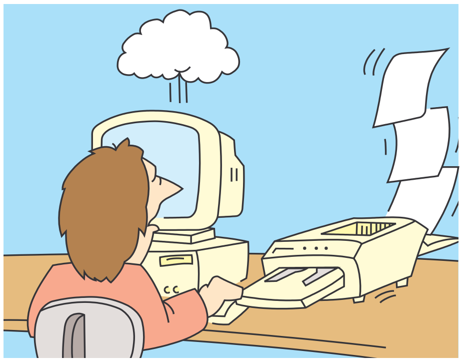
Stress is a funny thing. It affects us all in different ways. I'm one of the lucky ones that don't suffer from stress. There's never much point in losing sleep over a problem; it'll still be there in the morning so you may as well get a good night's rest! May sound a bit oversimplified, but let's face it: it's when you're panicky or stressful that you make those absolutely monumental blunders. Your decision-making process is a little cloudy and you can't see the forest for the trees. Life is a gas, and we should all learn not to take things so seriously. They're not that complicated. We merely shroud them in buzzwords and technical phrases to ensure that not everyone knows what we're talking about. It's how we protect our jobs. The IT industry isn't the only one that plays this game; everyone is at it. But should it be like this?

I guess not. But now that we're settled, let's start work....Well, let *me* start work. I have to write this column; you merely have to read it.

This was one of those months that could have proved extremely stressful if we had let things get out of hand. Have you ever built a system that you completely underestimated the success of? You never thought that particular class or method would be used quite as much as that. Well, this was only one of the things we had to contend with this month.

Popularity

As you may remember, some months ago we launched the system we've been working on to provide a free Web-based e-mail and newsgroup system, www.LiquidInformation.net. The growth of the system was something that took us all aback. We were signing up 300 new users a day, with around 40% daily usage from all of them. The system was being well stress tested. As you know, it was running on one Sun Solaris station with Oracle8 as the back-end database.



For up to around 3,000 users the system ran very well. Response from the Web site was fast. Sending and receiving e-mail was reliable and secure. When we topped 4,000 users, however, a different story evolved. Last month, remember, I ranted about the joys the JAR file brought to us. Well, this month I have an equally joyous rant to bring to you.

We couldn't understand why suddenly everything started to grind to a halt. With around 4,000 users we were holding around 30,000 e-mails. It was one of those situations where in attempting to locate the crux of the problem we discovered so many little anomalies on the way. For example, our mail server was a Java application that listened for requests on port 25 and processed incoming mail, rejecting unknown users, etc. But the log files started throwing up a rather strange error, java.lang.OutOfMemory. We spent ages trying to get to the root of it.

At first we upped the memory the virtual machine was starting out with, but that fixed the symptom, not the problem. Sure enough, it came back...suspiciously quick. So our mail server was crashing within 10 minutes of operation with memory problems. What could it be? The log files pointed toward the readLine(...) method from the BufferedReader class. Maybe there's a mail server hell-bent on sending us a line of a mail message that isn't terminated around the 80-character mark. Maybe that's overflowing the memory for the String class. At this point it was the best lead we had, but the problem still persisted after we wrote our own readLine(...) method to return a line no greater than 80 characters. We noted, however, a number of mail servers that never respected the 80-character rule of thumb, so this did indeed need addressing.

But the memory problem was still there. What could be taking away all the memory in such a short space of time? A spammer. We were being flooded with return e-mails from AOL where someone had put the return address of one of the users in Liquid. Our system permits only 10 originating e-mails per user per day, so we knew he wasn't using Liquid to spam thousands of users, but he was using our system as his return e-mail.

What happened was we forgot to limit the number of concurrent sessions our e-mail server could handle. AOL got a little excited and opened up in excess of 100 sessions with our mail server in an attempt to download all these return e-mails. That's what it was. The virtual machine couldn't handle the sudden stress of having to spawn over 100 threads and do all the necessary string handling. As soon as we built a throttling mechanism into the system to handle at most only 20 or 30 sessions at once, the problem never raised its ugly head again.

Phew! Thank goodness that was sorted out. But it never really solved the problem on the Web site. A major slowdown was still being experienced.

Looking at the load on the processor, we could see that Oracle was soaking up the majority of the CPU cycles. But why? The total size of the database was only around 80 MB, and we understood Oracle could handle far greater sizes than this. Granted, one of our tables had over a million rows in it but still we thought Oracle could handle it.

Developmentor

www.develop.com

We have our own in-house connection pool manager that does a whole host of housekeeping routines and ensures a safe and reliable database operation. This manager was reporting that each query was taking two to three seconds to complete. No wonder the Web site was grinding to a halt! So what was Oracle doing with its time?

First of all, we rationalized the table that had over a million rows and this indeed improved the situation. But things were still slow. We looked closely at all the settings Oracle had and nothing too untoward was popping up. Next we looked at our table design.

Guess what it was? We had forgotten to allocate a primary key to one of the main tables. Duh! Talk about kicking yourself in the butt and rolling your eyes. When we rebuilt the table with a primary key, what a difference it made! Speed was regained and operation was smooth.

This was a classic case of bugs or smaller issues not being a problem in a test or low-volume environment, but when deployed in a high-transaction environment it simply failed. We had taken the time to test all our classes and ensure all exceptions and errors were reported, etc., but neglected to look closely at the other parts of the system.

Farming

As the user base grew, the need for a second machine to support the overall system soon became apparent. We toyed with a number of different configurations. We looked at how the larger sites operate – Amazon, Dell and the Internet Movie databases. These are all high-volume sites and lots could be learned from them. Ironically, each of them employed completely different ways of handling large volumes of users, with no one solution shining through as the perfect way to handle it.

The first thing that had to be replaced was the Web server. The one we had employed was simply not up to the task. We had the Java Web Server running, handling all our servlet and static file requests. It was using more and more time to process requests. We also noticed it had a tendency to simply hang or freeze after a prolonged period of time. We put this down to more of the virtual machine as opposed to the Web server itself, but we couldn't confirm or deny this. The mail server was up equally as much and it didn't freeze or hang. A new Web server was required, one that supported servlets.

We looked at Netscape and seriously thought of deploying this solution. But two things put us off: the handling of servlets, which would have to be performed by a third-party vendor, and the cost. Which leads me to a wee aside.

Don't you think the Internet has spoiled us somewhat? With the quality of free software that's available, when a product that does charge comes along it really has to justify its existence. A number of months ago I talked about the real costs of owning a Sun station as opposed to a Linux box, and the feedback I got from you all on this suggested we weren't the only company to realize this. Which makes sense if you think about it. If it didn't, then Linux wouldn't be gathering the popularity it is at the moment. It makes it harder for companies to make money on the Internet if they're expected to give a certain amount of their crown jewels away in order to gain popularity. It's a worryingly unclear future for product companies, and one we here at n-ary are addressing sooner rather than later. I think we'll come back to this next month.

"The first thing that had to be replaced was the Web server. The one we had employed was simply not up to the task."

If I asked you to name the most popular free Web server, I'd bet $1,000 that 99% of you would shout back "Apache!" And it's Apache we turned to. We downloaded it, installed and configured the servlet module, and were up and serving requests within two hours. A truly remarkable piece of software, and for this I take my hat off to the Apache team and publicly congratulate them on a job well done.

Apache did as fine a job of serving the same level of requests as the Java Web Server, but didn't take up as many processing cycles. Which concerns me a little. We in the Java world are trying to promote the use of Java and get it into the high-transaction environments, and here we are, a leading Java servlet company, dropping a Java product in favor of a more robust, platform-independent solution. We ask forgiveness from the Java community for this act of Judas, but we had a job to do, and Apache was the man for the job. Nothing would have given us more pleasure than to proudly announce a complete Java solution, but sadly it wasn't to be. Our database is Oracle and our Web server is Apache. but all our code is Java!

Next month I'll take you into the wonderful world of databases for Linux as we build up our server farm. And I promise you, I can save you money in this field too. Our eyes were well and truly opened, and I wish to share this wondrous sight with you all. But more next month.

N-ary Poll

Last month we asked you on our main Web site (www.n-ary.com) if you felt Java applets were still slow to execute. The poll ended with a resounding 77% of you indicating that applets were still slow. This is a very interesting result and we can see that virtual machine developers still have a long way to go to improve the reputation of Java's speed. We'll watch this one carefully and possibly redo the poll in six months to see if the situation has improved.

This month we're asking if you've looked at deploying a Java servlet solution as opposed to a Microsoft ASP or CGI solution. The results are encouraging, with 75% of the votes showing a swing toward servlets. It's good to see Java servlets becoming a viable solution to you all. Next month I'll report the final findings.

I've run out of space for the book review. So please accept my apologies and I'll ensure it gets in next month. But I'd like to thank all of you that have signed up on the mailing list that accompanies this column. It's proving to be a much better way to discuss issues that are raised here, so if you're up for some stimulating conversation, then come along. See our main Web site for details.

On that note, I'll look forward to hearing from you, and prepare myself for the hazardous drive home. Fortunately for me, as we're situated in the lowlands of Scotland, rush hour consists of making sure you get past the farm before the farmer drives the cows along the single-track road for milking.

And you think you have problems!

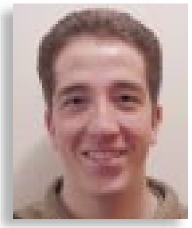
About the Author

Alan Williamson is CEO of n-ary 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) and welcomes all suggestions and comments.

alan@n-ary.com

Riverton

www.riverton.com



Meet the Swing JTable

It's a complex control, not well documented, but here's some how-to information

by **Michael Hatmaker**

Now that Swing (a k a JFC) has been officially released, an abundance of electronic and print material is available to serve as both tutorial and reference. The problem with the books and articles I've seen thus far is that they devote the same amount of space to each Swing component. For example, simple controls such as JLabel and JButton have as much (relatively) written about them as is written about more complex controls like JTree and JTable. This article attempts to rectify some of the imbalance by providing an in-depth look at the Swing JTable.

JTable uses the same Model-View-Controller–type scenario that is evident throughout Swing. If you're unfamiliar with the MVC paradigm, you can get a good introduction in several places (such as the Swing Connection Web site, for instance), but the gist of MVC is that any changes you want to make to the data that's displayed are made in a *model* while the actual presentation of the data is left to a *view* that receives the data it's to display from the underlying model. Table 1 summarizes the differences between using a non-MVC table and a Swing JTable.

With a non-MVC component, we actually tell the component itself to change the contents of one of its cells. With the Swing JTable, we update the underlying model for the table (which has been previously linked to this JTable) and then inform the JTable that its model has been updated.

JTable Classes and Interfaces

All of the JTable-specific functionality can be found in either the JTable class (which is within the javax.swing package) or one of the classes in the javax.swing.table package. Most of your programs should import both of these classes as follows:

```
import javax.swing.*;
import javax.swing.table.*;
```

Non-MVC Table Bean	Swing JTable
• Insert the new contents directly into the table at a specific row and column location	• Change the contents of the underlying table model (for a specific row and column) • Inform the JTable that the contents of its model have changed so it can update itself

Table 1: Non-MVC table vs Swing JTable

Note: The package-naming scheme within Swing changed for JFC version 1.1. Prior to this, the Swing packages started with com.sun.java.swing. For JFC versions 1.1 and after, the package names start with javax.swing.

When implementing a JTable in Swing, the TableModel, TableColumnModel, TableCellEditor and TableCellRenderer interfaces will be the most important. Swing also provides some default classes – AbstractTableModel, DefaultTableModel and DefaultColumnModel – that already implement these interfaces.

The main classes we'll use are the JTable, JTableHeader and TableColumn. Figure 1 shows how these classes and interfaces relate to one another.

Default Table Model

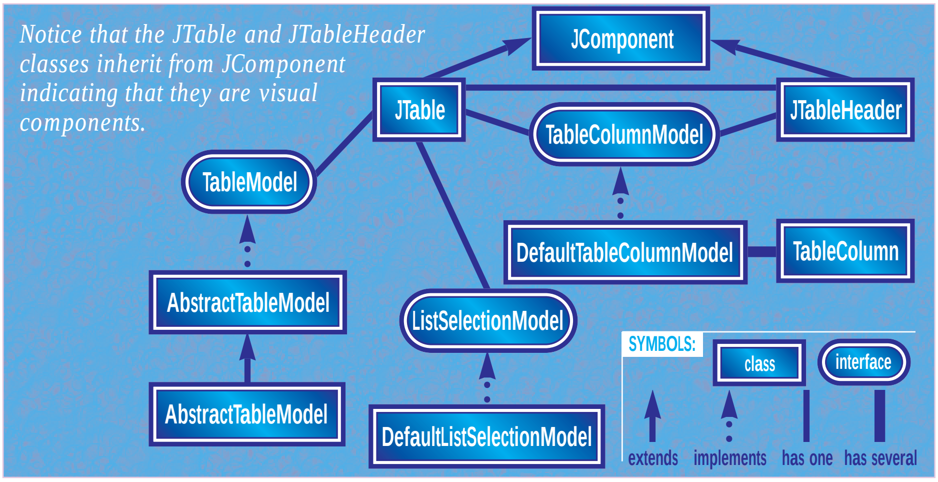
Often, the best way to present a detailed concept such as a JTable is to start with an example (see Listing 1). Figure 2 shows a basic example of a JTable.

In this example we're using the DefaultTableModel class (located in the javax.swing.table package) to implement our table model. The DefaultTableModel class extends the AbstractTableModel class, which implements the TableModel interface (as mentioned previously). This saves us work because we don't have to extend AbstractTableModel ourselves, but this ease of use comes at the expense of functionality, as we'll see in subsequent articles when we implement our own table models.

There are several constructors for the DefaultTableModel class, but I've chosen to use the constructor that requires both a two-dimensional array (for the actual data that should be displayed in the table) and a one-dimensional array for the column headers. Once we create the table model, we simply pass this table model to the constructor for our JTable and add the JTable to the frame.

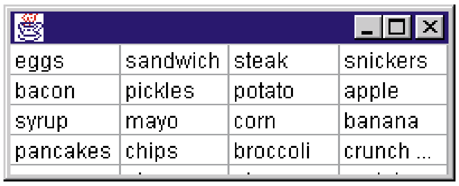
If you compile and run the code in Listing 1, you should see a JFrame similar to that in Figure 2. Here are a few things to notice, however, about our simple table example:

- No column headers are displayed.
- The table information is clipped at the frame border and there are no scrollbars showing, so we have to resize the frame in order to see all the data.
- All cells are editable – try typing over some of the existing data.



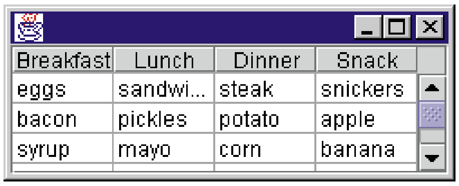
Notice that the JTable and JTableHeader classes inherit from JComponent indicating that they are visual components.

Figure 1: JTable classes and interfaces



eggs	sandwich	steak	snickers
bacon	pickles	potato	apple
syrup	mayo	corn	banana
pancakes	chips	broccoli	crunch ...

Figure 2: Simple JTable



Breakfast	Lunch	Dinner	Snack
eggs	sandwi...	steak	snickers
bacon	pickles	potato	apple
syrup	mayo	corn	banana

Figure 3: JTable with column headers and scrollbar

Insignia

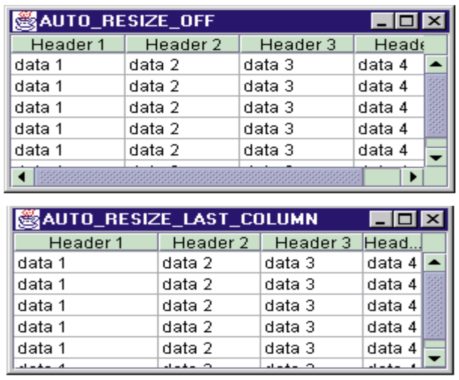
www.insignia.com

JTable Constant	Description
AUTO_RESIZE_ALL_COLUMNS	The columns are resized proportionately when the table is resized.
AUTO_RESIZE_LAST_COLUMN	When a column is resized, only the width of the last column (rightmost column) is adjusted to accommodate the new column width.
AUTO_RESIZE_NEXT_COLUMN	When a column is resized, only the width of the next column (the column to the right of the one that is resized) is adjusted.
AUTO_RESIZE_OFF	The columns are not resized when the table is resized. Instead, a horizontal scrollbar may appear (depending on the setting of the horizontalScrollBarPolicy property of the JScrollPane that contains our table).
AUTO_RESIZE_SUBSEQUENT_COLUMNS	All subsequent columns are adjusted proportionally when a column is resized.

Table 2: autoResizeMode property constants

ScrollPaneConstants constant	Description
HORIZONTAL_SCROLLBAR_ALWAYS	Always show a horizontal scrollbar, even if it isn't needed (even if the component fits fully within the visible region of the scroll pane).
HORIZONTAL_SCROLLBAR_AS_NEEDED	Show a horizontal scrollbar only when the width of the component contained in the JScrollPane is greater than that of the visible region.
HORIZONTAL_SCROLLBAR_NEVER	Never show a horizontal scrollbar.
VERTICAL_SCROLLBAR_ALWAYS	Always show a vertical scrollbar, even if it isn't needed (even if the component fits fully within the visible region of the scroll pane).
VERTICAL_SCROLLBAR_AS_NEEDED	Show a vertical scrollbar only when the height of the component contained in the JScrollPane is greater than that of the visible region.
VERTICAL_SCROLLBAR_NEVER	Never show a vertical scrollbar.

Table 3: Constants for the scrollbar policy properties of the JScrollPane



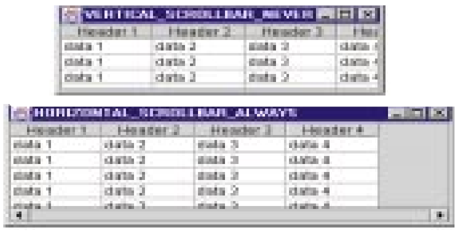
AUTO_RESIZE_OFF

Header 1	Header 2	Header 3	Heade
data 1	data 2	data 3	data 4
data 1	data 2	data 3	data 4
data 1	data 2	data 3	data 4
data 1	data 2	data 3	data 4
data 1	data 2	data 3	data 4

AUTO_RESIZE_LAST_COLUMN

Header 1	Header 2	Header 3	Head...
data 1	data 2	data 3	data 4
data 1	data 2	data 3	data 4
data 1	data 2	data 3	data 4
data 1	data 2	data 3	data 4
data 1	data 2	data 3	data 4

Figure 4: Two of the five frames showing different settings for the autoResizeMode property of the JTable



VERTICAL_SCROLLBAR_NEVER

Header 1	Header 2	Header 3	Hea
data 1	data 2	data 3	data 4
data 1	data 2	data 3	data 4
data 1	data 2	data 3	data 4

HORIZONTAL_SCROLLBAR_ALWAYS

Header 1	Header 2	Header 3	Header 4
data 1	data 2	data 3	data 4
data 1	data 2	data 3	data 4
data 1	data 2	data 3	data 4
data 1	data 2	data 3	data 4

Figure 5: Two of the six frames showing different settings for the scrollbar policy properties of the JScrollPane

The first two observations we can correct by using another Swing control – a JScrollPane. The third we'll save for a subsequent article when we look at creating our own table model.

JScrollPane

Almost no table would be complete without the ability to scroll through the data items in the table. Fortunately, Swing makes it easy for us to add this functionality; we simply insert our JTable into a JScrollPane. With a slight modification to our original code, we create a constructor for a JScrollPane that takes the Component that should be displayed within the scroll pane as its argument. Listing 2 contains the modified code, and Figure 3 shows the resulting JTable.

Note: There is a bug in version 1.0.1 of the Swing classes that won't show the column headers when you place a JTable inside a JScrollPane. This problem should be fixed in all versions of Swing 1.0.2 and above.

Placing the JTable inside a JScrollPane has automatically displayed our column headers and has added a vertical scrollbar. There is no horizontal scrollbar, however. Try resizing the frame; you'll see that the horizontal scrollbar won't appear regardless of how narrow you make the frame. The reason is that, by default, the autoResizeMode property of the JTable is set to AUTO_RESIZE_ALL_COLUMNS. Table 2 lists the valid settings for the autoResizeMode property along with a description of each.

When you set the autoResizeMode property to AUTO_RESIZE_OFF, the JScrollPane containing your table may display a horizontal scrollbar rather than resizing the columns to fit the existing column widths. I say *may* because whether or not a scrollbar is actually displayed depends on the setting of the horizontalScrollBarPolicy property of the JScrollPane. If it is set to HORIZONTAL_SCROLLBAR_NEVER, for instance, no horizontal scrollbar will be displayed regardless of the size of the JTable the scroll pane contains. There is also a corresponding verticalScrollBarPolicy property for a JScrollPane. Table 3 contains a list of some of the relevant constants that can be used for the horizontalScrollBarPolicy and verticalScrollBarPolicy properties of the JScrollPane.

Listing 3 contains a sample program that makes use of the autoResizeMode property of the JTable. It creates five different JFrames, each containing a JTable that has a different setting for its autoResizeMode property. Figure 4 shows the resulting frames. The code in Listing 4 takes the case where the autoResizeMode is set to AUTO_RESIZE_OFF and displays six frames, each with a different value for the scrollbar policy properties of the JScrollPane that contains the JTable. This code produces the output shown in Figure 5. Try adjusting the table widths and the column widths in each of these examples to see how the autoResizeMode, horizontalScrollBarPolicy and verticalScrollBarPolicy properties function.

In Listings 3 and 4 we use a different method to create the JTable than we have previously. Instead of first creating an instance of the DefaultTableModel class based on an array of data and an array of column headers, we simply pass the data and column headers directly to the JTable constructor. This essentially does the same thing, except it bypasses entirely the creation of a table model.

In the next part of this series we'll look at implementing the TableModel and TableColumnModel interfaces in order to achieve maximum flexibility in our JTable designs. Subsequent articles will reveal the techniques behind implementing editors and renderers for custom display and editing of your JTable data, and opportunities for enhancing the performance of the sometimes-laggard JTable component.

▼▼ FULL CODE LISTING BELOW ▼▼

About the Author

Michael Hatmaker has worked for several years as a consultant and as an instructor at Loyola University. He is currently specializing in Java and CORBA development and is working on a large-scale trading system project at a major financial exchange.

mhatmak@smartsystems.com

Listing 1: Code for Simple JTable

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

public class SimpleTableTest extends JFrame {

  public SimpleTableTest() {
    setLocation(100,100);
    setSize(250,100);

   String[][] data = { {"eggs", "sandwich", "steak", "snickers"},
                       {"bacon", "pickles", "potato", "apple"},
                       {"syrup", "mayo", "corn", "banana"},
                       {"pancakes", "chips", "broccoli", "crunch bar"},
```

Inetsoft

www.inetsoft.com

```
                 {"sausage", "pizza", "pie", "protein shake"}};
    String[] headers = {"Breakfast", "Lunch", "Dinner", "Snack"};
    DefaultTableModel model = new DefaultTableModel(data, headers);
    JTable table = new JTable(model);

    getContentPane().add(table);

    setVisible(true);
  }

  public static void main(String[] args) {
    SimpleTableTest simpleTableTest = new SimpleTableTest();
  }
}
```

Listing 2: JTable Inside JScrollPane

```
import javax.swing.*;
import javax.swing.table.*;

public class SimpleTableTest extends JFrame {

  public SimpleTableTest() {
    setLocation(100,100);
    setSize(250,100);

    String[][] data = {  {"eggs", "sandwich", "steak", "snickers"},
                 {"bacon", "pickles", "potato", "apple"},
                 {"syrup", "mayo", "corn", "banana"},
                 {"pancakes", "chips", "broccoli", "crunch bar"},
                 {"sausage", "pizza", "pie", "protein shake"}};
    String[] headers = {"Breakfast", "Lunch",
"Dinner", "Snack"};
    DefaultTableModel model = new DefaultTableModel(data, headers);
    JTable table = new JTable(model);
    JScrollPane scroll = new JScrollPane(table);

    getContentPane().add(scroll);

    setVisible(true);
  }

  public static void main(String[] args) {
    SimpleTableTest simpleTableTest = new SimpleTableTest();
  }
}
```

Listing 3: Code to test the autoResizeMode property of the JTable import javax.swing.*;

```
class TableColumnSize extends JFrame {
   private static int offset = 50;

   public TableColumnSize(int resizeMode, String title) {
      // Dummy data for table.
      String[][] tableData = {
         {"data 1", "data 2", "data 3", "data 4"},
         {"data 1", "data 2", "data 3", "data 4"},
         {"data 1", "data 2", "data 3", "data 4"},
         {"data 1", "data 2", "data 3", "data 4"},
         {"data 1", "data 2", "data 3", "data 4"},
         {"data 1", "data 2", "data 3", "data 4"}};
      String[] headerData = {"Header 1", "Header 2", "Header 3", "Header 4"};

      // Set the frame's title and position.
      setTitle(title);
      offset += 50;
      setLocation(offset,offset);
      setSize(300,150);

      // Create the JTable using the dummy data.
      JTable table = new JTable(tableData, headerData);

      // This is the important part of this example: Set the autoResizeMode
      // of the JTable.
      table.setAutoResizeMode(resizeMode);

      // Create a scroll pane and insert the JTable into the scroll pane.
      JScrollPane scroll = new JScrollPane(table);

      getContentPane().add(scroll);
   }

   public static void main(String[] args) {
      // Create 5 TableColumnSize frames - each demonstrating a
      // different value of the autoResizeMode property.
      (new
TableColumnSize(JTable.AUTO_RESIZE_OFF,"AUTO_RESIZE_OFF")).setVisible(true)
;
      (new
TableColumnSize(JTable.AUTO_RESIZE_ALL_COLUMNS,"AUTO_RESIZE_ALL_CO-
LUMNS")).setVisible(true);
      (new
TableColumnSize(JTable.AUTO_RESIZE_LAST_COLUMN,"AUTO_RESIZE_LAST_CO-
LUMN")).setVisible(true);
      (new
TableColumnSize(JTable.AUTO_RESIZE_NEXT_COLUMN,"AUTO_RESIZE_NEXT_CO-
LUMN")).setVisible(true);
      (new
TableColumnSize(JTable.AUTO_RESIZE_SUBSEQUENT_COLUMNS,"AUTO_RESIZE_S
UBSEQUENT_COLUMNS")).setVisible(true);
   }
}
```

Listing 4: Code to test the horizontalScrollBarPolicy and vertical Scroll BarPolicy of the JScrollPane

```
import javax.swing.*;

class ScrollBarTest extends JFrame {
   private static int offset = 50;

   public ScrollBarTest(int horizPolicy, int vertPolicy, String title) {
      // Dummy data for table.
      String[][] tableData = {
         {"data 1", "data 2", "data 3", "data 4"},
         {"data 1", "data 2", "data 3", "data 4"},
         {"data 1", "data 2", "data 3", "data 4"},
         {"data 1", "data 2", "data 3", "data 4"},
         {"data 1", "data 2", "data 3", "data 4"},
         {"data 1", "data 2", "data 3", "data 4"}};
      String[] headerData = {"Header 1", "Header 2", "Header 3", "Header 4"};

      // Set the frame's title and position.
      setTitle(title);
      offset += 50;
      setLocation(offset,offset);
      setSize(300,150);

      // Create the JTable using the dummy data.
      JTable table = new JTable(tableData, headerData);

      table.setAutoResizeMode(JTable.AUTO_RESIZE_OFF);

      // Create a scroll pane and insert the JTable into the scroll pane.
      JScrollPane scroll = new JScrollPane(table);

      // This is the important part of this example: Set the ScrollBar Policies
      // of the JTable.
      scroll.setHorizontalScrollBarPolicy(horizPolicy);
      scroll.setVerticalScrollBarPolicy(vertPolicy);

      getContentPane().add(scroll);
   }

   public static void main(String[] args) {
      // Create 6 ScrollBarTest frames - each demonstrating a different
      // value of the horizontalScrollBarPolicy and verticalScrollBarPolicy
      // properties.
      (new ScrollBarTest(JScrollPane.HORIZONTAL_SCROLLBAR_ALWAYS,JScroll-
Pane.-
VERTICAL_SCROLLBAR_NEVER,"HORIZONTAL_SCROLLBAR_ALWAYS")).setVisi-
ble(true);
      (new
ScrollBarTest(JScrollPane.HORIZONTAL_SCROLLBAR_AS_NEEDED,JScrollPane.VER-
TICAL_SCROLLBAR_NEVER,"HORIZONTAL_SCROLLBAR_AS_NEEDED")).-
setVisible(true);
      (new ScrollBarTest(JScrollPane.HORIZONTAL_SCROLLBAR_NEVER,JScroll-
Pane.-
VERTICAL_SCROLLBAR_NEVER,"HORIZONTAL_SCROLLBAR_NEVER")).setVisible(tr
ue);
      (new ScrollBarTest(JScrollPane.HORIZONTAL_SCROLLBAR_NEVER,JScroll-
Pane.-
VERTICAL_SCROLLBAR_ALWAYS,"VERTICAL_SCROLLBAR_ALWAYS")).setVisible(t
rue);
      (new ScrollBarTest(JScrollPane.HORIZONTAL_SCROLLBAR_NEVER,JScroll-
Pane.-
VERTICAL_SCROLLBAR_AS_NEEDED,"VERTICAL_SCROLLBAR_AS_NEEDED")).setVi
sible(true);
      (new ScrollBarTest(JScrollPane.HORIZONTAL_SCROLLBAR_NEVER,JScroll-
Pane.VERTICAL_SCROLLBAR_NEVER,"VERTICAL_SCROLLBAR_NEVER")).setVisible-
(true);
   }
}
```

▼▼▼ DOWNLOAD THE CODE ▼▼▼

This code listing can be downloaded at **www.JavaDevelopersJournal.com**

JBuilder 3

www.borland.com/jbuilder

Microsoft

www.microsof

Spread

ft.com/visualc

Digital Neural Network Control Using JNI

Preserve your legacy software investment or delay the conversion of your hardware control libraries while taking advantage of Java machine-independent GUIs

by **Bernie Arruza**

Most Java application developers would like to create pure Java implementations that provide solutions for their users. In practice, however, the fact that most hardware products are shipped with legacy software negates that lofty purity goal. With Java you can design your cross-platform GUI while taking advantage of JNI to create a wrapper layer around the code that can't be ported. For example, a requirement can be to create a Java application that provides support for a PCI Artificial Neural Network (ANN) card under OS/2 and Windows NT. The set of APIs that communicates with the Neural Network is written in C and wrapped in a JNI dynamic-link library (DLL). To control the Neural Network the Java application has to invoke the correct set of JNI ANN APIs.

Architecture of a Hardware Neural Network

A PCI ZISC card contains one onboard zero instruction set computer (ZISC) and as many as three single inline ZISC modules (SIZMs). Each SIZM has six ZISCs. You can also have a setup made up of several PCI ZISC slots using a mother-daughter arrangement. In addition, ISA cards are available. The ZISC, a digital ANN containing 36 neurons (see Figure 1), implements the restricted coulomb energy (RCE) and the K-nearest neighbor (KNN) algorithms. The neurons on each of the ZISC chips are arranged following a radial basis function (RBF)-like network topology (see Figure 2) composed of three layers. Each input node corresponds to a component (Vi) of a feature vector. The outputs are categories or solutions to a classification problem, and the connections between the second and third layers are established dynamically as a result of the learning process.

The ZISC C library DLL contains a rich set of APIs to do initialization, recognition, classification, learning, data extraction, neuron selection and saving/restoring ZISC registers.

Java vs C Data Types

Some C/C++ data types are not compatible with data types in Java. One of the first things to do when you get ready to create your Java wrapper class is to identify the native data types that will be matched on the Java side. Developers need to consider several factors, such as the performance of the resulting code and the desire to remain faithful to the structure of the native APIs being replaced (see Table 1 for the mapping used within the ZISC wrapper class).

Developer Beware!

Determine your mapping as best you can to fit the situations in which the variables will be needed. Ask yourself questions: Is the native code using the variable and passing it to a function by value or reference? Is any one of the types being used as an array, and if so will your mapping cause the array contents to be lost? Don't be afraid to spend time working on this aspect of the design – you have to understand the interfaces of the functions you're wrapping.

JNI: A Quick Overview

Java can run on multiple platforms because the Java Virtual Machine (JVM) can interact with the underlying operating system software. As a Java developer, you'll soon need to invoke some native code if you haven't already. However, to create Java applications/applets that can be written once and tested everywhere, you have to encapsulate the native code interfaces and document them in such a way that anyone can understand where the impure code is located. Don't forget: this native code isn't portable.

JNI acts as the glue between your Java application and the supporting native code. It lets you invoke native functions, pass parameters to them and capture the return codes. To create a JNI application:

1. Build a Java wrapper class that includes all the native functions you'll be calling from Java.
2. Use the javah tool included in the Java JDK to create a C/C++ language header file that contains the Java to C/C++ prototypes.

   Use javah -help for more information on how it can be used. To build the C/C++ header file for Java wrapper class XXXX you'd execute javah -jni XXXX. jni_md.h contains all JNI-accepted calling sequences, such as JNIEXPORT and JNICALL.

Native methods receive at least two of these parameters:

- JNIEnv – points to the existing JNI environment (interface pointer) during

C/C++ (Intel)		Java	JNI
char *	(32 bits)	String	jstring
byte*	(32 bits)	byte[]	jbyteArray
WORD	(unsigned short-16bit)	int	jint
short	(signed short-16 bit)	short	jshort
int	(32 bits)	int	jint
void	(32 bits)	void	void
BOOL	(unsigned short-16bit)	boolean	jboolean
BYTE	(8 bits)	byte	jbyte

Table 1: Variable types mapping within the ZISC wrapper class

method invocation, as well as to a table that contains the function pointers for all JNI conversion methods. This pointer should be used only within the thread where it was loaded by the JVM.
- jobject – for the class instantiated object method or jclass for static methods.

See file jni.h for all supported JNI types and data access methods. Also, don't be alarmed by the name mangling caused by javah when creating the native language prototypes, since they are invisible to the user. Notice how for the ZISC code a "_" is replaced by a "_1".

3. Create a new C/C++ DLL project using your favorite development platform. This project should point to your generated header file. Notice the functions in jni.h are referenced differently depending on the type of source module used to build the native DLL. From a C source file we'd invoke a JNI conversion method as:

```
 (*env)->GetByteArrayElements(env, componentArray, NULL);
```

From a C++ file the same function is referenced as:

```
  env->GetByteArrayElements(componentArray, NULL);
```

Other conversion methods handle:
- Obtaining the handle of the Java object's class

```
 j_class = (*env)->GetObjectClass(env, jobject_parameter);
```

- Getting the value of an integer variable from Java. Use GetStaticFieldID and GetStaticIntField for static variables (see GetXXXXField for other types):

```
 j_fieldID = (*env)->GetFieldID(env, j_class, "int_variable_name", "I");
 int_value = (*env)->GetIntField(env, jobject_parameter, j_fieldID);
```

- Calling a Java method. Use GetStaticMethodID and CallStaticVoidMethod for static variables (see CallXXXXMethod for other types of methods).

```
 j_methodID = (*env)->GetMethodID(j_class, "void_method_name", "()V");
 (*env)->CallVoidMethod(j_class, j_methodID);
```

I seldom use these conversion methods as my designs specify that the hardware should know nothing about how the Java software is implemented. One of the few exceptions could be the JNI exception-handling conversion methods. The following sections describe how the ANN hardware provides services for the Java software making the requests. As a rule, avoid developing JNI code that "knows" the interfaces for the service provider layer and service requester layer. This results in an interface that will be hard to maintain and upgrade.

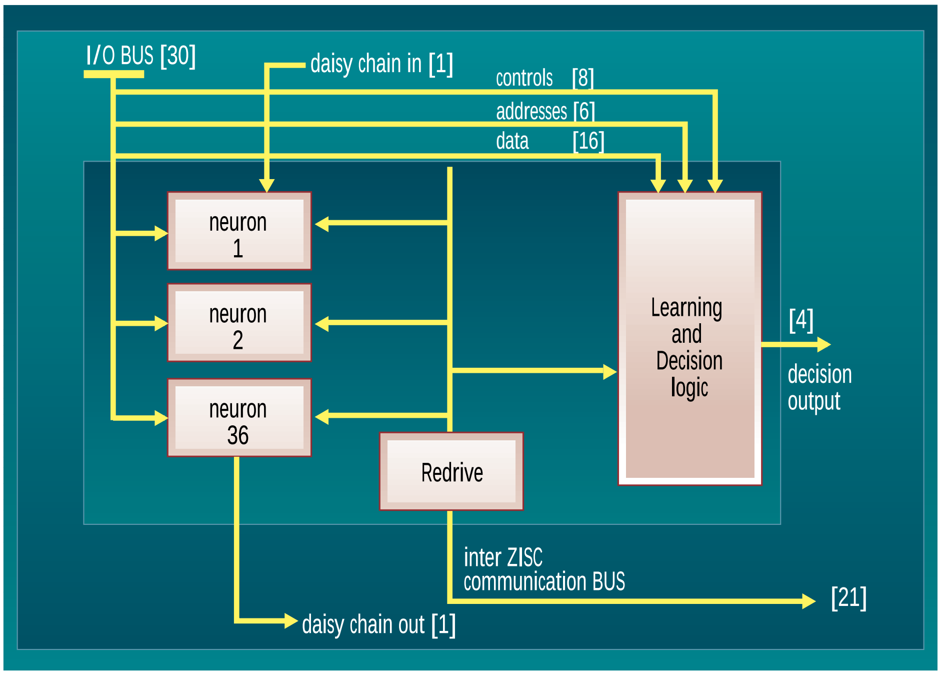


Figure 1: ZISC architecture block diagram

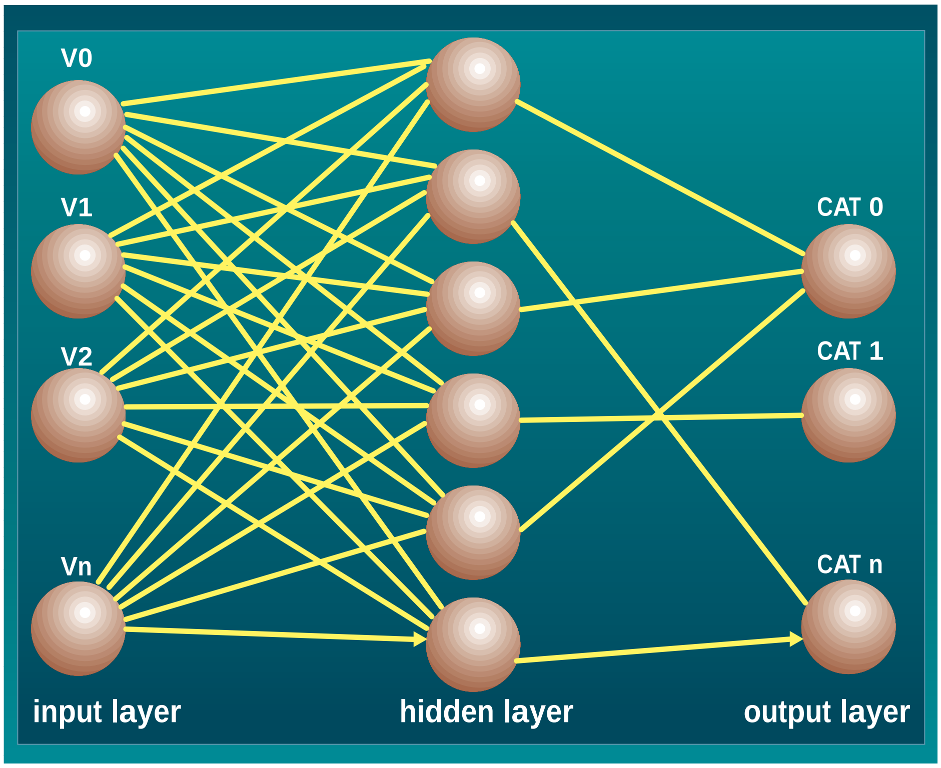


Figure 2: Three-layer RBF network topology

Neural Network Java Wrapper

The Java wrapper class JZISC was built using IBM VisualAge Java 2.0; the JNI library JZISC.DLL was built using Microsoft Visual C++ 5.0.

To create the JNI JZISC class, you can follow the steps listed in the previous section:

1. Build a Java wrapper class JZISC that includes all functions you will be calling from Java. An example of a native Java method looks like:

```
 public native void JZ_Identify(byte compArray[], int nComp, int status[], int category[]);
```

Notice the native keyword and the lack of a method body. Also, when the JZISC class is initialized it'll have to load the native DLL that contains the actual code.

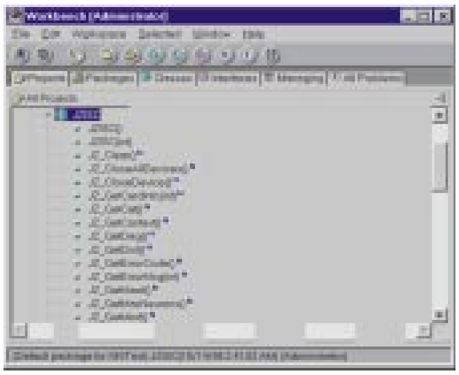


Figure 3: Partial view of JZISC class methods

```
  static {
    System.loadLibrary("JZISC");  // Make sure the
file's directory is in PATH statement
   }
```

2. Since the ANN Java wrapper class is called JZISC (see Figure 3), to build the C header file you would execute javah -jni JZISC. The corresponding generated prototype for the previously shown Java native method looks like:

```
  JNIEXPORT void JNICALL Java_JZISC_JZ_1Identi-
fy
       (JNIEnv *, jobject, jbyteArray, jint, jintArray,
jintArray);
```

3. Your source file within the C DLL project should include all required header files like:

```
  #include <jni.h>           /* JNI includes
*/
  #include "JZISC.h"          /* javah generated
header file     */
  #include "zapi.h"            /* ANN C prototypes
for ZISC card */
```

When creating your function definitions, follow the format of the prototypes in the header file. For our sample method the actual implementation would look like the contents of Listing 1.

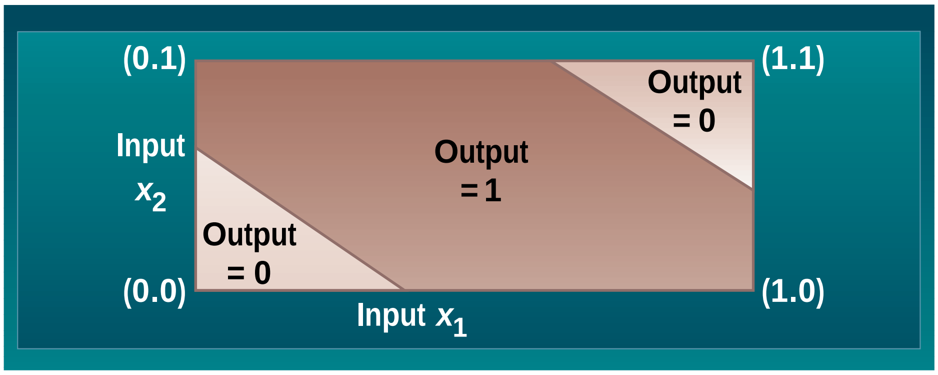


Figure 4: XOR problem decision boundary

A Solution to the XOR Problem

In 1969, Minsky and Pappert demonstrated the limitations of a single-layer neural network by proving that it couldn't do a simple pattern classification task. If a multilayer ANN can solve the exclusive OR (XOR) problem, f(0 0)=0, f(0 1)=1, f(1 0)=1, f(1 1)=0 (see Figure 4), then it can also classify more complicated input patterns that are not linearly separable.

The first thing the Java sample application does is to initialize the ANN PCI card, query it for the number of neurons available, the number of cards installed – ISA or PCI – and the version number (see Listing 2). Next we load the XOR value pairs and the predicted results (see Listing 3 for an example of a pair of inputs and expected output). Note that the ANN expects to see inputs of type byte, and zero isn't a good choice for an input value since it doesn't provide enough resolution. With ANN implementations the developer has to normalize the input values. For this implementation we equate zero (false) to a value of 1 and 1 (true) to a value of 5. Once the training process has been completed, the network is tested by specifying an input pair and asking the network to return the predicted value (see Listing 4 for a portion of the output generated by the Java program).

This card can perform other tasks, such as executing other types of recognition schemes, minimizing the number of neurons used and saving the ANN weights, but that's another article.

References and Resources

1. R. David, E. Williams, G. de Tremiolles and P.l Tannhof. "Noise Reduction and Image Enhancement Using a Hardware Implementation of Artificial Neural Networks." www.fr.ibm.com/france/cdlab/-contact.htm
2. IBM France ZISC home page: www.fr.ibm.com/france/cdlab/zisc.htm
3. Silicon Recognition: www.silirec.com/
4. S. Haykin (1994). *Neural Networks: A Comprehensive Foundation*, pp. 236–284. IEEE Press. ISBN 0-02-352761-7.

About the Author

Bernie Arruza is an advisory engineer at the IBM Manufacturing Technology Center in Boca Raton, Florida, with an MS in electrical engineering. His

▼▼ FULL CODE LISTING BELOW ▼▼

department designs and develops atomic force microscopes for the semiconductor industry. You can contact Bernie with questions and comments at arruza@ibm.net.

arruza@ibm.net

Listing 1: JNI prototype implementation

```
/* This is the actual C ZISC function prototype */
// void IMPORT Z_Identify(BYTE _far *componentArray, WORD _far NComp,
WORD _far *Status, WORD _far *Category);
/* This is a local wrapper for the actual C function. It can be used for debugging
purposes to
   verify data type conversions and returned values before they are passed to
the third party
   DLL */
void J_Z_Identify(BYTE *componentArray, WORD nComp, WORD *statusArray,
WORD *categoryArray) {
 Z_Identify(componentArray, nComp, statusArray, categoryArray);
}
/* Implementation of the javah generated prototype */
JNIEXPORT void JNICALL
Java_JZISC_JZ_1Identify(JNIEnv *env, jobject obj, jbyteArray componentArray, jint
nComp, jintArray statusArray, jintArray categoryArray) {
  // Get all component array elements
 jbyte *component_array =
  (*env)->GetByteArrayElements(env, componentArray, NULL);
 // Get components from status array
 jint *status_array =
  (*env)->GetIntArrayElements(env, statusArray, NULL);
 // Get components from category array
 jint *category_array =
  (*env)->GetIntArrayElements(env, categoryArray, NULL);
  // Call local wrapper for ZISC method
 J_Z_Identify((BYTE *)component_array, (WORD)nComp, (WORD *)status_array,
(WORD   *)category_array);
  // Release work buffer for component array
 (*env)->ReleaseByteArrayElements(env, componentArray, component_array, 0);
  // Update status array with new value
 (*env)->SetIntArrayRegion(env, statusArray, 0, 1, status_array);
  // Release work buffer for status array
 (*env)->ReleaseIntArrayElements(env, statusArray, status_array, 0);
  // Update category array with new value
 (*env)->SetIntArrayRegion(env, categoryArray, 0, 1, category_array);
  // Release work buffer for category array
 (*env)->ReleaseIntArrayElements(env, categoryArray, category_array, 0);
}
```

Listing 2: Extract information from ANN hardware.

```
     // Create instance of JFC dialog
 NeuralNet panel = new NeuralNet();
     // Create instance of Native class
 JZISC ziscCARD = new JZISC();
     // Initialize ANN card
 int initialized = (int)ziscCARD.JZ_Init(0);
 if (1 == initialized)
   System.out.println("cannot open ZISC device (type any key to exit)\n");
     // Query number of neurons in PCI board
 int numberOfNeurons = ziscCARD.JZ_GetMaxNeurons();
    System.out.println("The ZISC has " + numberOfNeurons + "neurons\n");
     // Get ANN version and type
 ziscCARD.info= ziscCARD.JZ_GetCardInfo(0);
 System.out.println("ZISC Info " + ziscCARD.info + "\n");
```

Listing 3: ANN learning procedure

```
/********************** learning *************************/
/* Symbol 0 is represented by 1 and symbol 1 is represented by a 5
/********* input a new set of 2 components (0 0) ***/
ziscCARD.array[0]= (byte)1;
ziscCARD.array[1]= (byte)1;
System.out.println("Vector (0 0) \n");
ziscCARD.JZ_PutVector(ziscCARD.array, 2);    /* input components */
ziscCARD.JZ_PutCat(1);              /* learn category 0 */
/********* input a new set of 2 components (0 1) ***/
ziscCARD.array[0]= (byte)1;
ziscCARD.array[1]= (byte)5;
System.out.println("Vector (0 1) \n");
ziscCARD.JZ_PutVector(ziscCARD.array, 2);    /* input components */
ziscCARD.JZ_PutCat(5);              /* learn category 1 */
/********* input a new set of 2 components (1 0) ***/
ziscCARD.array[0]= (byte)5;
ziscCARD.array[1]= (byte)1;
System.out.println("Vector (1 0) \n");
ziscCARD.JZ_PutVector(ziscCARD.array, 2);    /* input components */
ziscCARD.JZ_PutCat(5);              /* learn category 1 */
/********* input a new set of 2 components (1 1) ***/
ziscCARD.array[0]= (byte)5;
ziscCARD.array[1]= (byte)5;
System.out.println("Vector (1 1) \n");
ziscCARD.JZ_PutVector(ziscCARD.array, 2);    /* input components */
ziscCARD.JZ_PutCat(1);              /* learn category 0 */
```

Listing 4: Sample output

```
Recognize pair (0 1)
Status content after input vector
error flag= 0
deg= 0
unc= 0
full= 0
id= 1
Sequence of read dist read cat
========================================
CORRECT: dist = 0 Out= 1
```

▼▼▼ DOWNLOAD THE CODE ▼▼▼

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

Ask about the WINNER

JSCP™ Software Co-Processor for Java™ utilizing Sun's JVM

Proven in use on all leading RTOS's:

pSOS VRTX LynxOS VxWorks NucleusPLUS Windows CE DOS

On all major CPU's:

MIPS TriMedia x86 SH-3 i960 MicroSparc StrongARM ARM 68000 PowerPC

Recording faster development cycles, faster time-to-market in:

- Data Communications • Set Top Boxes
- Digital TV • Car Systems • Control & Automation
- Printers • PDAs and Hand Held Terminals
- Smart Terminals, Telephones, Cell Phones
- Wireless equipment • Web Phones

see us at JavaOne '99

JSCP is a trademark of NSI Com. Java and all Java-based trademarks and Logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and other countries.

NSICOM

E-Mail: info@nsicom.com
http://www.nsicom.com

SpatialVision

by Sedona Geoservices

Use geospatial data analysis in Java-based app for end users

JAVA DEVELOPER'S JOURNAL JDJ WORLD CLASS AWARD

by **Jim Milbery**

Recently I had the opportunity to work with Sedona Geoservices' SpatialVision, an end-user application for performing geospatial data querying, data visualization and analysis. SpatialVision is designed to help your organization harvest information from your data using a geospatial focus. Geospatial analysis can be defined as the process of comparing your relational data to location data – for example, where your customers are located on a map. Sedona has targeted applications as diverse as call center management, sales operations and even service dispatching as candidates for such analysis. The company estimates that over 80% of preexisting data archives have at least a portion of their data in an inherently spatial format.

The equipment I used for this evaluation was a Dell Pentium II with 200 MHz, 64 MB RAM, 4 gigabyte disk drive, Windows NT 4.0 (Service Pack 4), ViewSonic 15-inch SVGA monitor, 3COM Etherlink XL Adapter and 8x CD-ROM.

Installation and Configuration

You can download SpatialVision from the Sedona Geoservices Web site, but I used a CD-ROM to install it. SpatialVision works only with the Oracle Spatial Cartridge, and you'll need access to an Oracle database with the cartridge up and running. Sedona provides Internet access to their own demonstration database, however, so you can test the product without having your own Oracle database installed locally.

SpatialVision is a 100% Pure Java application that requires the JRE 1.2 environment to run. (Sedona provides the install kit for this version of the JRE as a link from the installation panel.) Once you have the JRE installed, you can use a second link from the main panel to install the product itself, which takes about five minutes and 10 megabytes of disk space. SpatialVision comes equipped with a series of manuals in Adobe PDF format, but the install program doesn't add them to the system menu for you. While this isn't a big deal, the product is targeted to end users, who may not be as comfortable searching for document files as a software developer would be.

Using SpatialVision

Once I installed the product, I opened up the Adobe tutorial, started SpatialVision and began playing around. Since I don't know much about geospatial data analysis, I relied on the tutorial for direction. The examples and images were sufficiently detailed and I found I could follow along reasonably well. I suspect that a user familiar with geospatial data analysis would find the tutorial even easier to use. I connected to my local Oracle database without difficulty via the database configuration panel, using a thin-client JDBC driver.

SpatialVision allows you to create groups of database connections and even to join data across different Oracle databases if you wish. Whatever information you need to connect to your database can be found on the configuration panel. The only parameter that may seem unfamiliar is the requirement to enter the Oracle System ID (SID) as the database value, which may be confusing to those who are used to providing Oracle TNSNAMES values. I was able to connect to my local Oracle database and select some data from my sample college application without difficulty. But I was stopped short of analyzing any of my data in my local database because none of it was stored in a geospatial format. Although the data in my preexisting Oracle database contains location information such as phone numbers and street addresses, SpatialVision can't analyze this data directly. To make use of SpatialVision you'll need to tie this data to a location on Earth through a process called *geocoding.* Geocoded addresses are stored in the Oracle Spatial Data cartridge, and your preexisting data is linked to these geocoded entries as attributes. It makes perfect sense when you think about it, but I didn't even consider this aspect of the problem when I started playing with SpatialVision. If you're already using Oracle's Spatial Data cartridge, this is unlikely to be a problem. However, if your data isn't already encoded, you won't have much success in using SpatialVision. Unfortunately, the product doesn't provide tools to assist you with the geocoding effort, but Sedona does offer consulting services to assist you. Sedona's marketing manager told me they plan to provide these types of utilities for a future release of the product. In the meantime, you'll need to work with Sedona directly.

At this point I thought I was going to be writing the shortest product review in history since I had no data to work with. I was pleasantly surprised, however, to find that Sedona makes their demo database available across the Web as part of the installation process. I was able to open a connection over the Internet using the Oracle thin-client JDBC driver to a Sedona database on a public server. I recommend a higher bandwidth connection than the 28.8 speed that I used for my remote testing. Nevertheless, despite the low bandwidth, I was able to connect to the Sedona database of information about locations in eastern Pennsylvania. SpatialVision includes a query-composer and editor for building queries against your database, and you'll find the panels easy to use for generating SQL queries against the database. On the query panel you can select both fields as well as display the range of data for a given field. For example, I selected the "restaurants" table and then searched for a list of restaurant names that I could use in a query for locating places to eat in eastern Pennsylvania. Part of the power of SpatialVision is its ability to provide extra geospatial intelligence in the formulation of queries. For example, I was able to search for all Burger King restaurants in the borough of Norristown.

You can also search for data using specialized geospatial values such as "within a radius of 50 miles," and this is the real power of SpatialVision. Information in your organization can be mapped back to geocoded locations and then also used in the formulation of queries. I can easily imagine using the product to find all customers that have ordered "blue widgets" within a 20-mile radius of Easton, Pennsylvania. Once you've located your data, SpatialVision can display it on a map and you're free to add your own images and symbols to the resulting display. SpatialVision can be deployed using an appletviewer, or you can embed the product inside a Web page for even greater flexibility. I found the data to be very usable over a 28.8 connection, so it's feasible to deploy SpatialVision to your organization over an extranet. However, given the volume of data you're likely to access, I'd strongly recommend a higher bandwidth connection for use with SpatialVision.

SpatialVision
Sedona Geoservices, Inc.
649 North Lewis Road
Suite 220
Limerick, PA 19468
610 495-6701
www.sedonageo.com
SpatialVision 1.1 is supported on Windows 95/98/NT and Sun Solaris 2.5.1 or higher.

Summary

I'm impressed with Sedona's efforts to provide a completely Java-based application for end users, and I like the geospatial data analysis concept. Oracle is certainly a good choice for the database, but you'll need to spend time geocoding your preexisting data to work with SpatialVision. Sedona's forthcoming utilities may be useful for organizations that want to make the jump to geospatial analysis but aren't sure how to get there.

About the Author
Jim Milbery, an independent software consultant based in Easton, Pennsylvania, has over 15 years of experience in application development and relational databases. You can visit his Web site at www.milbery.com.

jmilbery@milbery.com

Interland

www.interland.net

CASE STUDY

Universal ORB Will Help Networks Make Enterprises More Competitive

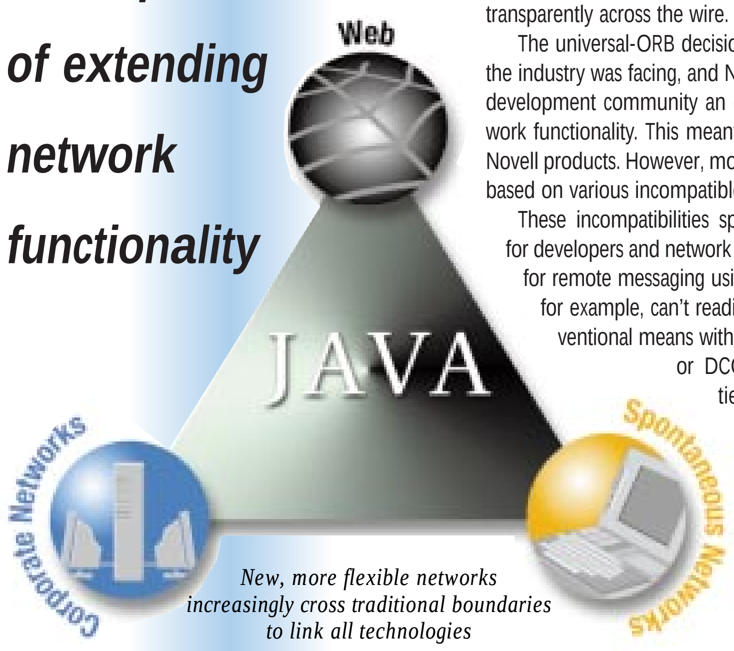
by **Art Nevarez**

Novell looks for – and finds – a way to give developers an easy solution to the problem of extending network functionality

When Novell, a global leader in networking, sought to implement powerful new Java-centric object request broker (ORB) technology in its products, the company forged a strategic partnership with ObjectSpace, Inc., an emerging leader in the distributed computing market. ObjectSpace's Voyager product family is 100% Pure Java and standards-neutral, and enables CORBA and RMI objects to communicate transparently across the wire.

The universal-ORB decision addressed a dilemma the industry was facing, and Novell wanted to give the development community an easy way to extend network functionality. This meant embedding an ORB in Novell products. However, most ORBs in use today are based on various incompatible standards.

These incompatibilities spawn difficult challenges for developers and network users. An object enabled for remote messaging using the CORBA standard, for example, can't readily communicate by conventional means with objects enabled for RMI or DCOM. ORB incompatibilities can be resolved through a combination of bridge software and manual programming. However, Novell rejected this alternative in an effort to free developers from complexity and extra labor rather than burden them.



New, more flexible networks increasingly cross traditional boundaries to link all technologies

Speed Development, Simplify Design

Novell needed a simpler, more straightforward solution – a universal ORB, with components that could communicate transparently with any client object and with clients that could interact with any server. Such an ORB could streamline application design, shorten development cycles, reduce costs and make applications easier to maintain and modify.

As a result, Novell and ObjectSpace are partnering to bring universal ORB functionality to networking customers, combining it with Java portability and the ability of directories – the heart of next-generation networking – to make networks easier to use and manage.

Leverage Opportunities

Together, the universal ORB, Java and advanced directories, such as Novell Directory Services (NDS), will enable companies to make people more productive and business operations more efficient. At the same time it will help companies to readily transform emerging opportunities, such as e-commerce, into competitive advantages.

To accomplish this, Novell is working with these technologies to achieve four key objectives:

1. Protect customers' investments in Novell platforms by extending the service life of its installed base into future heterogeneous environments.
2. Tightly integrate legacy networks and data infrastructures with next-generation networks based on distributed object technology.
3. Embrace telephony, JINI, Universal Plug and Play (UPP), nontraditional networks and other technologies through dynamic network support.
4. Integrate multiple infrastructures and transparently bring the result into people's lives by providing users with stable, secure and mobile identities on the network.

We believe that Java and a universal ORB are absolute requirements for efficiently achieving these objectives in today's real-world networking environments.

Generate Power at Runtime

To be universal, an ORB must provide a neutral meeting ground for multiple standards. Voyager holds a critical advantage in this regard. Java has dynamic capabilities that can enable objects to be extended at runtime, providing vast architectural flexibility and power. However, these capabilities are implicit only in the Java specification and are not implemented in the JDK.

A universal ORB enhances a protocol or platform-specific ORBs by allowing servers and clients to interoperate with other protocols, platforms and programming models without requiring an extensive retooling and recoding of the existing applications and services. Voyager leverages Java's dynamic capabilities, for instance, by transparently adding to any remote object at runtime all the "plumbing" it needs – stubs, skeletons, helper classes – to generate CORBA bindings. And by generating proxies transparently, also at run-

About the Author
Art Nevarez is chief architect for Novell's Java Technology Group, which produces the JVM, the NetWare SDK and a number of API- and Java-related technologies. A member of several Novell policy-generating bodies, Art holds a BS in computer science from Brigham Young University. You can e-mail him at art@novell.com.

art@novell.com

time, the ORB allows components to interact across the wire without being aware that they are communicating remotely. Source code isn't required to accomplish this "magic" – remote messaging is enabled even when source has been lost.

Preserve Investments, Minimize Change

NDS is key to Novell's Java-centric distributed environment. "Write once, run anywhere" portability makes Java the platform of choice because it lets Novell, their customers and industry partners rapidly develop network services and applications for heterogeneous, Internet-enabled environments.

Combining the use of NDS with Java and Voyager will help developers at customer companies integrate directory services with enterprise databases and applications. The ORB will transparently provide the object communications plumbing, thereby saving time, effort and cost.

There are far-reaching implications. One of the biggest challenges facing Novell's customers is how to efficiently integrate static, legacy data structures – "information silos" – with next-generation dynamic environments.

The goal is to use stored and real-time data with equal facility. This is important because companies have been amassing vast data assets in silos and other static inflexible structures for some 40 years, and continue to do so. Static code and data structures contain much of the logic and information that run businesses today, including e-commerce. They sit on disk in static structures – or are constrained by static bindings to network protocols, security mechanisms and platform-dependent APIs.

Competitive advantage depends on dynamically leveraging this data by, for example, enabling Web sites to customize themselves on the fly to individual visitors, and enterprise networks similarly tailoring themselves to individual users.

However, static and dynamic systems are based on largely incompatible programming paradigms. Development teams can get them to interwork, but not easily. And when business or technology changes occur, these interfaces must be manually reworked, a relatively slow, costly process, making the whole enterprise less competitive.

Directory services combined with a universal ORB and related technologies let static and dynamic systems seamlessly interwork with little development effort. Programmers using Voyager readily create objects or object references to static enterprise records. They make these records look and behave like real, dynamic data objects that can be directory-programmable from Java, CORBA and, soon, DCOM. The directory provides the stable identities for all the principals involved, and insinuates them into the existing management fabric of the enterprise.

As a result, it becomes remarkably easy to "inject life" into static data structures and code by transforming them into distributed objects that can be used dynamically. Further, dynamic capabilities open the entire enterprise code base to virtually any new protocol or standard. This enables companies to quickly gain new technology advantages while preserving IT investments and minimizing change.

Raise IT Productivity

A pure Java universal ORB will raise developer productivity at Novell customer sites in numerous ways. It's already making their own development team more productive. Dynamic capabilities plus a clean user interface are enabling them to rapidly make sophisticated functionality available to their customers through directory services and other Novell products while controlling their development costs. The company gains productivity advantages several ways:

- They instantly remote-enable any Java class without modification by generating proxies at runtime. As a result, their developers don't have to waste time and effort on stubs or helper classes.
- They use regular Java syntax to build remote objects, invoke messaging, propagate objects around the network and perform virtually all other development tasks.
- Their programmers have full native CORBA support for IDL, IIOP, bidirectional IDL/Java conversions and key RMI interfaces. All that is required is writing just a single line of code to dynamically CORBA-enable Java objects at runtime without modification.

Other capabilities, such as Dynamic Aggregation, an ObjectSpace exclusive, let Novell extend objects' characteristics and behavior on the fly.

Achieve Rapid Change

Novell's choice of ORB offers a multitude of advantages in addition to universality. Mobile agent technology embedded in the ORB enables the network directory to become even more programmable. Because the directory is the repository of user identifications and enterprise resources, programmability gives customers another powerful tool for achieving rapid change at low cost – a key competitive advantage.

From Novell's standpoint, the realistic way to achieve programmability today at the directory level is to take whatever representations of humans, products and data exist on the corporate backbone and wrap them with programmable objects.

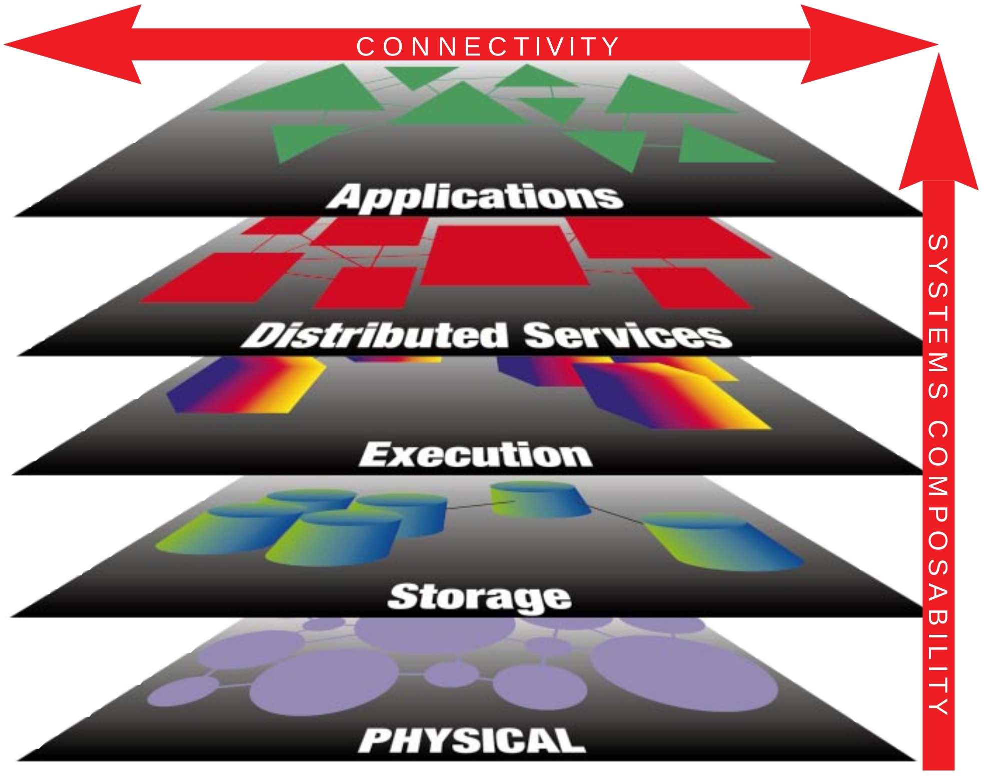
How might this work in a production setting? Suppose an IT administrator wants to distribute a custom spreadsheet component or any other piece of code. The code must go to heterogeneous clients across the enterprise. The administrator uses the directory's console to make choices about deploying this component, then the ORB transparently implements the choices by encapsulating the component in a wrapper that is actually a mobile agent.

This "smart wrapper" knows what it has to do and how to do it. It knows how to interact with the directory, navigate the network and deliver the component to designated clients.

The wrapper replicates itself and its cargo across the network to servers for deployment as specified by the administrator. The distributed component might be a Windows executable, a Java class, an ActiveX component – just about anything. It could be specialized code for a handful of users or a software update going out to thousands of clients across disparate systems enterprise-wide.

Throughout, the infrastructure remains transparent while giving the administrator tremendous control, flexibility and productivity. The administrator simply drops the code on a server, or in a directory container, and points to where it should go. The directory and Voyager do the rest.

The simplest, most immediate instance of this scenario takes place within the server domain, which has its own security safeguards. The power kicks in when servers, not humans, generate the dynamic code.



Complex, diverse environments like this one demand use of a standards-neutral ORB.

Capabilities like these let the network – directories, network servers and other services – automatically distribute code without requiring developers to target any particular networking infrastructure or protocol. IT productivity rises because developers are able to focus on solving specific business problems and deliver interfaces to the resulting logic. Administrators running the network are then free to focus on distributing and "networking" the objects these developers create.

Enhance Data Value, Simplify Administration

"Smart wrappers" and other ORB capabilities will make enterprise systems more robust and useful while reducing IT management costs.

Making Data More Useful

E-commerce is driving the need for Web sites that customize themselves to the individual's unique needs and interests. ORB capabilities will make it easier for networking customers to use their legacy databases to populate Web pages with individualized information. The directory will provide the personalization; Voyager will deliver database records in real time by dynamically wrapping them on the fly. This in turn will make it easier and less costly to create sites that "sell" more effectively and enrich companies' relationships with their customers.

Simplifying Systems Administration

When the network management service needs to manage a workstation, IT professionals won't have to preinstall a network management agent on that client. Instead, the service will send a mobile object that knows how to get there and, upon arrival, "unpack," install and activate the service. The target device could be a PDA or cell phone, enabling the device to readily consume network services such as accessing corporate address books, Web gateways and e-mail. Lite Client, a recent breakthrough in the Voyager ORB technology, lets developers remote-enable applets that leverage dynamic proxy generation and other advanced capabilities, while adding only a 14 K JAR file to the footprint. The directory will provide the pervasive management "dock" for these agents.

Making Networks More Flexible

When a console requires an alternative means of controlling a node on the network, such as when using ActiveX components for accessing network services, the ORB facilitates it by providing the service locally or delivering it remotely. This allows unequaled network-level diversity, opening up heterogeneous networks to all devices on the network.

Connect Globally and Personally

In the very near term, Novell believes, networks will dynamically interact to whatever degree is permitted by their security features and user-defined settings. Using directory-based credentials, a user will be able to send out delegates as mobile agents to act on their behalf. These agents will not only roam the user's own enterprise but will collaborate with networks worldwide.

Universal ORB technology has the unique ability to easily enable mobile agents and other capabilities, making it a perfect fit for achieving dynamic network flexibility – not just across the enterprise, but on a truly global scale.

Novell strategists also believe JINI will quickly become another important player in networking scenarios. JINI's auction-style capabilities will help them design more efficient workgroup networks with enormous flexibility.

ObjectSpace has already announced Voyager support for JINI. Combining directory services, Java, JINI and the universal ORB will provide an unprecedented "critical mass" of technology, propelling networking into a new, more productive era. It will give Novell and their customers even greater freedom to build networks that connect more globally than ever before – and also more personally, as these networks increasingly customize themselves to individual user needs.

Snowbound

www.snowbnd.com

Spring Internet

events.internet.com/summer99

INTERVIEW

SYS-CON RADIO

Broadcast live from the Jacob Javits Center in New York City, SYS-CON Radio's Chad Sitler spoke with David Norris of ObjectSpace

David Norris
President and CEO
ObjectSpace

JDJ: Let's start off with your product, Voyager Pro. Can you give us an idea of what it is, what it does and what makes it appealing to the consumer?

David: Voyager is a product line that is focused on building distributed applications. It provides work for the different standards that have emerged, CORBA, D-COM, RMI and overall integration with the Java language and technology. Voyager provides substantial capabilities to simplify the building of distributed systems. By integrating all of these different standards together, it provides the capabilities to build applications that can interoperate and integrate existing technologies as well as some of the new standards that are coming out. We have announced our support for some emerging standards, such as JINI and Enterprise JavaBeans, and we'll be introducing products that support those technologies.

JDJ: Now suppose I'm the consumer, the Java developer, and I'm out there looking for a prime product for my development – what is going to make me choose yours over anybody else's?

David: A couple of things. You can download it right off our Web site. You can try it out, and it has received significant accolades. In fact, at the show we won the Best Product Award for Voyager [*JDJ* Editor's Choice Awards] and some of our other technologies have won awards. So we believe in a direct model distribution right off our Web site. We provide substantial white papers and technical assistance right there online, which allows you to get started very easily. And then, as you begin to use the product, some of the differentiations that it offers become very obvious to you. So I believe one of the main reasons people use our product – in fact, it is used by over 10,000 companies today worldwide – I believe the reason is that it's very accessible and it provides substantial value that is easy to actually get started with.

JDJ: What do you see in the immediate future for your company and your product?

David: Well, our goal is to make Voyager, the product line, very pervasive to really get everybody that is trying to get into distributed computing and build significant Enterprise applications to use our technology. So when they think of building distributed systems, they will think of Voyager and the way that Voyager can simplify their life. Over the next couple of years I would like to see a lot of people begin to use Voyager. We're shipping it with a lot of the development tool vendors such as VisualAge for Java and Symantec's Visual Café. We'll be exposed to a lot of Java developers. And in the next couple of years, with the introduction of Enterprise JavaBeans or some of the higher level functionalities such as JINI, I believe we can become a pervasive technology and really get widespread adoption so that people can build systems much simpler than they could in the past.

JDJ: You mentioned before about a Web site where you can download a demo. Can you give us the Web address?

David: Sure. If you go to www.objectspace.com, you'll see the company information there. If you go to the product section, you can download Voyager directly. You can also have access to all the technical white papers and comparisons and all that, that're online.

Certify On-line

certifyonline.com

Slangsoft

www.slangsoft.com

Outsourcing Development One Bean at a Time

Companies post requests for specified JavaBeans that are then bid on by developers

by Charles Stack

The JavaBean component model presents entirely new ways of developing software. Once a component interface is specified, the actual implementation can be accomplished by another programmer down the hall, across the country or anywhere in the world. This allows outsourcing of software development at the component level.

At Flashline we're publicly testing a new service called Beans by Design, which allows companies to post requests for specified JavaBeans that are then bid on by developers. This service is sort of a cross between an eBay auction and a Match.com dating service. Using a double-blind bidding system, we help match developers with companies seeking component development: "SWD seeks EJB for software application!"

A component outsourcing service is a logical outgrowth of the general trend toward outsourcing software development. Outsourcing information technology is an increasingly popular solution for corporations unable to meet the burgeoning demand for software. IT outsourcing is a $100 billion business (IBM, Yankee Group) growing at 15% annually (Giga). Outsourcing software development offers many advantages:

- Access to greater pool of resources
- Cost control
- Cost savings
- Access to specific expertise
- A focus on core competencies
- Shorter development cycle
- Flexibility in meeting demands

Outsourcing at the component level offers not only these traditional outsourcing advantages, but additional ones as well. Because you're only outsourcing components and not an entire application, your risk is reduced. You remain in control of the overall development process. Software components by their very nature lend themselves to a much more precise description than entire applications. If you're doing large-scale Java development, you're using an object-oriented modeling system with UML to specify your system's development. Once you've defined the interface specifications for each component, it's quite straightforward to parcel out development to individual developers.

Component-based development is poised for explosive growth. It's faster, more reliable and easier to maintain. I highly recommend Jacobson, Griss and Jonsson's book, *Software Reuse* (Addison Wesley Longman), which outlines order-of-magnitude (5x–10x) improvements in development speed, reliability and maintenance from reusable software components. The GartnerGroup projects that 70% of corporate applications development will utilize components by 2003. For components that lie outside your in-house development expertise or exceed your available resources, outsourcing their development will come quite naturally.

Beans by Design sits at the confluence of these two powerful forces, combining the benefits of component-based design with the benefits of outsourcing. We've implemented this service on our Web site, where it reaches a global marketplace of developers and companies. Flashline is the first to implement this revolutionary approach to software design and we'd welcome some feedback.

Here's how Beans by Design works (see Figure 1). Companies seeking JavaBeans post a request on our Web site describing the component – name, description, interface specification, licensing terms and delivery dates. They can also attach files containing detailed product specifications. Once submitted, e-mail notification is sent to registered developers announcing a new bean request.

To become a registered developer you enter your e-mail address, company name and country. This information isn't publicly displayed on the Web site – your identity is revealed only if you submit a winning bid. Additionally, you enter your areas of technical expertise, spoken languages, development languages, years of experience, certifications, number of developers, billing rates and expertise keywords. This information is publicly displayed (see Figure 2).

After registering as a developer, you begin to receive e-mail notification of bids. E-mail is used extensively throughout the system to notify you of all relevant events. It's sent to and from blind e-mail addresses and then automatically forwarded to the correct address. This allows easy communication between users while retaining complete anonymity. You review compo-

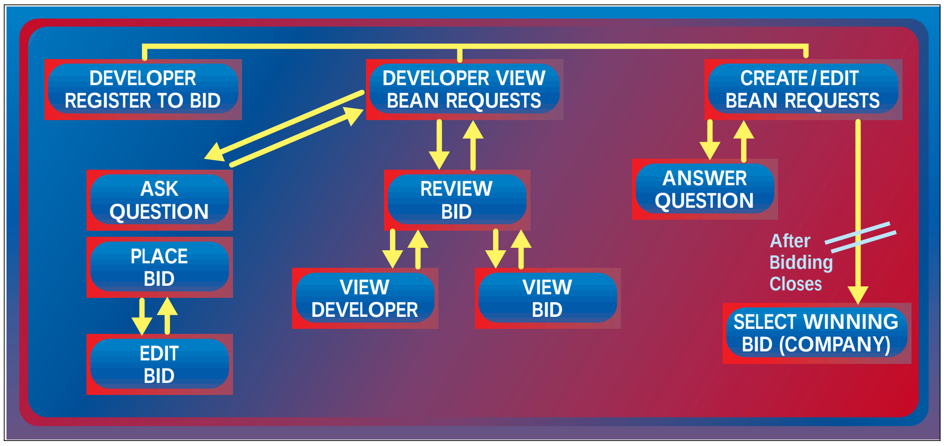

DEVELOPER REGISTER TO BID
DEVELOPER VIEW BEAN REQUESTS
CREATE / EDIT BEAN REQUESTS
ASK QUESTION
REVIEW BID
ANSWER QUESTION
PLACE BID
VIEW DEVELOPER
VIEW BID
After Bidding Closes
SELECT WINNING BID (COMPANY)
EDIT BID

Figure 1: Beans by Design flowchart

KL Group

www.klgroup.com

nent requests on our Web site as they become available. You can also download and review any file attachments. Through a fully integrated question-and-answer facility you can post questions to be answered by the bean requester. Questions and answers are publicly posted with the request. Our brief experience (Beans by Design had been open only a few weeks at the time this article was written) has shown the Q & A feature to be extremely valuable in clarifying and refining the requests.

To bid on a request you simply enter a price in U.S. dollars and any necessary notes or terms. The company requesting the component is e-mailed notification that a bid has been entered.

To prevent confusion over what was originally entered, the original bean request (see Figure 3) can't be edited. However, the requester can add information to the request at any time. This information is date-stamped and appended to the original request.

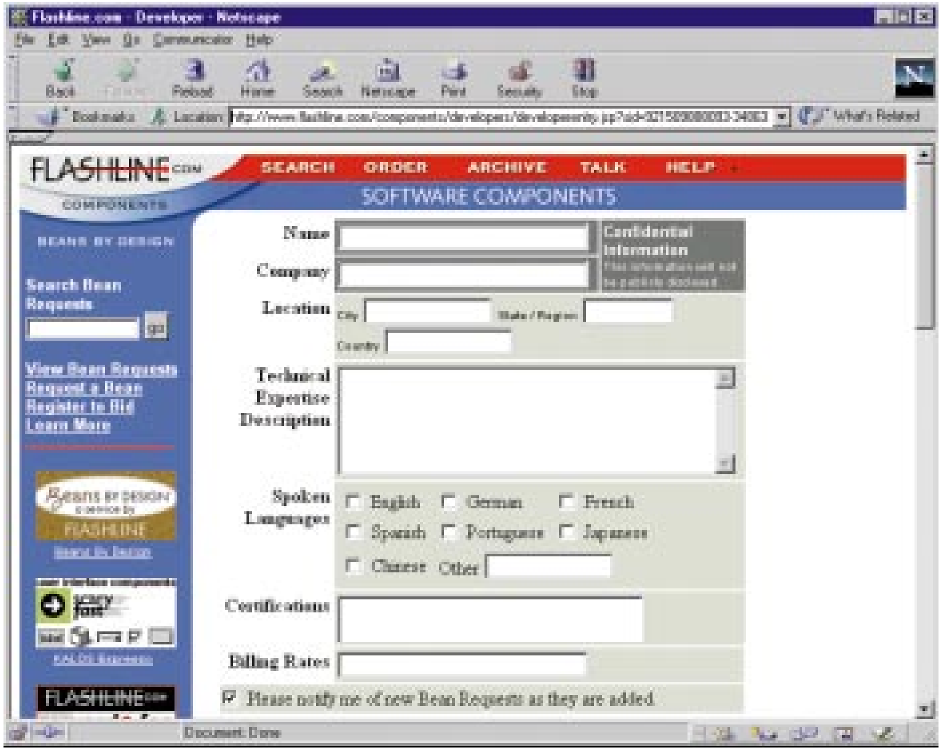
The average request is open for 24 days and receives three bids. Companies requesting components can review the bids and bidders at any time. The system allows requesters to ask confidential questions of any bidder. To aid in the bid review process, requesters can segment the bids into several categories. When first received, the bid is marked NEW. The requester then reviews the bid, and the bidder can mark it either REJECT or INCLUDE. This allows easy ongoing review of the bids as they arrive.

Requesters review the bidder information online. They can view their areas of expertise, number of developers, previous projects and technical certifications. They can request additional information from the bidder by blind e-mail. Most important, they can review feedback from other companies that have used this development company through Flashline.com Beans by Design.

Once the bidding closes, the requester is free to accept a bid. The winning bidder is notified by e-mail. The requester and developer then contact each other directly to begin development and Flashline's direct involvement in the process ends.

Upon accepting a bid, the requester is asked to briefly explain why it was accepted. The requester may indicate that it was the lowest priced, or the bidder had the most experience in a particular area, the best credentials, or the best online reviews – or any other applicable reason. This information is e-mailed to nonwinning bidders as feedback so they can improve their bidding.

After the JavaBean component is delivered, Flashline requests feedback on the


Flashline.com - Developer - Netscape
File Edit View Go Communicator Help
Back Forward Reload Home Search Netscape Print Security Stop
Bookmarks Location: http://www.flashline.com/components/developers/developerentry.jsp?sid=9215090800033-34083
What's Related
FLASHLINE.COM COMPONENTS
SEARCH ORDER ARCHIVE TALK HELP
SOFTWARE COMPONENTS
BEANS BY DESIGN
Search Bean Requests
go
View Bean Requests
Request a Bean
Register to Bid
Learn More
Beans by Design a service by FLASHLINE
Document: Done

Name

Company

Confidential Information

Location City State / Region Country

Technical Expertise Description

Spoken Languages ☐ English ☐ German ☐ French ☐ Spanish ☐ Portuguese ☐ Japanese ☐ Chinese Other

Certifications

Billing Rates

☑ Please notify me of new Bean Requests as they are added.

Figure 2: Developer registration screen

developer. This feedback is then incorporated into the developer's database for review by companies considering using the developer in the future.

Preliminary results for Beans by Design are encouraging. All requests have received multiple bids. Over half were successfully awarded. Unsuccessful requests seem to fall into two categories: either no bids were deemed acceptable or the request was met through other means – a noncomponent solution was found or a prewritten component was located. There may also be a certain amount of testing the waters just to see how the system works and what type of response might be generated.

Interestingly, the dollar range between the lowest and highest bid on a single request has been surprisingly large, often 10x or more. This, we think, reflects the immaturity of the component industry more than anything else. It's also reflective of the opportunities for large cost reductions compared to outsourcing entire applications. Developers who have extensive experience and libraries of already written components can deliver components cost-effectively.

Currently, more than 250 developers with a wide range of experience are registered from 14 different countries.

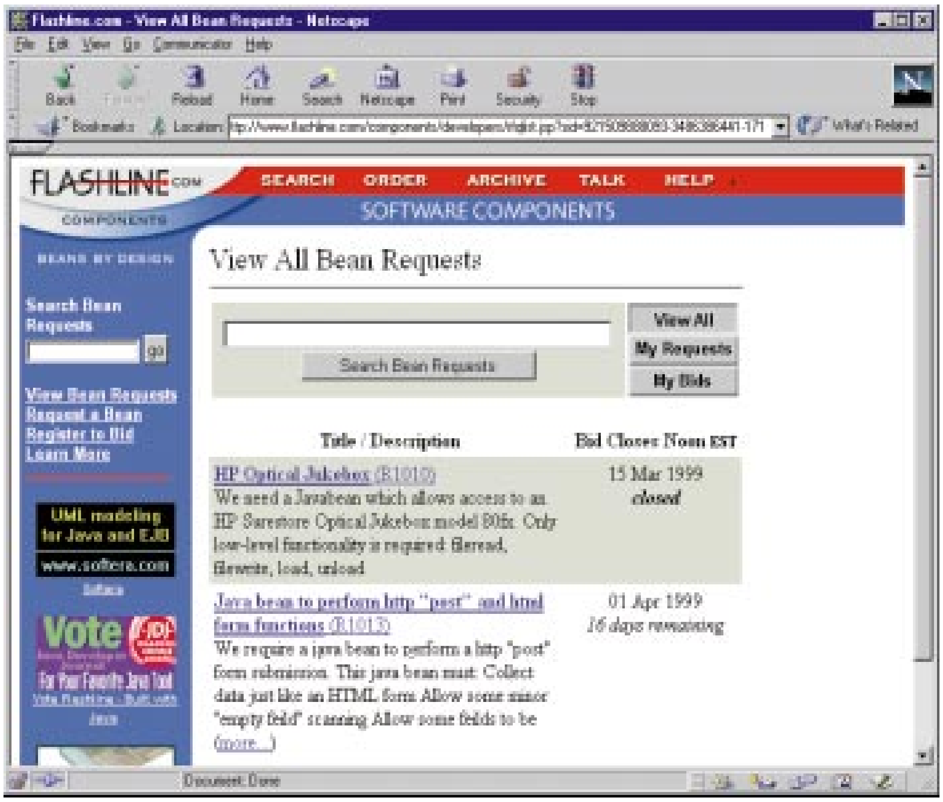
Beans by Design is currently a free service. Flashline's primary mission is to build an online marketplace for component software. We sell downloadable JavaBean components from over 40 vendors, and are completely committed to building a true software component industry. The service will increase the number of components in the marketplace. Every filled request means one more component. Many developers retain intellectual property rights to the custom components. These components can then be commercialized and listed in the Flashline database of prewritten components. Beans by Design also generates revenue for component developers, which should increase their number and profitability. Most essentially, it provides a source for components that would not be available otherwise, which should aid companies that have committed to migrating to component-based architectures.

As Beans by Design progresses, we may charge a nominal annual fee and a listing fee for each component request. We believe these nominal fees would act to weed out inactive developers and eliminate the possibility of frivolous requests.

Under the "Eat your own dogfood" theory of software development, our entire e-commerce Web site, including Beans by Design, has been developed using JavaBeans. We use LiveSoftware's implementation of the Java Server Pages (JSP) specification to embed JavaBeans within HTML pages and then call methods as needed for session management, forms processing, database access, credit card processing and file downloads. The JSP JavaBean approach allows us to separate presentation from logic -- and programmers from designers. Keeping designers out of software code and keeping programmers from

SalesVision

www.salesvision.com


Flashline.com - View All Bean Requests - Netscape

View All Bean Requests

Title / Description	Bid Closes Noon EST
HP Optical Jukebox (B1015) We need a Javabean which allows access to an HP Surestore Optical Jukebox model 80fx. Only low-level functionality is required: fileread, filewrite, load, unload	15 Mar 1999 *closed*
Java bean to perform http "post" and html form functions (B1013) We require a java bean to perform a http "post" form submission. This java bean must: Collect data just like an HTML form. Allow some minor "empty field" scanning. Allow some fields to be (more...)	01 Apr 1999 *16 days remaining*

Figure 3: Bean request screen

trying to do HTML design qualifies as a "really good thing." They don't share the same skill sets, and they barely speak the same language. The JSP/JavaBeans approach to Web site programming has proved to be robust, reliable and reusable. It is also eminently scalable and about 200 to 300% faster than traditional C/CGI programming.

There is an enormous amount of work to be done to create the vibrant software component marketplace we visualize. More Java programmers are needed. Better tools are needed. We particularly need to enhance UML to support component architectures. Sun and IBM must meet the announced targets for development of the Enterprise JavaBeans specification. Application servers, which function as component containers, need to mature. The benefits of JavaBean software components – faster development, increased reliability and reduced maintenance – present compelling motivation. Beans by Design offers one small piece to this immense and exciting challenge.

About the Author

Charles Stack is the founder, president and CEO of Flashline.com, a software component marketplace servicing IT professionals. His Books.com, the first retail store on the Internet, is credited with making the Internet's first sales transaction, in 1993. A graduate of Case Western Reserve School of Law, Charles can be reached at cms@flashline.com.

cms@flashline.com

J-Spell

www.wallstreetwise.com

The Theory Center

www.theorycenter.com

Pervasive

www/pervasive.com

Cascading JFrames

Making your infrastructure invisible and easy to use

by **David Anderson**

If you haven't tried it yet, Swing is Good. For those of us who've had to wrestle with the java.awt to build GUIs, Swing is a much simpler and more powerful alternative. With its "coming soon" status in the com.sun.java.swing classes in JFC 1.0 upgraded to "officially blessed into Java" as javax.swing in JDK 1.2, Java application developers everywhere should be pleased. Although it's only a Java extension and not part of the core library, we should be fine since right now we really don't need GUI support for doorbells, toasters or electric frying pans.

For general Java programming, Swing works pretty well out of the box. Text fields, text areas, buttons, labels, lists and trees cover a large majority of what the average GUI developer needs. However, for harder-core development, Swing still needs some customization. I'm an infrastructural programmer at heart; every time I need to customize something in Java, I try to do it right, once, and add the improvements to my personal Java packages. Every time I do a new project my toolbox expands a little bit more. While working on a recent project I needed to build a facility for automatically positioning windows. Over the years, I've hacked autopositioning several times in C, C++ and Objective C. This time I decided to do a nice, clean Java implementation.

The Autopositioning Problem

The class that most developers use as the base for a window on a screen in Swing is JFrame. Left to themselves, JFrames are initially positioned at (0,0). When they're shown, each new JFrame will stack up, one on top of the other, with all of their origins at (0,0) – obscuring previously created JFrames, as shown in Figure 1a. In any real application, this gets to be difficult for the user: each new window has to be selected and moved by hand. Autopositioning should be done prior to the first appearance of a window, in a way that lets the user identify it easily in relation to its neighbors.

The JFrame inherits the java.awt.Frame lineage, along with its ancestor java.awt.-Component handling positioning. A component is positioned using its setBounds() method, and the new intended position and size of the component are provided as arguments. Assuming we know the size, we need to assign a reasonable new (*x*,*y*) location to the JFrame for autopositioning prior to the first call to setBounds().

A popular strategy for keeping windows from obscuring each other is by cascading them – staggering them within an area so that new windows roll across it in an orderly fashion, each slightly offset from the previous one, as shown in Figure 1b. When the position reaches a predetermined limit, it's wrapped around and the staggering cycle starts again. Many windowing applications use this strategy today, and it's the one I constructed, but in a very robust way.

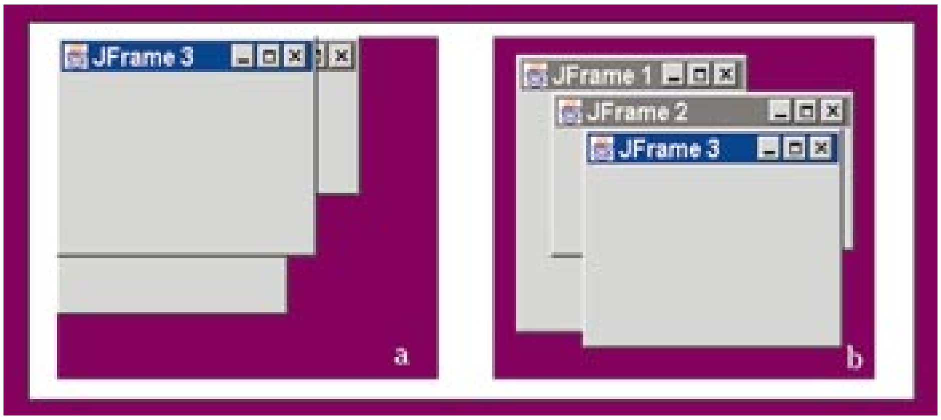


Figure 1: Several JFrame-based windows in (a) default positions, and (b) cascaded positions

The Cascade Object

The workhorse of our code is the Cascade object. A Cascade implements the cascading behavior in one dimension: an instance steps across locations within an area, wrapping around at the end. The object has two principal behaviors, one to get the current location:

```
public int location ()
```

and another to increment it:

```
public void increment ()
```

When a Cascade is created it is set with default values that implement a default staggering and wrapping behavior. These values will all have set and get methods so that custom cascading may be configured. Each Cascade requires four parameters to implement a robust behavior: an offset, a position, a step and a wrap, related schematically in Figure 2. The location is equal to the offset+position, and when the Cascade is incremented, step is added to position. Position is then effectively adjusted to be within wrap's limits via a modulus function. The values for wrap must be greater than zero, while the values for offset and step may be assigned any value. Although the initial value of position may be set arbitrarily, subsequent values are calculated using the other parameters.

Besides these behaviors, there are the requisite canonical inclusions for serialization, cloning and content-based equality tests. It's always a good practice to throw them in because you never know what you may want to do with the object later, and they're easier to add while you're writing the code rather than as an afterthought. In addition to the default constructor, a version that takes the four parameters is also provided. The source code for the Cascade class appears in Listing 1.

It should be noted that after enough cycles of stepping and wrapping, the generated position will begin to repeat. That's the nature of a modulus function applied to

Instantiations

www.instantiations.com

ADVERTISER INDEX

Advertiser	Page
BEA WebLogic	2
www.weblogic.beasys.com	800-817-4BEA
borland.com	13
www.interbase.com/products/demojdj.html	800-451-7788
borland.com	35
www.borland.com/jbuilder/ads/javadev	800-336-6464
ColdFusion Developer's Journal	25
subscribe@sys-con.com	800-513-7111
Career Opportunity Advertisers	65-69
	800-582-3089
Cerebellum Software	6
www.cerebellumsoft.com	888-862-9868
CertifyOnline.com	48
www.certifyonline.com	877-528-2937
DevelopMentor	27
www.develop.com	800-699-1932
EnterpriseSoft	11
www.enterprisesoft.com	510-742-6700
Flashline.com	21
www.flashline.com	216-861-4000
InetSoft Technology Corp.	11
www.inetsoftcorp.com	732-235-0137
INPRISE Corporation	4
www.inprise.com/appserver	800-336-6464
Insigna Solutions	31
www.insignia.com	800-848-7677

Advertiser	Page
Instantiations, Inc.	57
www.instantiations.com	503-612-9337
InterLand	43
www.interland.com	800-217-0985
Java Developer's Journal.com	20
www.javadevelopersjournal.com	800-513-7111
JBuilder Developer's Journal	13
subscribe@sys-con.com	800-513-7111
KL Group	17
www.klgroup.com	888-328-9596
KL Group	51
www.klgroup.com/hunt	888-328-9597
KL Group	72
www.klgroup.com/culprits	888-328-9597
Microsoft Corporation	36-37
www.msdn.microsoft.com/visualc	800-509-8344
NSI Com	41
www.nsicom.com	977-480-3311
Object Domain Systems Inc.	19
www.objectdomain.com	919-461-4904
Object International	23
www.togetherj.com	919-772-9350
ObjectSpace	71
www.objectspace.com/train_now	800-OBJECT 1
OneRealm, Inc.	7
www.onerealm.com	303-247-1284

Advertiser	Page
Pervasive Software	55
www.info@pervasive.com	800-884-6235
ProtoView	3
www.protoview.com	800-231-8588
Riverton Software Corporation	29
www.riverton.com	781-229-0070
Sales Vision	53
www.salesvision.com	800-275-4314
Summer Internet World 99	47
www.events.internet.com/summer99	800-500-1959
Slangsoft	49
www.slangsoft.com	972-375-18127
Snowbound Software	46
www.snowbnd.com	617-630-9495
Software Development 99	62-63
www.sdexpo.com	800-441-8826
Specialized Software	61
www.SpecializedSoftware.com/jdj/	800-328-2825 x6576
The Theory Center	54
www.theorycenter.com	888-843-6791
Tidestone Technologies	25
www.tidestone.com	888-880-0665
Wall Street Wise Software	55
www.wallstreetwise.com/jspell.html	212-348-5031
WEB99	61
www.mfweb.com	800-441-8826

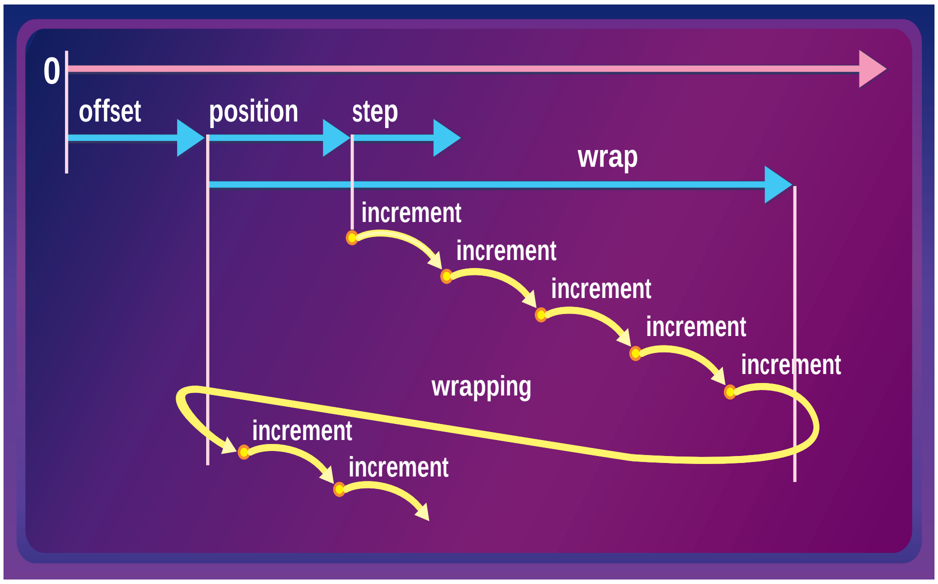


Figure 2: The relationship between the values in a Cascade and the results of successive increments on the location

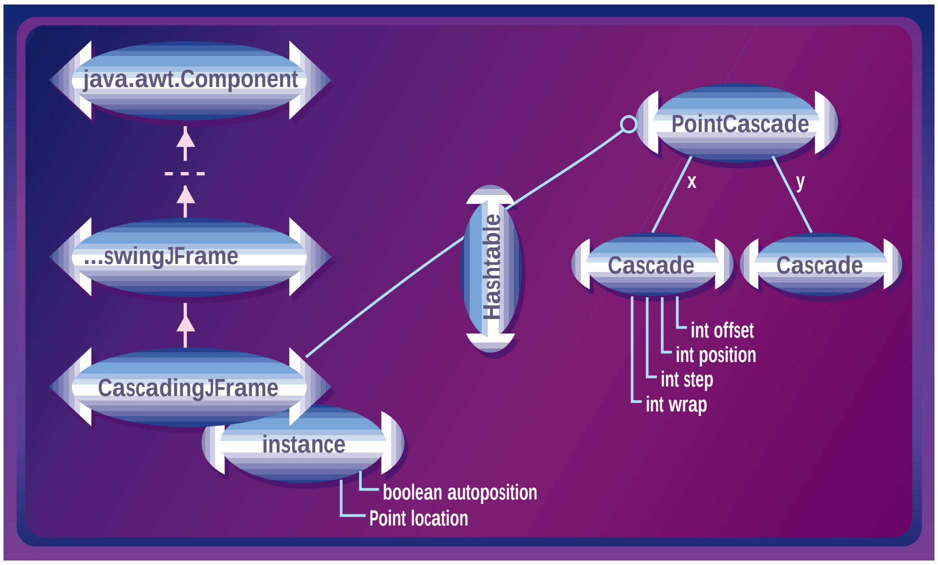


Figure 3: The cascading JFrame inherits from Swing's JFrame and, in turn, the java.awt.Component class, which manages positioning. The class contains a static hashtable whose values are PointCascades and keys are arbitrary names managed by the developer. The PointCascade contains two orthogonal Cascades, one each for the x and y dimensions. When an instance of CascadingJFrame is created, the class is accessed to retrieve the next location for the named PointCascade. When the instance's setBounds() method is subsequently called, the component is autopositioned at the location.

a monotonically increasing value. Even though it can't be eliminated, by choosing the step and wrap values carefully, the cycle of repetition can be made quite long – long enough so that only applications that are incrementing the Cascade many, many times will cause it to cycle.

The PointCascade Object

In order to cascade a two-dimensional location, you simply use two Cascade objects, one for the *x* dimension, and another for the *y* dimension. The PointCascade class wraps together two Cascade objects under one interface. It allows access to the underlying Cascades directly or manages them indirectly through messages that take java.awt.Point objects as arguments. The same set and get methods that are provided in Cascade are also provided in PointCascade, but with Points as arguments instead of integers. Methods to set and get the xCascade and yCascade are provided as well.

The signature of the increment() method is the same in the PointCascade as it was in the Cascade, but the PointCascade's location method now returns a Point object. This point is cascaded in *x* and *y* – we'll use it to position a JFrame. The staggering and wrapping can be set independently for the *x* and the *y* coordinates so a large set of noncycled, cascaded points can be created. However, if the Cascade defaults are used, the *x* and *y* coordinates will increment in lockstep, and only points on a diagonal will be generated. Again, the way to overcome this is to set up the parameters of the embedded Cascade objects carefully. By adjusting the step and wrap values of either or both of the Cascade objects, the cascading can be made to behave very differently in *x* and *y*. The source code for PointCascade is in Listing 2.

The CascadingJFrame Object

To make JFrames cascade themselves automatically, a new location must be retrieved from the PointCascade object when a new JFrame is created, and then the PointCascade must be incremented. Subsequently, when the JFrame is first positioned, its setBounds() method places the JFrame at the new location. The easiest way to do this in Java is through inheritance.

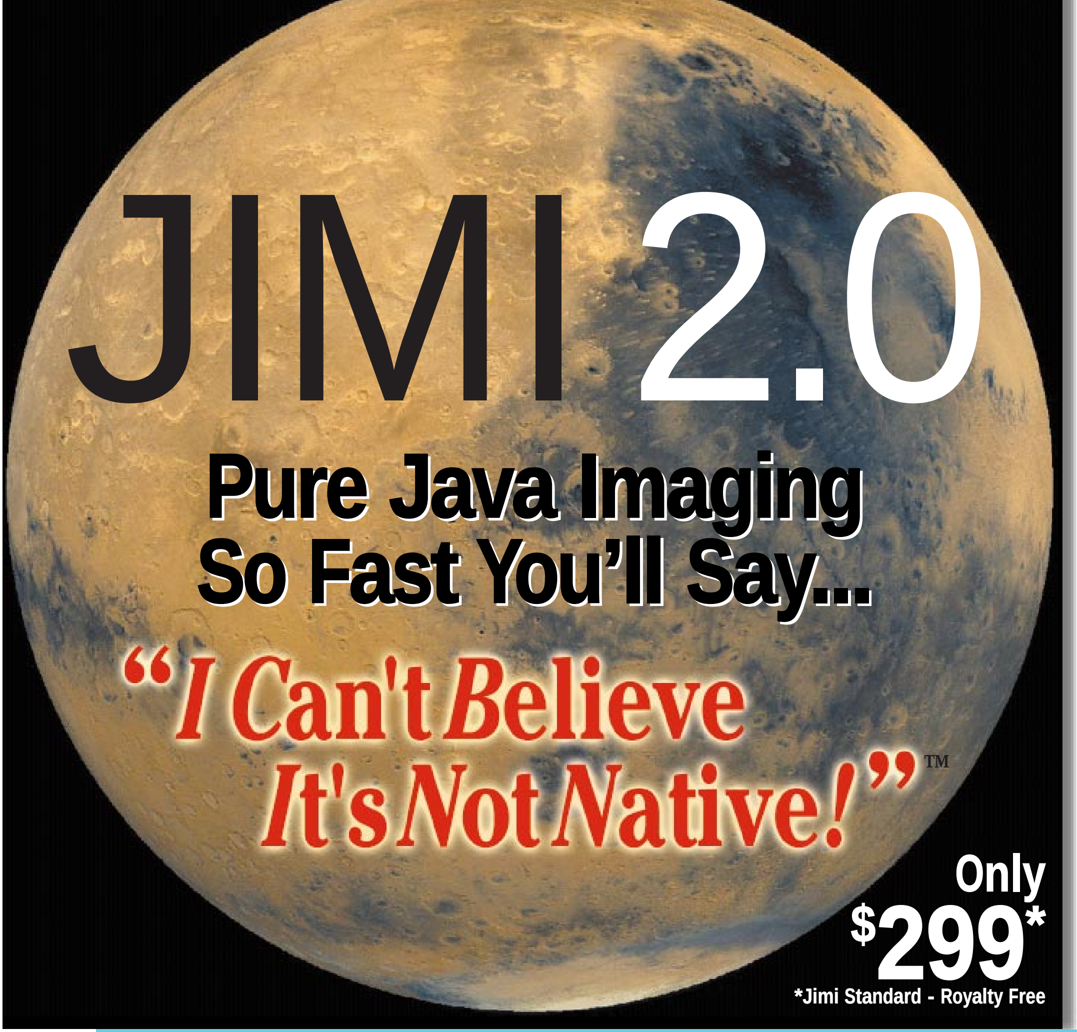
The CascadingJFrame class is a subclass of JFrame that contains a static PointCascade. After the JFrame has been created, the constructor gets the PointCascade's location, increments the PointCascade to set up for the next invocation and sets an autoposition flag to true. The class also overrides both of the JFrame's setBounds() methods. If the autoposition flag is set, the generated location is used in the call to the JFrame's setBounds() method to do the actual positioning. The PointCascade is static since it has to be shared between all the instances of CascadingJFrame. Each CascadingJFrame stores its generated location, but the PointCascade that generates these locations is a common resource. If any of the PointCascade's values are changed after locations have been generated, only the subsequently generated CascadingJFrame locations will be affected. The CascadingJFrame class has static methods to set and get its underlying PointCascade.

The cascading mechanism can also be negated in certain cases if desired. If the application determines that a particular CascadingJFrame is somehow special, then the initial positioning can either be adjusted by calling the CascadingJFrame's setLocation() method with the special location as a Point, turning off the autopositioning altogether, calling the CascadingJFrame's setAutoposition() method with a false value or using an extended constructor.

With a good understanding of the cascading mechanism, complete control of autopositioning is possible. It's as simple as that.

Multiple Cascades

But we can always make it more robust! It isn't enough to simply cascade. I immediately found that I wanted to have different types of windows cascading in different ways. For instance, in one area of the screen I wanted a stack of small windows, and in another a staggered set of larger



JIMI 2.0

Pure Java Imaging So Fast You'll Say...

"I Can't Believe It's Not Native!"™

Only $299*

***Jimi Standard - Royalty Free**

ACTIVATED INTELLIGENCE

Watch **Jimi Pro** open this **100-million pixel image** of Mars in just a **10 meg Java runtime heap.** You won't **believe** it's not **native. Jimi** delivers **power, speed,** and **simplicity.** Read and write tons of image formats including **TIFF, BMP, PNG, GIF, PSD, TARGA, JPEG, PICT** and more

Try It Now FREE at

WWW.ACTIVATED.COM

or call

212-896-8220

Are You a Java Start-up with a Shoestring Advertising Budget?

-or-

Are You a Software Giant with a Shoestring Advertising Budget?

Either way, your success depends on who your strategic business partners are. For your partners and the Java industry, your product looks as good as your display ad in *Java Developer's Journal!*

Carmen Gonzalez
Vice President,
JDJ Advertising Sales
carmen@sys-con.com

INTERVIEW

A SUCCESS STORY

Harold Ross & Roy Crippen of PowerCerv

"We had a very successful initial public offering. The total capital raised is close to fifty-three million dollars."

We know how to create success stories!

PowerCerv Corporation
(Nasdaq: PCRV)
On their way to a successful IPO, this two-year consecutive SYS-CON advertising partner did not miss advertising in one single issue!

ones. Applications all have their own idiosyncrasies; my goal was simply to cover most of the regular ones with the simple CascadingJFrame objects. The solution I implemented was to make the CascadingJFrame class contain multiple static PointCascade objects that could be arbitrarily named by the application. When a CascadingJFrame object is constructed, the application can specify the particular PointCascade to use for autopositioning.

To avoid adding complexity for the developer, the simple, unnamed-PointCascade case should continue to work; the calls to CascadingJFrame without a named PointCascade need to remain available. After all, plenty of applications do not require rigorous control or multiple streams of autopositioning. For most developers, using a single PointCascade would suffice, with only a few odd JFrames positioned directly. For these situations a default PointCascade is managed behind the scenes for the unnamed-PointCascade calls.

To implement the multiple PointCascade objects, the CascadingJFrame is made to contain a static hashtable of named PointCascades instead of just a single static PointCascade. The calls to CascadingJFrame without names are connected to the named calls using the default PointCascade's internal name. This way, when an unnamed PointCascade call comes in, the default PointCascade object is retrieved from the hashtable, used to get the location and incremented. If a named call comes in, the particular PointCascade is looked up, used and incremented. If the specified PointCascade is not in the hashtable, a new one is created and used. In this way the application maintains complete control over the autopositioning namespace. The source code for CascadingJFrame is in Listing 3.

Summary

The overall Design of the CascadingJFrame is shown in Figure 3. The important thing to notice is that the CascadingJFrame class contains the hashtable of PointCascade objects, not a CascadingJFrame instance. The instance simply contains the location that was retrieved from a specific PointCascade and an autoposition flag indicating whether or not the retrieved location should be used. The hashtable is a static – effectively shared by all the CascadingJFrame instances.

Autopositioning frees the developer's head a little, which is good – one less thing to think about. By providing the popular window-cascading behavior, in multiple streams when desired, sophisticated window positioning may be achieved with relatively little effort – by simply setting a few parameters and inheriting from the CascadingJFrame class. It's also interesting to consider extending additional subclasses that offer alternate positioning policies, or perhaps "position-memory" in which the boundaries of particular windows are written to persistent storage whenever they are moved or resized, and recovered when they are reconstituted by the application in a subsequent session.

Finally, a philosophical note: the best infrastructures are invisible and work automatically. The CascadingJFrame class envelops regular initial window positioning, typically to the degree of letting the applications programmer forget about it. And that is the best an infrastructure developer can hope for...making it so easy for others to do their work that they don't have to concern themselves with the underlying framework. If that goal is achieved, the framework was done right.

About the Author

David Anderson is a consulting software engineer at LEXIS-NEXIS, an online information service in Dayton, Ohio. At night he puts on his Java shoes and cranks out reams of code as a contractor. He has been developing software since 1974. Dave can be reached at monkey@one.net or through the monkey house (http://w3.one.net/~monkey).

monkey@one.net

Specialized Software

www.specializedsoftware.com

Web 99

www.mfweb.com

SD

www.sde

99

expo.com

KL Group Appoints New President

(Toronto, Ont.) – KL Group has appointed David Black as its new president. Former president and cofounder Greg Kiessling will retain his position as chairman.

Prior to coming to KL Group, Black served as president of Sun Microsystems in Canada from 1984 to 1989, and was instrumental in earning Sun's designation as one of the "Top 100 Companies to Work for in Canada" by the *Canadian Financial Post.*

For more information visit www.klgroup.com.

Borland Names Dale Fuller Interim President and CEO

(Scotts Valley, CA) – The board of directors of Borland has appointed Dale Fuller as interim president, chief executive officer and director. A broadly experienced technology executive, Fuller, 40, most recently served as president and CEO of WhoWhere? Inc., one of the world's leading Internet sites, before it was acquired by Lycos Inc. last year.

Fuller succeeds Delbert W. Yocam, the former chairman and chief executive officer, who resigned on March 31, 1999.

For more information visit www.borland.com.

Borland Announces JBuilder 3

(Scotts Valley, CA) – Borland, the software development tools division of Inprise Corporation, has announced Borland JBuilder 3, a major new version of its award-winning family of visual development tools for creating platform-independent Java business and database applications. JBuilder 3 provides comprehensive support for the Java 2 platform and allows individual and corporate developers to create platform-independent business and database applications, distributed enterprise applications and JavaBean components even more easily. JBuilder 3 is planned to be available on multiple platforms: first, on Microsoft Windows, then on Solaris and Linux.

Insignia and Next Nets Announce Selection of Jeode Platform

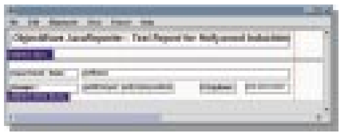
(Freemont, CA) -- Insignia Solutions and Next Nets Corporation of Japan have jointly announced that Next Nets will use Insignia's Jeode platform to implement its Windows CE-based Cessna mobile terminal system with Java-compatible technology. The Cessna system is designed for retail, logistics and distribution applications. The Jeode platform is Insignia's implementation of Sun Microsystems' Java technology for embedded systems. Cessna was released in January and runs Microsoft Windows applications with Windows CE at the same speed as most Windows NT servers.

For more information visit www.insignia.com.

ObjectWave Expands Java-Based Tool Offerings

(Chicago, IL) - ObjectWave Corporation has announced ObjectWave JavaReporter, a productivity tool written in Java that simplifies the process of creating reports. JavaReporter consists of an advanced toolset that permits programmers to embed documentation development capabilities for users within a Java application.

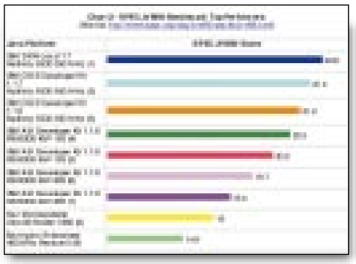
For more information visit www.objectwave.com.



Sales Vision Announces Support for 3Com's Palm Computing Devices

(Charlotte, NC) – Sales Vision, Inc., has released a conduit that synchronizes CRM information between Jsales, Sales Vision's flagship product, and 3Com's Palm Computing connected organizers. Sales Vision's Palm Computing conduit enables Jsales users to automatically coordinate contacts, calendar/activities and to-do items on their laptops as well as on their Palm Computing devices.

For more information you can visit their Web site at www.salesvision.com.



IBM Releases the Industry's Fastest JVM for Windows

(Somers, NY) – IBM has announced the industry's fastest Java Virtual Machine for the Microsoft Windows operating system. According to widely used performance benchmarks, IBM's JVM for Windows outperforms competitors' most recent offerings by an average of 30%.

The JVM is available at no charge for Windows 95, 98 and NT operating systems. Developers can immediately download IBM's offering of the Java Virtual Machine for the Microsoft platform from www.ibm.com/java/.

Blue Lobster Announces Linux Version of Web-to-Legacy Connectivity Solution

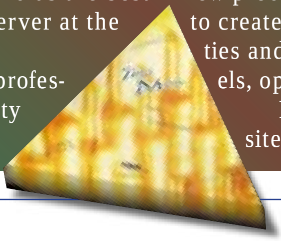
(San Jose, CA) – Blue Lobster Software has released its Stingray Software Development Kit (SDK) with support for Red Hat Software's version of the Linux operating system. Now Stingray will provide Linux developers with the ability to Internet-enable 3270 and 5250 legacy applications and automate the process of building reengineered interfaces or server-side applications for 5250 applications.

Cybelius TouchMore! 2.0 – Best of Show Award

(Los Angeles, CA) – Cybelius TouchMore! 2.0 received the "Best of Show" award as the best Web development application server at the Spring Internet World '99.

Cybelius TouchMore! 2.0 is a professional tool for adding interactivity and functionality to 3D VRML product models in Internet applications. The new product offers benefits that allow users to create real, physical, product-like properties and interactivity in their product models, optimized into small file sizes.

For more information visit their Web site at www.cybelius.com.



Employment Ad

Employment Ad

Employment Ad

Employment Ad

Employment Ad

THE GRIND

by Rick Ross

'Make the Iron Hot by Striking it'

—*Oliver Cromwell*

Everyone wins when the platform is supported by thousands of innovative developers and the solutions they create

What I want most for Java developers is opportunity! I don't mean just the opportunity for a steady job in the corporate world that any competent Java developer should enjoy with confidence. Rather, I mean the awe-inspiring opportunities that come from the sense that Java developers can potentially change the world of technology, and quite possibly the world as a whole.

In truth, it just wasn't that long ago that two guys named Steve started Apple Computer in a California garage and helped spark a furious explosion of innovation and economic growth. In fact, it wasn't even so long before that that Bill Hewlett and Dave Packard started their company in a different California garage! (*Note to Myself*: Perhaps I should consider working in a California garage.) Energy, enthusiasm, skill and dedication are undoubtedly required ingredients for such successes, but my countless conversations with Java developers have proved that there is no shortage of these within this fantastic and diverse community.

Java and the Internet have sparked an explosion of their own, in a different time and place in history, but no less significant and no less full of awesome potential. Before Java, the conventional wisdom was that our industry had matured and consolidated too much for the small developer to be able to make a difference. The dream that hard work and perseverance might result in the creation of the next Apple or Hewlett-Packard had given way to a different dream. For many, it went something like this, "I hope that when the Borg cube starts moving into the market sector I work in, they will assimilate my company rather than crush it." It was bleak by comparison – but consistent with apparent industry trends. Fortunately, it seems that hope for a better future is alive and well, and Java has played a big part.

This Java explosion should present a myriad of opportunities for developers, and the impending boom in smart devices and information appliances should amplify it dramatically. For some reason, however, not enough of these opportunities have emerged, and the fullness of Java's platform-neutral potential seems to be sidetracked. In fact, the theme of the year from corporate Java spin-doctors seems to be: "We all know Java isn't too great for client-side computing, but the real future of Java is on the server-side anyway."

Yeah, right! It's interesting that the people pushing this amended version of the Java value proposition sell – you guessed it – servers! It's no surprise that they place less emphasis on the power of client-side Java computing than they do on server-side, but I feel it is a wholesale abandonment of the original Java vision and of some of the greatest Java opportunities. I want those opportunities to be kept alive and to help Java developers everywhere thrive while pursuing them with all their prodigious talent and energy.

What can we do, as individual developers and as a community, to make this happen?

First, we can think for ourselves and keep the Java vision alive, despite what the corporate publicity and marketing machines are pushing. This is a simple and potent step, but an essential one. Who cares that Sun may be floundering without focus and that Microsoft quietly continues its anti-Java attack? The reality of the situation is that we are far less dependent on these big companies than they are on us. The corporate functionaries who spread the conventional wisdom that Java doesn't have what it takes for client-side applications couldn't compile "hello world" if their lives depended on it. If we keep our eye on the ball and move in the right direction, then the corporate interests will eventually have no choice but to be responsive. They are reliably opportunistic, but slow to perceive opportunity.

Second, we can develop a whole lot more top-quality client-side Java software. The new IBM implementation of Java for Win32 is a blazing speed demon, so use it to build something great. Knock the pundits and naysayers out with Java applications whose performance, reliability and value plainly prove them wrong. Take them by surprise and blow them away by using Java to deliver the distributed network applications they've all been predicting for so long. Client-side Java starts to matter when you write client-side Java applications that matter. It's not only possible to build great client-side Java solutions – the opportunity to do it is here at our fingertips. Don't wait, just do it!

Finally, we can collectively make it clear to the platform providers and their major corporate partners that we insist on having a piece of the action, too. It's obvious that it should be this way and that everyone wins when the platform is supported by thousands of innovative developers and the solutions they create. We need to let manufacturers know we want full and immediate access to Java in the tools and devices that will form the foundation of the next generation of information technology. Sony, Motorola, TCI, ATT, Sun – are you listening?

I want a device the size of a Palm Pilot that will run for six months on a single set of batteries, with a brilliant color display and perfect audio, that offers high-speed wireless connectivity anywhere – to my family, friends and colleagues, the Web, and all my personal data. I want this device to help me achieve better productivity in my work and enjoy greater satisfaction from my play – and I want it to be powered by Java! What are you waiting for? Please get out there and build it for me! I'll be forever grateful that you kept the vision of Java's real opportunities alive!

javalobby

Rick Ross is president and founder of the Java Lobby (www.javalobby.org), which currently has more than 34,000 members. He is also president of Activated Intelligence (www.activated.com) and can be

rick@activated.com

Object Space

www.objectspace.com

KL Group

www.klgroup.com