INFOBUS & JAVA APPLICATIONS

U.S. $4.95 (Canada $6.95)

JAVA™ DEVELOPER'S JOURNAL

JavaDevelopersJournal.com

Volume: 3 Issue: 2

SYS-CON PUBLICATIONS

Stingray
Full Page Ad

Protoview
Full Page Ad

Suntest
Full Page Ad

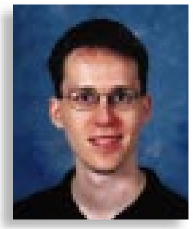
EDITORIAL OFFICES

SYS-CON Publications, Inc.
39 E. Central Ave., Pearl River, NY 10965
Telephone: 914 735-1900 Fax: 914 735-3922
Subscribe@SYS-CON.com

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

GUEST EDITORIAL

Ethan Henry



Roasting JavaBeans - Java's Component Architecture

In 1997 there was an explosion of third-party tools for Java. A variety of integrated development environments (IDEs), class libraries and visual components became available. Web sites that review and catalog Java tools like Gamelan (http://www.developer.com/directories/pages/dir.java.html) and JARS (http://www.jars.com) saw their listings swell. So, what was the key factor that led to this growth? JavaBeans™.

JavaBeans is the Java component architecture standard. It allows developers to create components and expose their capabilities in a consistent, standardized manner. JavaBeans is in many ways comparable to other component architectures, like Microsoft's COM/OLE/ActiveX for Windows. Unlike OLE, however, it is specific to one language, Java, and not to any one operating system.

The JavaBeans specification was announced by Sun at the first JavaOne conference in May of 1996. The announcement came along with announcements for a number of other APIs – the Media APIs (Java 2D, Java 3D), RMI, Security, Commerce – but JavaBeans was the focus of most of JavaSoft's initial development effort. The JavaBeans specification was completed ahead of schedule (October '96) and was released to the world as part of the Java 1.1 core API in February 1997. From the amount of effort JavaSoft put into it, it was obvious that they felt that a robust, well-designed component architecture was key to Java's future success.

The inclusion of JavaBeans as a core Java API has opened the door to the third-party components market. All of the major Java IDE vendors have made Beans support an integral part of their strategies. Beans gives the IDEs the ability to deal with components, both visual and non-visual, in a standard way based on core Java APIs. Any one of the IDEs could have come up with a specification like JavaBeans by themselves, but the fact that Beans is a standard, core API makes it more attractive than a proprietary, single-vendor solution. With the availability of an open component standard that was supported in multiple IDEs, component vendors rushed to adopt Beans. Although none of the IDEs support every JavaBeans feature, the level of support is amazing for a language that's just approaching its third birthday. JavaBeans has been a big win for IDE and tool vendors but ultimately the biggest win is for developers who can feel free to choose their tools and development environments without worrying whether or not they'll all work together.

While JavaBeans has been great so far, it isn't finished by any means. The 'Glasgow' specification maps out the immediate future for Beans with three much needed additions to the specification: the Containment and Services protocol, the drag and drop API and the JavaBeans Activation Framework. These new pieces of functionality, most of which will be provided in the upcoming Java 1.2 release, will make Java an even more compelling platform for application developers.

The ability to do integrated drag and drop operations with the desktop OS, the ability for Beans to interact better with other tools at design time (via the Containment and Services protocol) and the ability for Java programs to exchange self-describing data objects will attract more and more developers, perhaps even pulling some of them away from more established RAD software building tools.

There's still more that needs to be done after 'Glasgow' though. Part of the original specification was an object aggregation and delegation model that would allow Bean developers to construct Beans out of other Beans with a standard way to expose the internal Beans to the outside world. This would allow Bean vendors to create Beans that are more powerful, yet simpler to use and understand. Another major change that should be made is that in order to support this sort of functionality, IDEs are going to have to move away from their current methods of handling JavaBeans through code generation alone and add a serialization-based mechanism in order to support a number of the JavaBean features that they're currently missing, like support for Bean customizers. A number of JavaBean vendors have been pushing for this change, but the IDEs haven't shown any signs of moving yet.

Overall, JavaBeans has been the most important new API to come along for Java since the initial Java 1.0 release. In the three years since its release, Java has gone from being an interesting toy language to a serious, industrial-strength development platform largely because of the capabilities offered by JavaBeans. If you're not using JavaBeans in your Java development, ask yourself – why not?

About the Author

Ethan Henry is the Java Evangelist at KL Group. He spreads the good news about Java and KL Group's Java components and tools. Before joining KL Group, Ethan was a Java instructor. Ethan can be reached at egh@klg.com

Zero G
Full Page Ad

SYS-CON PUBLICATIONS

PHONE, ADDRESS & WEB DIRECTORY

CALL FOR SUBSCRIPTIONS
1 800 513-7111

International Subscriptions & Customer Service Inquiries
914 735-1900
or by fax: 914 735-3922

E-Mail: Subscribe@SYS-CON.com
http://www.SYS-CON.com

MAIL All Subscription Orders or Customer Service Inquiries to

JAVA DEVELOPER'S JOURNAL
Java Developer's Journal
http://www.JavaDevelopersJournal.com

VRML DEVELOPER'S JOURNAL
VRML Developer's Journal
VRMLJournal.com

National JAVA LEARNING CENTER
National Java Learning Center, Inc.

JAVA DEVELOPER'S JOURNAL 1998 JAVA Buyer's Guide
JDJ Buyer's Guide
JavaBuyersGuide.com

WEB-PRO DEVELOPER'S SUPPLEMENT
Web-Pro Developer's Supplement

SYS-CON Publications, Inc.
39 E. Central Ave.
Pearl River, NY 10965 – USA

EDITORIAL OFFICES
Phone: 914 735-1900
Fax: 914 735-3922

ADVERTISING & SALES OFFICE
Phone: 914 735-0300
Fax: 914 735-7302

CUSTOMER SERVICE
Phone: 914 735-1900
Fax: 914 735-3922

DESIGN & PRODUCTION
Phone: 914 735-7300
Fax: 914 735-6547

DISTRIBUTED in the USA by
International Periodical Distributors
674 Via De La Valle, Suite: 204
Solana Beach, CA 92075
Phone: 619 481-5928

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

FROM THE INDUSTRY

Eric Shapiro

Creating Java Tools to Fulfill the "Write Once, Run Anywhere" Promise

Imagine Henry Ford developing the first widely available automobile. He was a pioneer, engaged in the most exciting new industry of the time.

Imagine how frustrated he must have been. Where would drivers buy gas? Were the wrenches and screwdrivers advanced enough to build the cars? Would the cars hold up on the variety of roads out there? How would he ship the cars? And… would anybody buy them?

Nearly 100 years later, Java developers are pioneers all over again. It is an exciting time, but there are new obstacles to deal with. Instead of wrenches and screwdrivers, we use compilers and debuggers. Instead of gas, we use the AWT and JFC. And instead of roads, we run our programs on a variety of Java VMs. And we're still wondering what the right market is for our new software.

For Java developers, all phases of product development are complex. While native developers only need to cope with making their code work with mature compilers and APIs, Java developers must tame nascent technologies at every stage of development. When our software doesn't work, we can't be certain if we've coded something wrong, the compiler generated bad output, the APIs have bugs or the Java VM has an incompatible implementation.

Take Java VMs as an example. VM vendors are juggling features and compatibility, to the growing frustration of developers. Every VM does things differently, whether it's the implementation of the Abstract Windowing Toolkit or security management. These inconsistencies make it difficult to achieve the "write once, run anywhere" promise.

Current compilers and integrated development environments add to the frustration. The latest IDEs often lack many features of their native counterparts and fail to integrate the newest Java technologies. How many IDEs implement JavaBeans introspection, presenting a "customizer" if available, and linking events to listeners? And how many have integrated debuggers that can handle complicated, multi-threaded code? Not many.

But the situation is improving. Sun's recently announced Java Porting Centers hopefully will establish a common codebase for Java VMs and improve compatibility. It's still not enough though, and we should insist that all Java VMs be 100% compatible with each other.

The Java APIs are improving as well. While the AWT was quirky and needed adjustment for each platform's specific graphical user interface, the Java Foundation Classes eliminate most inconsistencies – a giant step in the right direction.

But Java tools still need improvement. Java developers shouldn't have to work with compilers that crash or IDEs that only partially support the Java APIs. Compilers must work perfectly every time. I can't tell you how many times I've seen people resort to the command line JDK tools since their "paid for" tools crash or are incompatible. It just proves the point that compatibility is far more important than features.

As Java developers, we need to help send this message to the Java VM vendors, IDE vendors and everyone else working with Java. We need to insist on core functionality with robust compatibility and completeness – before we ask for cool new features.

It's a whole new industry – we should treat it that way. Just imagine if the first Fords had saddles for seats and ropes instead of parking brakes… we'd probably still be wearing spurs in our fancy new sports cars.

About the Author

Eric Shapiro is co-founder and Director of Zero G Software, Inc., which produces several specialized installer tools for the Java enterprise and developer markets and provides Java installation-related consulting services.

Take a Ride on the InfoBus

The InfoBus is expected to become a standard framework for the future of Java enterprise applications

by **Ajit Sagar and John Sigler**

Components transcend the programming language and support a very high degree of reuse. They greatly simplify the construction of large and complicated software architectures. One of the main reasons why Java promises such a bright future for the computing world is because of its inherent support for component architectures. Some examples of Java's component support are JavaBeans™, Java Foundation Classes (JFC), JavaBeans Activation Framework (JAF) and the InfoBus.

This article introduces the InfoBus, a specification for interconnecting JavaBeans by defining the interfaces and the protocol for their interaction. First, the need for a framework like the InfoBus for current Java architectures is identified. Then the InfoBus, its components and the InfoBus API are briefly described. This is followed by an example that illustrates how to build an application using the basic elements of the InfoBus. The article concludes with a discussion on its current status and future issues.

We assume that you are familiar with Java component architecture concepts in general and with the JavaBeans paradigm in particular. Sources for additional information are listed where required.

JavaBeans and the Infobus

JavaBeans, Java's implementation of the component model, is built mostly using features of the language itself. The JavaBeans model allows developers to construct applications by connecting components programmatically and/or visually. The model consists of an architecture and an API. These combine to provide a framework for writing components. Programmatically, JavaBeans simply enforces a set of design patterns on top of Java's existing event model.

The InfoBus architecture and API were developed by Lotus Development Corporation, Inc. and Sun Microsystems, Inc. to define standards by which a wide range of Java components acting as data producers and data consumers can communicate. The InfoBus facilitates the creation of applications built from JavaBeans that exchange data asynchronously within the same Java Virtual Machine. Hence, the Infobus defines JavaBeans component interaction within a single process.

So Why InfoBus?

JavaBeans uses introspection to discover and learn about other Beans at runtime. This is based on certain design patterns in the names of methods used for interacting Beans. Communication between Beans is achieved via the AWT's event-response mechanism.

The InfoBus is meant for a more specific kind of JavaBeans interaction and its design adds the following constraints to generic JavaBeans interaction:

JavaBeans that are loaded by the same Java class loader can "see" other Beans and can make direct calls between them. However, this involves examining the interface of the other Bean using introspection and, thus, has substantial overhead. In contrast, the InfoBus interfaces form a tightly coupled contract between cooperating Beans. Procedure calls are direct and no inferring is required.

JavaBeans use the standard event/response model, where the semantics of the interaction depend upon understanding Bean-specific events and then responding to these events with Bean-specific callbacks to the event raiser. The InfoBus interfaces, on the other hand, have very few events and have an invariant set of method calls for all components. The semantics of the data flow are based on the contents of data that flow across the InfoBus interfaces, not in the names or parameters from events, nor in the names of parameters or blocks.

The InfoBus promotes the notion of data aware components and supports semantics that allow data to be communicated in a canonical format that involves both the encoding of data and the navigation of the data structure.

InfoBus in a Nutshell

The InfoBus applications can contain three types of JavaBeans:

• Data Producers
• Data Consumers
• Data Controllers

As the names suggest, the Data Producers produce data and the Data Consumers consume data. The exchange of data takes place using a data interchange protocol called the InfoBus. In an InfoBus application, a component may act as a Data Producer, Data Consumer or both. Data between InfoBus components flows in the form of named objects called DataItems. Data Controllers are specialized components that control the rendezvous between producers and consumers.

The InfoBus protocol defines the following steps for data exchange:

1. The components connect to the InfoBus. This is called "joining" the Infobus. After a component has joined the InfoBus, it listens for bus notifications which are generated in the form of known events.
2. Data Producer components announce the availability of new data.
3. Data Consumer components register to listen for named DataItems. Upon retrieving a DataItem that has been published, the Data Consumers can retrieve the encoding of the data values in the form of a String or a Java object.
4. Consumers can attempt to change the value of DataItems. Producers enforce a policy on whether anyone can change data. The data change notifications take place using named events.
5. The components can disconnect from the InfoBus. This is called "leaving" the InfoBus.

InfoBus Events

The InfoBus specification 0.04a, which has been used for the example application in this article, defines one event called

DataItemChangedEvent. The InfoBus components, i.e., Data Producers and Consumers, access the data by calling methods on the InfoBus. These methods are listed in Table 1.

The InfoBus API

This section provides a brief introduction to the main interfaces of the InfoBus API. For specific API details, please refer to the JavaBeans Web site at http://java.sun.com/beans/infobus/Package-javax.infobus.html

The InfoBus API has two major interface groups. The first is concerned with the rendezvous or brokering mechanism for announcing and locating data items within an InfoBus. The major rendezvous classes are shown in Table 2.

The other group of interfaces includes data items that are exchanged on the bus and their associated classes. These have changed significantly from the earlier releases. The latest versions are listed in Table 3.

An InfoBus Example

In this section we will lead the reader through an example application that illustrates the use of the main entities defined in the Infobus and the interaction between them. The example uses two applets: One is a producer of temperature data items and the other is a consumer of these same data items. Both are displayed on a single Web page and they communicate across a single InfoBus. Figure 2 illustrates the building blocks for the temperature gauge. The source is provided in Listings 1 and 2.

The producer applet simulates a temperature data source. Initially, it announces to the InfoBus the availability of a "temperature" data item. The data item then sends out temperature change notifications to any listeners.

The other applet is the data consumer. It is used as a simple control center to display the current temperature along with a color-coded background that indicates the state of the equipment. Green indicates normal temperature, yellow is a warning and red indicates overheating.

The producer applet is implemented by the TemperatureSource class in Listing 1. It implements an InfoBusDataProducer interface which allows it to announce data items on an InfoBus. The applet requires access to both an InfoBusMember and the data item to be produced, so it also needs to implement the InfoBusMember interface and a data item interface. Finally, it also implements Runnable in order to update the temperature value in a background thread.

When the applet's init() method is called, the TemperatureSource joins a new InfoBus, adds itself as a listener for any InfoBus property changes and then adds a Label to display the temperature. In the producer applet, the memberSupport_ data member handles this as follows:

```
memberSupport_.joinInfoBus(this);
memberSupport_.addInfoBusListener(this);
```

The InfoBusMember interface is used primarily by TemperatureSource to get a handle to the current InfoBus. The implementation is delegated to the InfoBusMemberImpl class delivered as part of the InfoBus package. As you go through the listing you will notice that there are several other methods that InfoBusMembers delegate to the memberSupport_ variable.

Next, in the applet's start() method, the TemperatureSource adds itself as a producer on the bus that it just joined. This allows it to send notifications to listeners when a data item becomes available.

```
ib.addDataProducer(this);
```

At this point, we have a data producer that is ready to announce data items. The run() method of the thread that is spawned

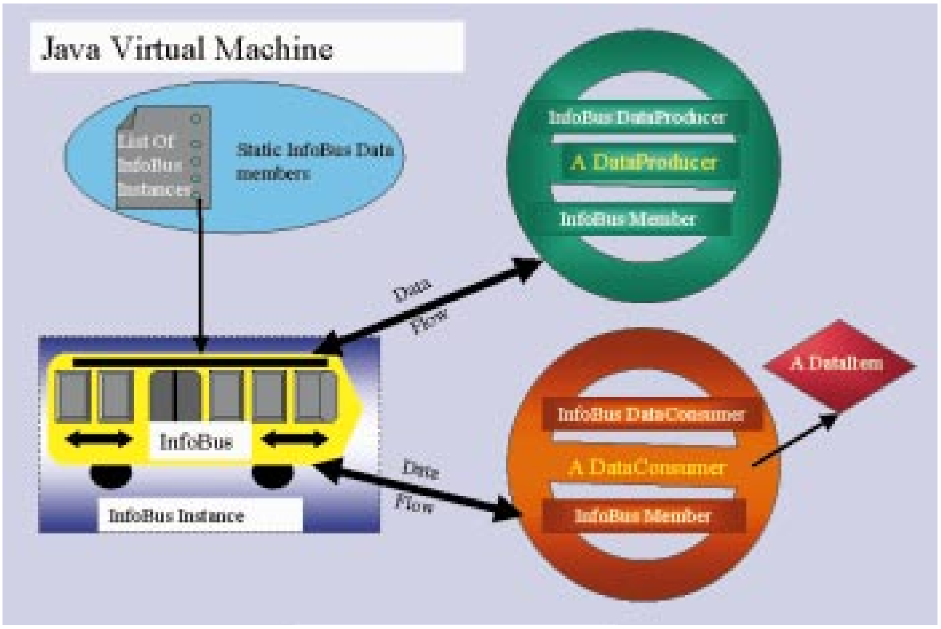


Figure 1: How these components play together in the InfoBus

Event Name	Description
InfoBusItemAvailableEvent	Broadcast on behalf of a producer to let consumers know about the availability of a new DataItem.
InfoBusItemRevokedEvent	Broadcast on behalf of a producer to let consumers know that a previously available DataItem is no longer available.
InfoBusItemRequestedEvent	Broadcast on behalf of a consumer to let producers know about the need for a particular DataItem.

Table 1: InfoBus Events

Interface Name	Description
InfoBus	The heart of the InfoBus technology. It maintains a list of existing InfoBus instances, bus members, producers, consumers and enables communication among them for announcing and locating data items.
InfoBusDataProducer	Used to indicate that an object provides data on the InfoBus. Producers announce the availability of new data items
InfoBusDataConsumer	Implemented by objects that are seeking data from an InfoBus. Often these are visual components that will display the data item but they can also act as filters where they modify the data and then forward it on to other consumers.
InfoBusMember	Members are able to join and leave an InfoBus, as well as change their current bus

Table 2: InfoBus Rendezvous Interface

Rogue wave
Full Page Ad

DataItem Interface Name	Description
DataItem	The base interface for data items. This is a very lightweight component and is the basic data unit for data interchange.
ImmediateAccess	An access wrapper class for simple data items which are not collections of other data items. Offers methods to set the data and extract the data as either a String or an Object.
ArrayAccess	Collections of data items organized in a bounded n-dimensional array.
RowSetAccess	A data item to handle basic database table access with support for modifying, inserting, deleting and cursoring through a table
DbAccess	Used for simple lifetime and transaction management of a Rowset Access data item.

Event-Handling Interface Name	Description
DataItemChangeListener	Implemented by data item recipients to handle data item change notifications.
DataItemChangedEvent	The base class of all data item events. Upcoming versions include specialized value changed, item added, item deleted and item revoked events.

Table 3: InfoBus Data Interface

in the applet's start() method is used to announce the temperature data item:

```
getInfoBus().fireItemAvailable("Tempera-
ture", this);
```

"Temperature" is the name of the data item while this (i.e., the TemperatureSource) is passed as the producer of the item. After this, the program enters a loop in which the temperature data item is modified every 2 seconds to simulate changes in the data source. This change occurs in the producer's setInternalValue() method. Modifying the data item causes an itemValueChanged() method to be called on all DataItem-ChangedListeners (i.e., the consumers).

As mentioned before, the producer implements the ImmediateAccess interface which is a DataItem. Therefore, both interfaces must be implemented. DataItems have methods to get the source of the item as well as methods for adding and removing data item listeners. The interface also includes a getTransferable() method which is used to return a Java Transferable object if other flavors of the data item are available. For our example, we will just return null.

ImmediateAccess data items are standalone (i.e., non-collection) data items. They offer access to the underlying data through getValueAsObject() and getValueAsString() methods which, in our example, will return the value of our underlying temperature value as a Double and String respectively. ImmediateAccess items also have a setValue() interface, but in this example we disallow consumers attempted temperature modifications by throwing an InfoBusAccessException inside setValue. We now have the basic outline of a data producer.

Our data consumer, the Temperature-Viewer, will receive notifications of "Temperature" data items and then display the value. Like the producer, the consumer delegates to its InfoBusMemberImpl instance member all InfoBus Member method implementations.. The applet listens for data item changes by implementing the DataItemChangedListener interface. So our consumer/component applet, TemperatureView, is defined to be:

```
 public class TemperatureViewer extends
Applet
     implements InfoBusMember, InfoBusData-
Consumer, DataItemChangedListener {
```

Like the producer, the consumer joins the default InfoBus in the applet's init() method and then adds an AWT Label to the applet for visually displaying the temperature data. In the start() method, we add the applet as a data consumer and then request a "Temperature" item. If one is found, we then update the temperature display.

Updates from the temperature data item are sent to the TemperatureViewer, a DataItemChangedListener. This interface consists of itemValueChanged, item-SizeChanged() and itemDeleted() methods. In our example, itemValueChanged() is the only item changed method of interest and is implemented as:

```
public void itemValueChanged (DataItem-
ChangedEvent e) {
     setDisplayValue(e.getDataItem());
}
```

The TemperatureViewer method setDisplayValue() is used to update our view of the data item. SetDisplayValue() extracts the data from the ImmediateAccess data item and then updates the display. It uses the ImmediateAccess getPresentation-String() method as the text for the label.

Program Environment

Figures 3 and 4 show screen captures of the example. To run the example, you will need JDK 1.1 or a later version, an InfoBus implementation and a 1.1-based browser. The example has been tested with the TP2 implementation (i.e., the 0.04a spec) of the InfoBus using JDK 1.1.5 on NT 4.0. It should work with any 1.1 compliant browser and has been successfully tested with the Netscape 4.04 (with the JDK 1.1 patch), Hot-Java 1.0 and the JDK 1.1 appletviewer.

Beginning with JDK 1.2, the InfoBus will become a standard Java extension. Until then, you will need to download an InfoBus implementation from JavaSoft to run the example locally. An implementation and the specifications are located at http://java.sun.com/beans/infobus

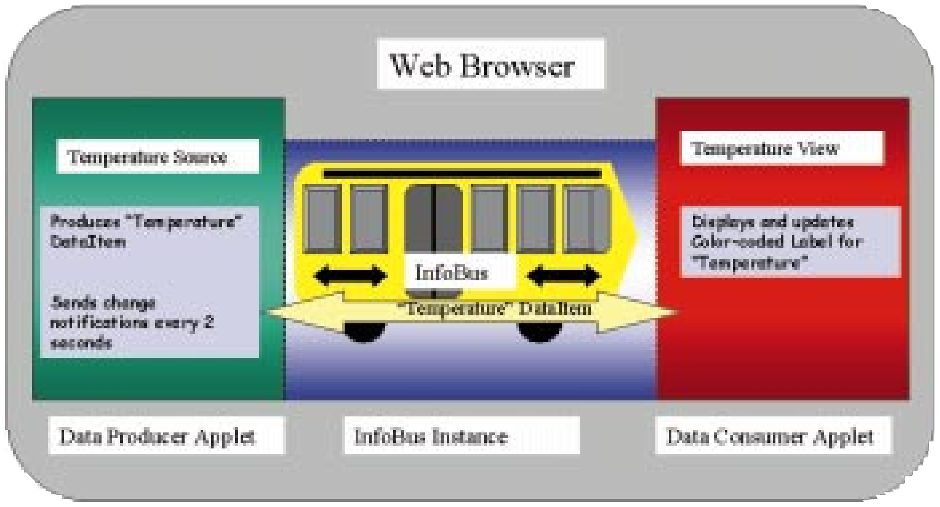


Figure 2: The building blocks for the temperature gauge

Thought Inc
Full Page Ad

Note that due to space restrictions, comments, exception handling and error handling have been removed. A more detailed explanation of the example and commented example source can be located either at the ***JDJ*** Web site or the authors' Web site: http://cambrian.netin.com/jdj-examples/

Current Status

The InfoBus specification is still evolving. At the time of this writing, the latest InfoBus implementation is TP2 which is based upon the 0.04a specification. The latest spec is 0.06 and the TP3 implementation of this spec should be out by the time you read this article. The 0.06 specification is expected to be quite close to the final 1.0 release.

The example here is based upon the 0.04a spec. The newer specs have made some very useful changes to data items which should make for a much more robust environment, so we recommend moving to the newer as soon as possible. The primary differences you'll need to adapt to are:

1. Change method signatures; primarily involves converting DataItem to Object in most interfaces
2. Use JDK 1.2 collections instead of CollectionAccess and keyed Access interfaces.
3. Implement DataItemChangeManager on DataItems where you want to allow listeners, and remove the add/removeListener() methods for the others
4. Check InfoBus and data item naming conventions to make sure they don't conflict with the newer versions' recommendations.
5. Consider whether other new features are useful, such as data controllers, security and new data items

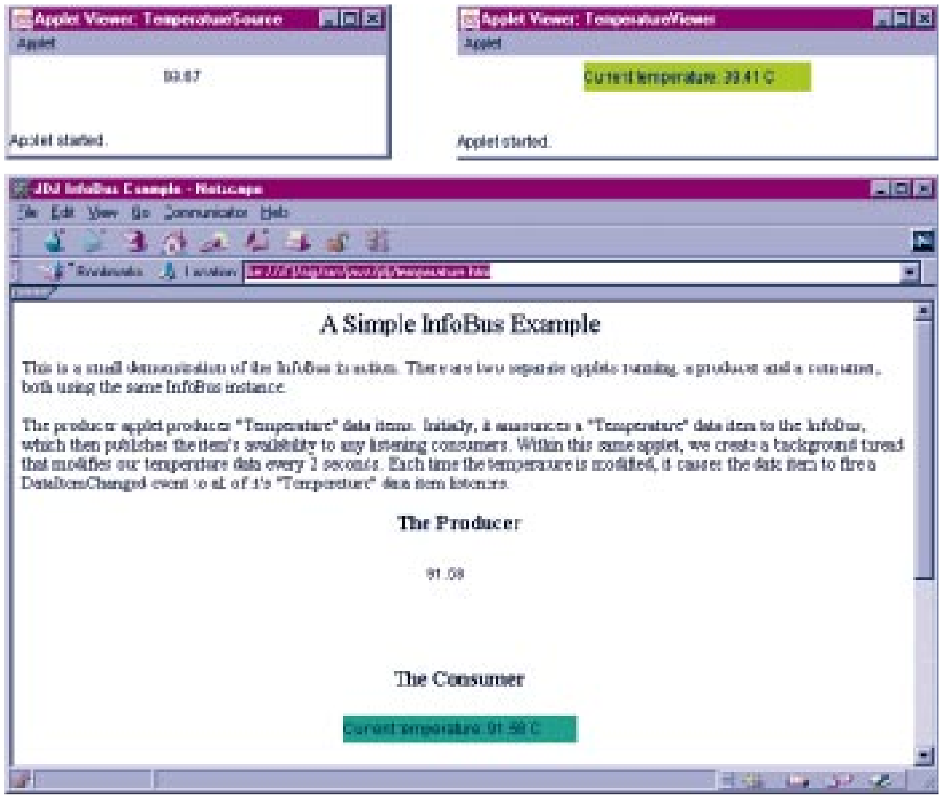
Related Issues

Lotus Development Corporation is using the InfoBus as the basic building block for a suite of desktop applications called eSuite. Sun Microsystems, Inc. is finalizing the specs for the JavaBeans Activation Framework (JAF), which adds data awareness to JavaBeans based components. It will be interesting to see how these technologies evolve and how they supplement each other. More information on these may be found at the following sites:

http://esuite.lotus.com/eSuite/seSuite-site.nsf
http://java.sum.com/beans/glasgow

Conclusion

In this article we've examined a new Java component model called the InfoBus which is based on JavaBeans. We looked at an example application using its main components. The InfoBus can be used to build UI application suites and data-aware architectures and is expected to become a standard framework for the future of the Java enterprise applications.



Applet Viewer: TemperatureSource
Applet
Applet Viewer: TemperatureViewer
Applet
JDJ InfoBus Example - Netscape
File Edit View Go Communicator Help
Bookmarks
Location

93.67

Applet started.

Current temperature: 39.41 C

Applet started.

A Simple InfoBus Example

This is a small demonstration of the InfoBus in action. There are two separate applets running, a producer and a consumer, both using the same InfoBus instance.

The producer applet produces "Temperature" data items. Initially, it announces a "Temperature" data item to the InfoBus, which then publishes the item's availability to any listening consumers. Within this same applet, we create a background thread that modifies our temperature data every 2 seconds. Each time the temperature is modified, it causes the data item to fire a DataItemChanged event to all of it's "Temperature" data item listeners.

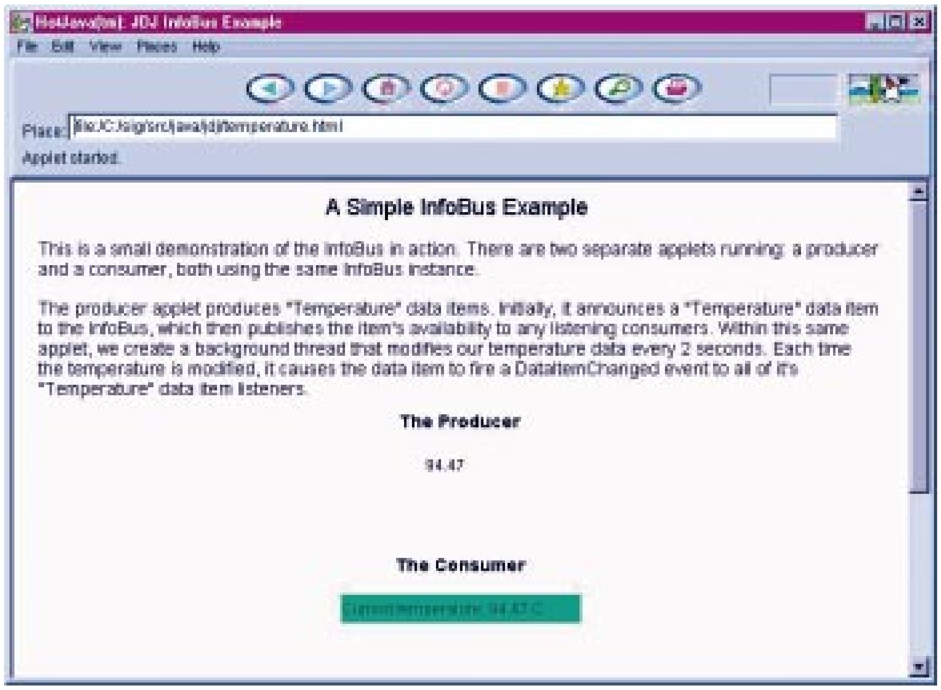
The Producer

91.58

The Consumer

Current temperature: 91.58 C

Figure 3



file:/C:/sig/src/java/jdj/temperature.html

Applet started.

A Simple InfoBus Example

This is a small demonstration of the InfoBus in action. There are two separate applets running: a producer and a consumer, both using the same InfoBus instance.

The producer applet produces "Temperature" data items. Initially, it announces a "Temperature" data item to the InfoBus, which then publishes the item's availability to any listening consumers. Within this same applet, we create a background thread that modifies our temperature data every 2 seconds. Each time the temperature is modified, it causes the data item to fire a DataItemChanged event to all of it's "Temperature" data item listeners.

The Producer

94.47

The Consumer

Current temperature: 94.47 C

Figure 4

About the Authors

Ajit Sagar is a member of the technical staff at i2 Technologies, in Dallas, Texas. Ajit has 7 years of programming experience in C, 3 years in C++ and 1-1/2 years in Java. His focus is on networking, system and application software development. To reach Ajit, call 214 860-6906 or e-mail him at Ajit_Sagar@i2.com

John Sigler is a member of the technical staff at i2 Technologies, Dallas, TX. He has a B.S. in Computer Science from Texas A&M University. John has 10 years of software development experience and 1-1/2 years in Java.His focus is on UI, system and application development. John can be reached at John_Sigler@i2.com

Ajit_Sagar@i2.com John_Sigler@i2.com

Install shield
Full pg

Developing 3-Tier Database Applications with Java Servlets

Building a Web interface to an existing corporate database

by **Chád Darby**

The drive to create a successful Web site has resulted in Web applications that are interactive and informative. A wealth of information is stored in corporate databases and there is a rush to publish this information on the Web. Corporations' traditional client/server applications are being edged out by Web-based applications. This occurrence is possible thanks to the universal client, the Web browser.

This article is the second in a three-part series on Java servlets. Last month (***JDJ,*** Vol. 3, Iss. 1), I gave you an overview of the Java servlet technology and how to migrate your existing CGI scripts to Java servlets. This article will not reintroduce those concepts. I assume that you are familiar with the basics of the Java Servlet API and the Java Database Connection API.

In this article, you will learn how to build a 3-tier database application that uses Java servlets and the Java Database Connection (JDBC). You will witness the construction of each tier and understand the techniques used to create Java Servlets with database connectivity.

The Challenge

A prominent public speaker keeps track of students who attend her Internet seminars. After each seminar, she exchanges business cards with the interested students. She then enters the student data into her database program.

Instead of entering the data from each business card, she envisions a Web application that would do the work for her. At each seminar, the Web application is set up on several Web terminals. Each student registers at a Web terminal and provides their name, company, e-mail address, course title and expectations. The Web application also has the option to display an updated list of all students.

3-Tier Solution

The Web application is made up of three tiers: Web browser, servlet middleware and database server. The three tiers are illustrated in Figure 1.

The first tier uses a Web browser to take advantage of the installed user base of this universal client. An HTML form is used for user-input and the results of the database query are returned as an HTML page. Using

HTML for user-input and displaying the data lowers the requirement of the client's browser version. This Web application does not impose the requirement of a Java-enabled browser with the latest JDK patch.

The second tier is implemented with a Web server running Java servlets. The Java servlet is able to access the database and return an HTML page listing the data. Please note that Java servlets are not restricted to Sun Microsystem's Java Web Server. You can also use Java servlets with the following servers: Microsoft IIS, Netscape FastTrack and Enterprise Server and O'Reilly WebSite Professional. Servlet functionality is possible with Live Software's JRun product, http://www.livesoftware.com. Sun's Java Server page, http://jserv.javasoft.com, also has a list of servlet-enabled Web servers.

The third tier is the back-end database server. The Java servlet can access information in the database provided that a JDBC driver exists. In our situation, the public speaker's database is MS-Access so we can use the JDBC-ODBC driver that is bundled with the Java Development Kit versions 1.1 and higher.

Application Interaction

As you can see, the application is partitioned into three different tiers. Figure 2 illustrates the interaction between the different tiers of the application.

Each step of the interaction is described below.

Step 1:

The user enters information into an HTML form. The form data is passed to the Java servlet running on the Web server.

Step 2:

The Java servlet parses the form data and constructs an SQL statement. The SQL statement is passed to the database server using the Java Database Connection (JDBC).

Step 3:

The database server executes the SQL statement and returns a result set to the Java servlet.

Step 4:

The Java servlet processes the result set and constructs an HTML page with the data. The HTML page is then returned to the user's Web browser.

Analyzing the Database Schema

Our public speaker is currently storing the student information in a MS Access database. The database contains one data table called Students. The data fields are defined in Table 1.

Designing the Web Browser Interface

The browser interface is composed of a main menu page. This page presents the user with the option of student registration or displaying the students in the database. HTML code for the main menu is provided in Listing 1.

Student Registration Form

The students register using an HTML form. The form collects name, e-mail address, company name and other course information. A snapshot of the student registration form is given in Figure 3.

Once the user enters their information then the "Register" button sends the data to the Java servlet.

Developing the Servlet Middleware

The servlet middleware encapsulates the business logic of the application. The servlet parses the form data and constructs an SQL statement. The SQL statement is then passed to the database server. After executing the SQL statement, the database server returns a result set back to the servlet. At this time, the servlet processes the result set and constructs an HTML page for the user.

The servlet being created is called StudentDBServlet. The StudentDBServlet has methods to perform the following functions: initialization, servicing requests, displaying students and registering a student. Let's look at each of these functions in detail.

Initializing the Servlet

In the life cycle of a servlet, the init() method is called the first time the servlet is invoked. Listing 3 is the code listing for the init() method.

For the StudentDBServlet, a database connection is opened and prepared statements are created for displaying a student list and registering a student. The database connection is left open for the lifetime of the servlet. Depending on your design, you can open and close a connection for each SQL query. However, in this application the database connection is opened only once.

Servicing User Requests

Whenever a servlet is invoked the service() method is called. The service() method is the main entry point for servlets. However, if this is the first time the servlet is being invoked, then the init() method is called followed by the service() method.

The service() method in this application is used to branch the request to the appropriate method. The student registration form has a hidden field called Register. The service method checks the value of the Register field. If the value is non-null then the registerStudent() method is called. If the field does not exist on the HTML page then a null value is returned. A null value results in the execution of the displayStudents() method.

Displaying the Student List

The displayStudents() method encapsulates the business logic to access the database and display the student list. This is accomplished by using a supporting Stu-

Field Name	Data Type	Length
ID Autonumber	-	
LastName	Text	50
FirstName	Text	50
Email	Text	50
Company	Text	50
CourseTitle	Text	50
CourseLocation	Text	50
CourseStartDate	Date/Time	-

Table 1: Database Schema

dent class. The code for the Student class is given in Listing 4.

The Student class has data members to hold information for one student. The Student class also has constructors that can create an object based on form data or a database result set. The code below demonstrates how a student's last name is accessed from the form data.

```
lastName = request.getParameter("LastName");
```

The *request* object is an instance of HttpServletRequest. The *request* object contains the form data. The form data is accessed by calling the getParameter() method and providing the name of the form field. The student registration form has a *LastName* field. Refer to my earlier article in ***JDJ*** for a detailed discussion of accessing form data with servlets.

The *Student* class has methods for accessing its data members and for providing a string representation of its data. Listing 4 contains the code listing for the methods of the Student class. The toString() method returns a normal string version of the data members. The toWebString() method returns the data as an HTML-formatted unordered list. The toTableString() method returns the data as an HTML-formatted table row. These methods are used to build the student list.

Constructing an HTML page creates the student list. In the displayStudents() method of Listing 3, the heading of the HTML page is created. Next the table heading is created to display the information as shown below.

Student Name	E-mail	Company	Course Expectations

The servlet sends a request to the database server to get a list of students. The following SQL statement was prepared in the init() method.

```
select * from Students order by LastName;
```

The SQL statement will return a list of students in alphabetical order based on the last name. The result set is used to create the body of the HTML table. A while-loop is created to iterate through each record of the result set. The code fragment for the while-loop is:

```
    int rowNumber = 1;
    while (dataResultSet.next())
    {
       aStudent = new Student(dataResult-
Set);
       tableBody += aStudent.toTa-
bleString(rowNumber);
       rowNumber++;
    }
```

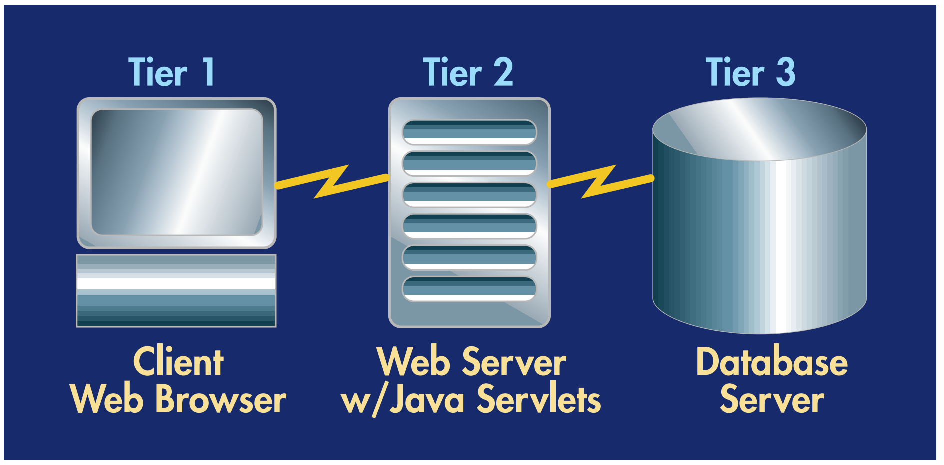


Figure 1: Three-tier solution

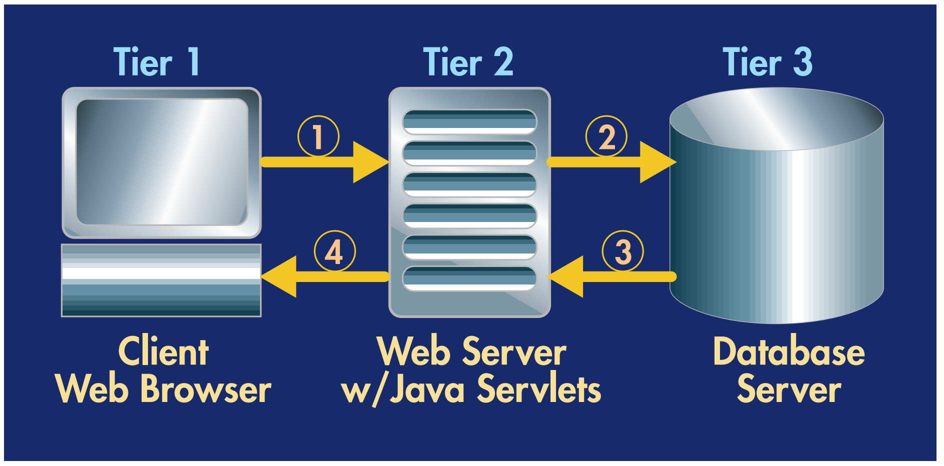


Figure 2: Application Interaction

Each record is used to create a new Student object. The toTableString() method is called to get a string representation of the student data. Recall that the toTableString() method returns the data as an HTML-formatted table row.

After the body of the table is constructed the result set is closed. At the bottom of the Web page, navigation links are provided to the main menu page.

A large amount of server-side processing has taken place. However, we are not finished yet. The HTML page must be returned to the Web browser. This is accomplished by opening an output stream on the *response* object. The *response* object is an instance of HttpServletResponse. The *response* object is used to respond to the client. The code for returning the HTML page to the user is:

```
   PrintWriter outputToBrowser =  new
PrintWriter(response.getOutputStream());
   response.setContentType("text/html");
   outputToBrowser.println(htmlPage);
   outputToBrowser.close();
```

The content-type is set for HTML and the *htmlPage* string is returned to the browser using the println() method. Figure 4 is a sample student list that is returned by the StudentDBServlet.

Registering A Student

The registerStudent() method creates a new Student object based on the HTML form data. The Student object is used to set the parameters on the SQL statement prepared in the init() method. The code fragment below shows how a parameter is set.

```
registerStatement.setString(LAST_NAME_POSI-
TION, aStudent.getLastName());
```

Once all of the parameters are set then the SQL statement is executed. After the statement is executed the new student data is successfully inserted into the database.

A confirmation page is constructed for the user. The confirmation page contains a list of the data that was successfully entered into the database. The Student.toWebString() method is called to provide an HTML string for an unordered list.

Pulling It All Together

At this point, all three tiers of the appli-

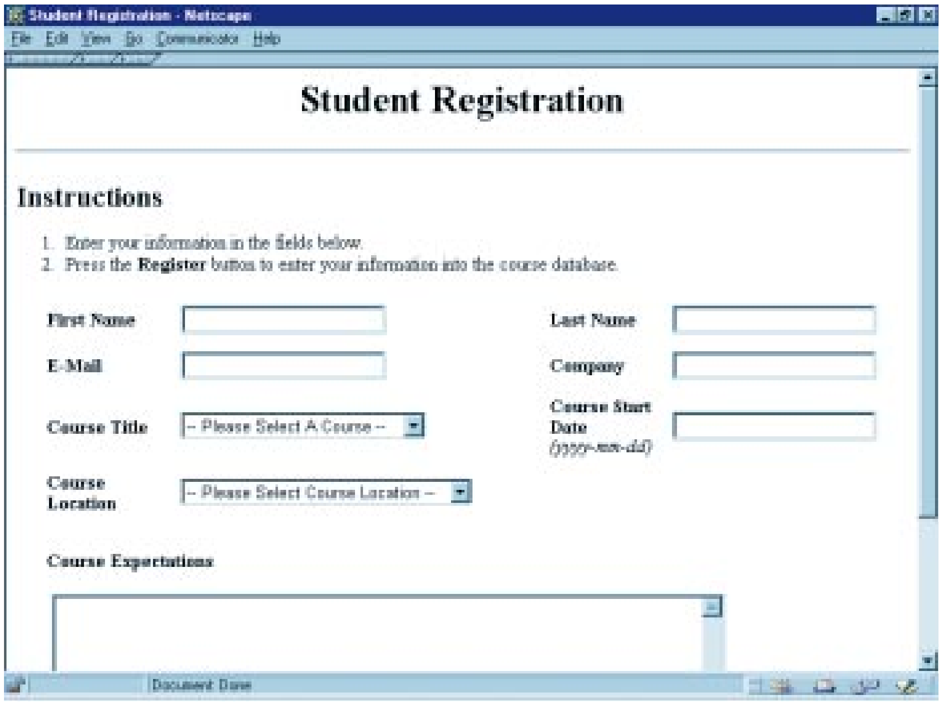
Greenbrier &
Russell
Full pg

cation are constructed. Collections of HTML pages represent the user interface component for the browser. The only requirement on the browser is the ability to display HTML tables. The two leading browsers available from Microsoft and Netscape easily satisfy this requirement, thus making the Web application browser-friendly.

The back-end database was developed with Microsoft Access. However, any database could have been used provided that a JDBC driver was available for the database. In our scenario, the public speaker was already tracking student data with MS Access. The Web application gave her the ability to access her legacy data and build on it.

The middleware was the critical piece of the application. The servlet middleware encapsulated the business logic and provided the "glue" between the Web browser interface and the backend database information. The database access was made possible by using a nice blend of Java-based technologies: Java Servlet API and JDBC.

Each of the components in the 3-tier application can reside on different computers. The application can easily be distributed across the network. With the worldwide reach of the Web browser, a user can enter information from a networked computer. The Java servlet middleware can reside on any servlet-enabled Web server. The servlet can, in turn, interact with any networked database server in a different location.


Student Registration - Netscape
File Edit View Go Communicator Help
Document: Done

Student Registration

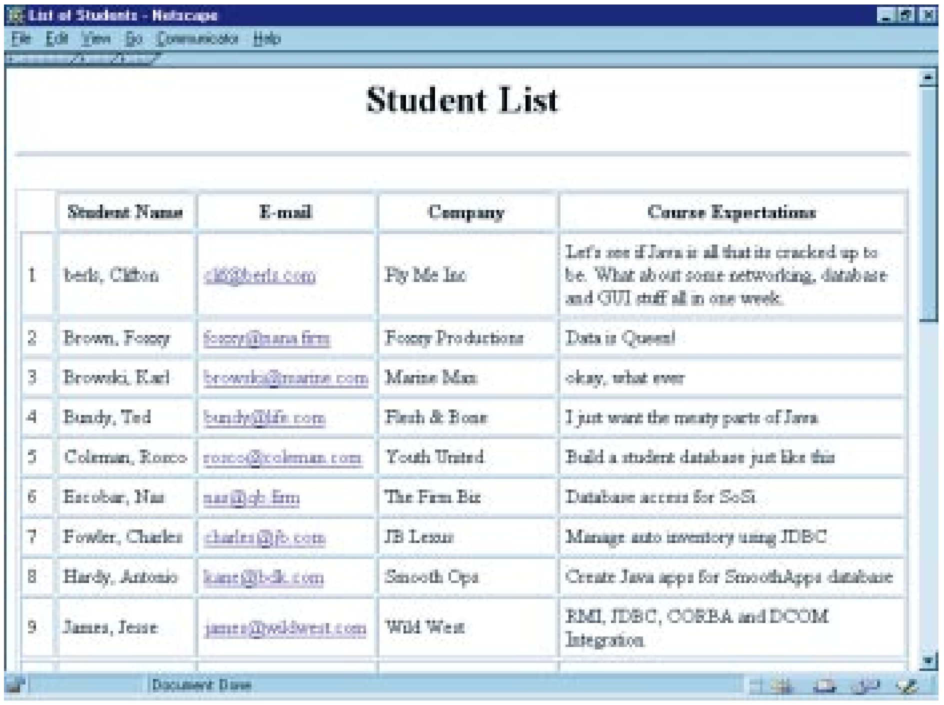
Instructions

1. Enter your information in the fields below.
2. Press the **Register** button to enter your information into the course database.

First Name		Last Name	
E-Mail		Company	
Course Title	-- Please Select A Course --	Course Start Date (yyyy-mm-dd)	
Course Location	-- Please Select Course Location --		

Course Expectations

Figure 3: Student Registration Form


List of Students - Netscape
File Edit View Go Communicator Help
Document: Done

Student List

	Student Name	E-mail	Company	Course Expectations
1	berk, Clifton	clif@berk.com	Fly Me Inc	Let's see if Java is all that its cracked up to be. What about some networking, database and GUI stuff all in one week.
2	Brown, Foxxy	foxxy@nana.firm	Foxxy Productions	Data is Queen!
3	Browski, Karl	browski@marine.com	Marine Man	okay, what ever
4	Bundy, Ted	bundy@life.com	Flesh & Bone	I just want the meaty parts of Java
5	Coleman, Rosco	rosco@coleman.com	Youth United	Build a student database just like this
6	Escobar, Nas	nas@gb.firm	The Firm Biz	Database access for SoSi
7	Fowler, Charles	charles@jb.com	JB Lexus	Manage auto inventory using JDBC
8	Hardy, Antonio	kane@bdk.com	Smooth Ops	Create Java apps for SmoothApps database
9	James, Jesse	james@wildwest.com	Wild West	RMI, JDBC, CORBA and DCOM Integration

Figure 4: Student List

Future Enhancements

This 3-tier Web application allowed students to register their student information. The application also gave the option of displaying an updated student list. However, the application is by no means complete. There are a number of enhancements that can be made to it.

The application can be enhanced to display students from a specific city, company or course. A user could create a custom query to generate the student listing. Also, the application can be enhanced to allow remote database administration features such as updating and deleting student entries. Currently, the application does not perform error checking on the form data entered. Client-side JavaScript can be used to verify user data. The application does not consider the case where a student will attend multiple seminars. The application can be enhanced to hold multiple course titles for a student.

Conclusion

This article presented the basic components and techniques to build a 3-tier database application. You can use this information to quickly and easily build a Web interface to your existing corporate database. The Java servlet technology coupled with the Java Database Connection are the key components in creating a Web application that is informative and interactive.

Resources

Article Source Code Listings:
http://www.j-nine.com/pubs/dbservlets
Sun Microsystem's Java Server Page:
http://jserv.javasoft.com
Live Software, JRun 2.0
http://www.livesoftware.com

About the Author

Chád (shod) Darby is a Java consultant for J9 Consulting, www.j-nine.com. He specializes in developing server-side Java applications and database applications. In his spare time he enjoys running 10K races and half-marathons. Chád can be reached at darby@j-nine.com.

darby@j-nine.com

Phaos
Full pg

Listing 1: HTML code for main menu.

```
<!-- Main Menu Page:  index.html -->
<HTML>
<HEAD>
   <TITLE>Student Database Connection</TITLE>
</HEAD>
<BODY>

<CENTER>
<H1>
Student Database Connection (SDBC)</H1></CENTER>

<HR WIDTH="100%">
<H2>
Options</H2>

<UL>
<LI>
<A HREF="StudentRegistration.htm">Register Online!</A></LI>

<LI>
<A HREF="/servlet/StudentDBServlet">View the Student List</A></LI>
</UL>
&nbsp;
</BODY>
</HTML>
```

Listing 2.

```
<!-- Student Registration page:  StudentRegistration.html -->

<HTML>
<HEAD>
   <TITLE>Student Registration</TITLE>
</HEAD>
<BODY>

<CENTER>
<H1>
Student Registration</H1></CENTER>

<HR>
<H2>
Instructions</H2>

<OL>
<LI>
Enter your information in the fields below.</LI>

<LI>
Press the <B>Register </B>button to enter your information into
the course
database.</LI>
</OL>
<FORM method="GET" action="/servlet/StudentDBServlet">
<CENTER><TABLE BORDER=0 CELLPADDING=5 WIDTH="95%" >
<TR>
<TD WIDTH="36%"><B>First Name&nbsp;</B></TD>

<TD WIDTH="50%"><INPUT type="text" name="FirstName" size="20"></TD>

<TD WIDTH="43%"><B>Last Name</B></TD>

<TD WIDTH="57%"><INPUT type="text" name="LastName" size="20"></TD>
</TR>

<TR>
<TD WIDTH="36%"><B>E-Mail&nbsp;</B></TD>

<TD WIDTH="50%"><INPUT type="text" name="Email" size="20"></TD>

<TD WIDTH="43%"><B>Company</B></TD>

<TD WIDTH="57%"><INPUT type="text" name="Company" size="20"></TD>
</TR>
<TR>
<TD WIDTH="36%"><B>Course Title</B></TD>

<TD WIDTH="50%"><SELECT name="CourseTitle" size="1"><OPTION select-
ed value="-- Please Select A Course --">--
Please Select A Course --</OPTION>&nbsp;<OPTION value="Java Intro-
duction">Java
Introduction</OPTION>&nbsp;<OPTION value="Java Database Apps">Java
Database
Apps</OPTION>&nbsp;<OPTION value="Java Network Programming">Java
Network
Programming</OPTION>&nbsp;<OPTION value="Java Distributed Computing
">Java
Distributed Computing</OPTION>&nbsp;<OPTION value="JavaBeans Intro-
duction">JavaBeans
Introduction</OPTION>&nbsp;<OPTION value="JavaBeans for the Enter-
prise">JavaBeans
for the Enterprise</OPTION>&nbsp;<OPTION value="Java Servlets">Java
Servlets</OPTION>&nbsp;<OPTION value="Java AWT &amp; JFC">Java
AWT &amp; JFC</OPTION>&nbsp;</SELECT></TD>

<TD WIDTH="43%"><B>Course Start Date </B><I>(yyyy-mm-dd)</I></TD>

<TD WIDTH="57%"><INPUT type="text" name="CourseStartDate"
size="20"></TD>
</TR>

<TR>
<TD><B>Course Location</B></TD>

<TD><SELECT name="CourseLocation" size="1"><OPTION selected
value="-- Please Select Course Location--">--
Please Select Course Location --</OPTION>&nbsp;<OPTION value="Hous-
ton, TX">Houston,
TX</OPTION>&nbsp;<OPTION value="Washington, DC">Washington,
DC</OPTION>&nbsp;<OPTION value="New York City, NY">New
York City, NY</OPTION>&nbsp;<OPTION value="Los Angeles, CA">Los
Angeles,
CA</OPTION>&nbsp;<OPTION value="Chicago, IL">Chicago,
IL</OPTION>&nbsp;<OPTION value="Atlanta, GA">Atlanta,
GA</OPTION>&nbsp;<OPTION value="Boston, MA">Boston,
MA</OPTION>&nbsp;<OPTION value="Biloxi, MS">Biloxi,
MS</OPTION>&nbsp;</SELECT></TD>

<TD></TD>

<TD></TD>
</TR>
</TABLE></CENTER>
&nbsp;
<CENTER><TABLE BORDER=0 CELLPADDING=5 WIDTH="95%" >
<TR>
<TD WIDTH="100%"><B>Course Expectations</B>&nbsp;

<P>&nbsp;<TEXTAREA rows="5" name="Expectations"
cols="66"></TEXTAREA></TD>
</TR>
</TABLE></CENTER>
&nbsp;
<CENTER><INPUT type="submit" value="Register"
name="Register"><INPUT type="reset" value="Reset Form"
name="B2"></CENTER>
</FORM>
<HR>
<CENTER><A HREF="index.html">Return to Course Home Page</A></CEN-
TER>

</BODY>
</HTML>
```

Listing 3: init() method.

```
//  File:  StudentDBServlet.java
//  Listing 3
//
```

Salesvision
Full pg

```
import javax.servlet.*;
import javax.servlet.http.*;

import java.sql.*;
import java.io.*;

import shod.register.Student;

/**
 *  This servlet provides data entry and retrieval of
 *  student data in a database.
 *
 *  @author Chad (shod) Darby,  darby@j-nine.com
 *  @version 0.6, 5 Jan 1998
 *
 */
public class StudentDBServlet extends HttpServlet
{
    // data members
    protected Connection dbConnection;
    protected PreparedStatement displayStatement;
    protected PreparedStatement registerStatement;

    protected String dbURL = "jdbc:odbc:StudentDatabase";
    protected String userID = "";
    protected String passwd = "";

    protected String CR = "\n";

    protected final int LAST_NAME_POSITION  = 1;
    protected final int FIRST_NAME_POSITION = 2;
    protected final int EMAIL_POSITION      = 3;
    protected final int COMPANY_POSITION    = 4;
    protected final int EXPECTATIONS_POSITION = 5;
    protected final int COURSE_TITLE_POSITION         = 6;
    protected final int COURSE_LOCATION_POSITION     = 7;
    protected final int COURSE_DATE_POSITION   = 8;

    public void init(ServletConfig config) throws ServletException
    {
        super.init(config);

        // use println statements to send status messages to Web
server console
        try {
            System.out.println("StudentDBServlet init: Start");

            System.out.println("StudentDBServlet init: Loading
Database Driver");
            Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");

            System.out.println("StudentDBServlet init: Getting a
connection to - " + dbURL);
            dbConnection = DriverManager.getConnection(dbURL,
userID, passwd);

            System.out.println("StudentDBServlet init: Preparing
display statement");
            displayStatement =
               dbConnection.prepareStatement("select * from Stu-
dents order by LastName");

            System.out.println("StudentDBServlet init: Preparing
register statement");
            registerStatement =
               dbConnection.prepareStatement("insert into Stu-
dents "
                 + "(LastName, FirstName, Email, Company, Course-
Expectations, CourseTitle, CourseLocation, CourseStartDate)"
                 + " values (?, ?, ?, ?, ?, ?, ?, ?)");

            System.out.println("StudentDBServlet init: End");
        }
        catch (Exception e)
        {
            cleanUp();
            e.printStackTrace();
        }
    }

    public void service(HttpServletRequest request,
                        HttpServletResponse response)
           throws ServletException, IOException
    {

        String userOption = null;

        userOption = request.getParameter("Register");

        if (userOption != null)
        {
            // hidden form field "Register" was present
            registerStudent(request, response);
        }
        else
        {
            // simply display the students
            displayStudents(request, response);
        }
    }

    public void displayStudents(HttpServletRequest request,
                                HttpServletResponse response)
    {
        Student aStudent = null;

        try {
            // build the html page heading
            String htmlHead = "<html><head><title>List of Stu-
dents</title></head>" + CR;

            // build the html body
            String htmlBody = "<body><center>" + CR;
            htmlBody += "<h1>Student List</h1>" + CR;
            htmlBody += "<hr></center><p>" + CR;

            // build the table heading
            String tableHead = "<center><table border width=100%
cellpadding=5>" + CR;
            tableHead += "<tr>" + CR;
            tableHead += "<th> </th>" + CR;
            tableHead += "<th>Student Name</th>" + CR;
            tableHead += "<th>E-mail</th>" + CR;
            tableHead += "<th>Company</th>" + CR;
            tableHead += "<th>Course Expectations</th>" + CR;
            tableHead += "</tr>" + CR;

            // execute the query to get a list of the students
            ResultSet dataResultSet = displayStatement.execute-
Query();

            // build the table body
            String tableBody = "";

            int rowNumber = 1;
            while (dataResultSet.next())
            {
                aStudent = new Student(dataResultSet);
                tableBody += aStudent.toTableString(rowNumber);
                rowNumber++;
            }

            dataResultSet.close();

            // build the table bottom
            String tableBottom = "</table></center>";

            // build html page bottom
            String htmlBottom = "</body></html>";
```

Live software
Full pg

```
            // build complete html page
            htmlBody += tableHead + tableBody + tableBottom;
            htmlBody += "<p><hr>";
            htmlBody += "<center><a
href=/StudentDB/index.html>Return to Course Home Page</a>";
            htmlBody += "<p><i>" + this.getServletInfo() +
"</i>";
            htmlBody += "</center>";
            String htmlPage = htmlHead + htmlBody + htmlBottom;

            // now let's send this dynamic data
            // back to the browser
            PrintWriter outputToBrowser =  new
PrintWriter(response.getOutputStream());
            response.setContentType("text/html");
            outputToBrowser.println(htmlPage);
            outputToBrowser.close();

        }
        catch (Exception e)
        {
            cleanUp();
            e.printStackTrace();
        }
    }

    public void registerStudent(HttpServletRequest request,
                                HttpServletResponse response)
    {
        try {
            // create a new student based on the form data
            Student aStudent = new Student(request);

            // set sql parameters
            registerStatement.setString(LAST_NAME_POSITION, aStu-
dent.getLastName());
            registerStatement.setString(FIRST_NAME_POSITION, aStu-
dent.getFirstName());
            registerStatement.setString(EMAIL_POSITION,
aStudent.getEmail());
            registerStatement.setString(COMPANY_POSITION, aStu-
dent.getCompany());
            registerStatement.setString(EXPECTATIONS_POSITION,
aStudent.getExpectations());
            registerStatement.setDate(COURSE_DATE_POSITION, aStu-
dent.getCourseDate());
            registerStatement.setString(COURSE_TITLE_POSITION,
aStudent.getCourseTitle());
            registerStatement.setString(COURSE_LOCATION_POSITION,
aStudent.getCourseLocation());

            // execute sql
            registerStatement.executeUpdate();

            // build confirmation page
            String htmlPage = "<html><head><title>Confirmation
Page</title></head>";

            htmlPage += "<body>";
            htmlPage += "<center><h1>Confirmation Page</h1></cen-
ter><hr>";
            htmlPage += "The following information was entered
successfully";
            htmlPage += aStudent.toWebString();

            htmlPage += "<hr>";
            htmlPage += "<center><a
href=/StudentDB/index.html>Return to Home Page</a> | ";
            htmlPage += "<a href=/servlet/StudentDBServlet>View
Student List</a>";
            htmlPage += "<p><i>" + this.getServletInfo() +
"</i>";
            htmlPage += "</center></body></html>";

            // now let's send this dynamic data
            // back to the browser
            PrintWriter outputToBrowser =  new
PrintWriter(response.getOutputStream());

            response.setContentType("text/html");
            outputToBrowser.println(htmlPage);
            outputToBrowser.close();
        }
        catch (Exception e)
        {
            cleanUp();
            e.printStackTrace();
        }
    }

    public void cleanUp()
    {
        try {
            System.out.println("Closing database connection");
            dbConnection.close();
        }
        catch (SQLException e)
        {
            e.printStackTrace();
        }
    }

    public void destroy()
    {
        System.out.println("StudentDBServlet: destroy");
        cleanUp();
    }

    public String getServletInfo()
    {
        return "<i>Student Registration Servlet, v.06</i>";
    }
}
```

Listing 4: Student class.

```
//  File:  Student.java
//  Listing 4
//
package shod.register;

import java.sql.*;
import javax.servlet.http.*;

/**
 *  The Student class has data members to describe
 *  a student. String methods are available to
 *  display the data members to the console or Web page.
 *
 *  @author Chad (shod) Darby,  darby@j-nine.com
 *  @version 0.6, 5 Jan 1998
 *
 */
public class Student
{
    // data members
    protected String lastName;
    protected String firstName;
    protected String company;
    protected String email;
    protected String courseTitle;
    protected String courseLocation;
    protected String expectations;
    protected java.sql.Date courseDate;

    protected final String CR = "\n";     // carriage return

    // constructors
    public Student()
    {
    }
```

ILOG
Full pg

```
    public Student(HttpServletRequest request)
    {
        lastName = request.getParameter("LastName");
        firstName = request.getParameter("FirstName");
        email = request.getParameter("Email");
        company = request.getParameter("Company");

        String dateString = request.getParameter("CourseStart-
Date");
        courseDate = java.sql.Date.valueOf(dateString);

        courseTitle = request.getParameter("CourseTitle");
        courseLocation = request.getParameter("CourseLocation");
        expectations = request.getParameter("Expectations");
    }

    public Student(ResultSet dataResultSet)
    {

        try {
            // assign data members
            lastName = dataResultSet.getString("LastName");
            firstName = dataResultSet.getString("FirstName");
            email = dataResultSet.getString("Email");
            company = dataResultSet.getString("Company");
            expectations = dataResultSet.getString("CourseExpecta-
tions");
            courseTitle = dataResultSet.getString("CourseTitle");
            courseLocation = dataResultSet.getString("CourseLoca-
tion");
            courseDate = dataResultSet.getDate("CourseStartDate");
        }
        catch (SQLException e)
        {
            e.printStackTrace();
        }
    }

    //  accessors
    public String getLastName()
    {
        return lastName;
    }

    public String getFirstName()
    {
        return firstName;
    }

    public String getEmail()
    {
        return email;
    }

    public String getCompany()
    {
        return company;
    }

    public String getExpectations()
    {
        return expectations;
    }

    public String getCourseTitle()
    {
        return courseTitle;
    }

    public String getCourseLocation()
    {
        return courseLocation;
    }

    public Date getCourseDate()
    {
        return courseDate;
    }


    //  methods
    //  normal text string representation
    public String toString()
    {
        String replyString = "";

        replyString += "Name: " + lastName + ", " + firstName +
CR;
        replyString += "E-mail: " + email + CR;
        replyString += "Company: " + company  + CR;
        replyString += "Course Expectations: " + expectations +
CR;
        replyString += "Course Title: " + courseTitle + CR;
        replyString += "Course Location: " + courseLocation + CR;
        replyString += "Course Start Date: " + courseDate + CR +
CR;

        return replyString;
    }

    //  returns data as HTML formatted un-ordered list
    public String toWebString()
    {
        String replyString = "<ul>";

        replyString += "<li><B>Name:</B> " + lastName + ", " +
firstName + CR;
        replyString += "<li><B>E-mail:</B> " + email + CR;
        replyString += "<li><B>Company:</B> " + company  + CR;
        replyString += "<li><B>Course Expectations:</B> " + expec-
tations + CR;
        replyString += "<li><B>Course Title:</B> " + courseTitle +
CR;
        replyString += "<li><B>Course Location:</B> " + courseLo-
cation + CR;
        replyString += "<li><B>Course Start Date:</B> " + course-
Date + CR;

        replyString += "</ul>" + CR;

        return replyString;
    }

    // returns data formatted for an HTML table row
    public String toTableString(int rowNumber)
    {
        String replyString = "";
        String tdBegin = "<td>";
        String tdEnd = "</td>" + CR;

        replyString += "<tr>" + CR;
        replyString += tdBegin + rowNumber + tdEnd;
        replyString += tdBegin + lastName + ", " + firstName +
tdEnd;
        replyString += tdBegin + "<a href=mailto:" + email + "> "
                             + email + "</a>" + tdEnd;

        replyString += tdBegin + company + tdEnd;
        replyString += tdBegin + expectations + tdEnd;
        replyString += "</tr>" + CR;

        return replyString;
    }
}
```

Softech
1/2 Ad

Mind Q
1/2 Ad

Designing Objects for CONCURRENCY in Java

Designing Objects for
Designing Objects for
in Java
in Java

Meta-level Programming Model Part 3

Ensuring that server components can be deployed on any system

by Jordan Anastasiade

Despite extensive development over many years and significant demonstrated benefits, the object-oriented paradigm remains poorly formalized. Several concurrent object-oriented languages have been designed and implemented based on the concurrent object model. However, upon attempting to apply formal techniques to a significant application, several well known shortcomings actually impeded progress dramatically right at the outset.

*In the second part of our series (**JDJ**, Vol. 2 Iss. 6), we defined the meaning of the inheritance anomaly to describe the conflict between inheritance and concurrency in object-oriented languages. It has been proven that Java at the semantic level is powerful enough to provide a mechanism for solving the inheritance anomaly. However, semantic conflicts often occur using even a simple and elegant language like Java. The concept of Meta-Object will be introduced as a methodology to obtain a separation of the protocol from functionality in class definition and an evaluation framework for a server component model will be defined. This part deals mostly with design topics and leaves implementation details and performance issues for future articles.*

Reasons for a Meta-Level Approach

Despite their popularity, the basic principles of object-oriented programming – encapsulation, inheritance, polymorphism – are still not used correctly. How do we implement encapsulation today? The object fields are private/protected but the object has better methods like pleaseSetMyPrivateData(). I once saw the following code:

```
public class SplendidEncapsulation {
 private String s = "SENSITIVE DATA";
 public void setS
  (SplendidEncapsulation anotherObject) {
  anotherObject.s = "MODIFY SENSITIVE DATA";
 }
}
```

Imagine x, y object of type SplendidEncapsulation and an invocation of type y.setS(x). Would you consider using the x object after y.setS(x) is executed? Personally, I wouldn't recommend it. Although the intention was clearly to set the data of object y, questionable in itself, the sample shows how today these hurdles arise from the current practical software development. There is clearly a problem here, as nothing prevents us from constructing language in syntactically correct forms [3], that could lead to the miserable failure of a software product. It is obviously a semantic interoperability problem between object requester and object provider. Indeed, we might now wonder how a viable concept like encapsulation could be enforced.

I believe that making implicit semantics **explicit** and accessible at the **meta level** would allow semantic forms to be converted into syntactic forms and thus make them amenable to automated detection and possible resolution. Information hiding must be maintained and the ability to modify the object state at varying levels of abstraction must be provided. Various authors have proposed different solutions, most of them converging to a meta-level model that will be explained shortly. A comprehensive approach relies on design patterns methodology [1]. An object has a state, behavior and identity. The object behavior needs to change as its state is modified. The State pattern provides a good conceptual model and it is a valuable pattern to master because it is a practical way of avoiding mistakes such as those described in the SplendidEncapsulation class. However a StateType object can encapsulate the state-transition behavior and may be used to predict the result of an inconsistent state of the object. Correcting the error using the State pattern is, however, an important prelude to a formal and comprehensive object manipulation model. It seems likely that the use of formal models will become standard practice in software engineering. It is generally recognized that no viable model, describing a self controlled object, can be designed. Objects are becoming increasingly concurrent and distributed and also, in many cases, they are also mobile. Therefore, the semantic models, assuming a sequential execution context or those not addressing concurrency and distribution, will be of limited value.

As you can see, there are many reasons to consider a fundamental and unified approach for a new semantic foundation. To accommodate objects in a newly created computing environment we need a way to encapsulate and control not only the object state (data) but also the object behavior (methods). Hold on; you may say using Java and encapsulating object behavior means having only private/protected methods. How could I further use my object? In fact, your object will not be used, at least not directly. A client object will see only an object wrapper, a controlling object called Meta-Object.

Meta-Object

So what is a meta-object? A meta-object of a base-level object is an object that defines the semantic of the base-level object behavior. The client object will never interact directly with a base-level object. Instead, the client interacts with an external object representation, the meta-object. The main goal of the new concept of a meta-object is to separate **what an object does** – the base-level implementation – from **how it does it** – the object meta-level implementation.

A clear separation of concerns is defined in the meta-level programming model. Thus, the main concern at the base-object level is to solve the application domain problem. The meta-object can alter the computational characteristics of an object regardless of its base-level semantics. A passive object, for example a sequential base-level object, could exhibit a different behavior into a concurrent environment without modifying object code. Using the meta-objects for concurrency control is the ideal control form since it provides the most flexible control mechanism [2]. Meta-objects are responsible for delivering messages between base-level objects so that the meta-objects' implementation can provide optimal communication protocols and scheduling policies. The data of a meta-object is called metadata and it is the base-level object data representation at its metalevel. Even at this earlier stage of the meta level object model definition, several mechanisms might be beneficial to us. A meta-object may inspect the state of a base-level object within its execution environment and free us from writing object code which depends on one particular computational system. Eventually you ought to deal with concurrency, security, distribution and other services, but only once at the meta-object level.

How could a base-level object maintain its state in different contexts of execution? At this point in our model design, I believe it is an easy question to answer: The persistence will be implemented at the meta-level and it is the responsibility of the meta-object to maintain the persistence of its base-level object. As an exercise, let's try to find one way of implementing the persistence at the meta-level. A handle type object that identifies a meta-object will solve our problem; from a meta-object we can obtain its handle, and vice-versa, from handle we can generate the meta-object. What we need is a class, let us call it Handle, that implements java.io.Serializable. The meta-object will implement the default serialization mechanism for its base-level object, using a symbolic model for binding its fields in the stream to the fields in the corresponding base-level object.

A client that has a reference to a meta-object can obtain meta-object's handle by invoking getHandle() method on the reference:

```
 ObjectOutputStream stream =
new ObjectOutputStream(…);
 MetaObject metaobject = new MetaObject(…);
 Handle handle = metaobject.getHandle();
 stream.writeObject(handle);
```

A client can later read the handle for the storage and will recreate a meta-object reference from the handle:

```
 ObjectInputStream stream =
new ObjectInputStream(…);
 Handle handle = (Handle)stream.readOb-
ject();
 MetaObject metaobject =
handle.getMetaObject(…);
```

Would you recognize the advantage of redirect serialization through a meta-object handle? Consider the scenario in which the parts of technology used by the container have been replaced or upgraded. The base-level object can be resurrected because the client stored only a meta-object handle. It seems a small gain, but if you think of the speed of today's transformations and the diversity of our technologies, you will realize the advantages of a model that is not domain-specific. Let me assume that you agree about the viability of a meta-object concept. You may ask about a methodology to check the consistency of a newly created meta-object. It is a natural question and I hope the answer will satisfy your concerns. In order to validate a meta-object we need to extrapolate the type of base-meta relationship to another layer. A new kind of relationship should be established between our meta-objects and a meta-meta-object if you wish, that will be called, for simplicity, object Cluster or Container.

"A meta -object separates what it does from how it does it"

Container: An Object for Controlling Meta-Objects

To achieve the separation of execution domain from object programming, defining meta-objects is a necessary condition but not sufficient. Recently, it has become evident that programming by contract at the interface level is a step in the right direction, but at the same time we need to define all the operational constraints into a uniform space where the meta-objects live. In programming, as in life, our list of basic concerns is open-ended. However, we should not lose sight of the goal: to define a unified domain, a context where all the laws of object existence are simple to express and apply.

Thus, a container could provide the life cycle management for each meta-object component. If you think of the Factory pattern, we already have in place the container methodology for allowing clients to create meta-objects inside a container. The list of basic constraints can be extended, including location control, failure recovery, security services and so on. Effectively implementing a portability layer, a container lets us express design constraints and high-level system properties in a modular way, so it provides a context environment for meta-objects. The interaction between base-level objects and the container is depicted in Figure 1. The better we understand the model, the more we will know how to apply it to the framework for practical software development. If we are to understand the model design's conceptual simplicity, I need to introduce and define two terms: reflection and reification.

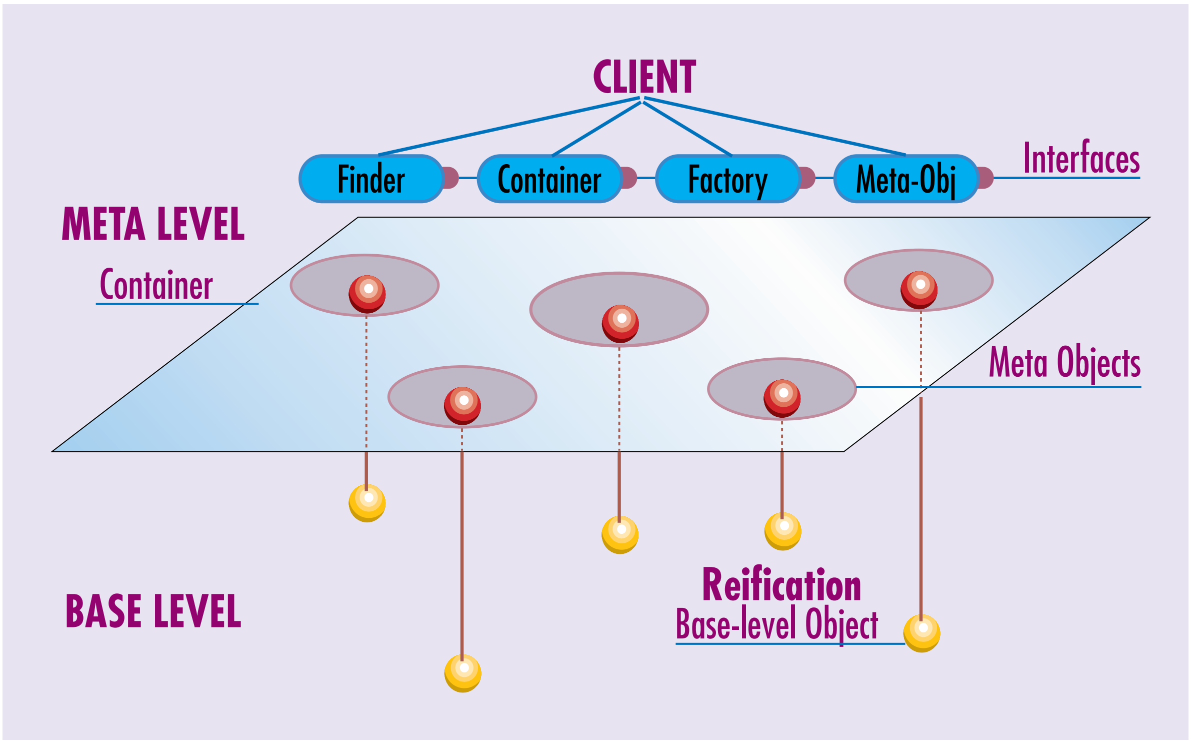


Figure 1: Interaction between base-level objects and container

The ability of a computational system to reason about and act upon itself is called reflection. Structural reflection reifies the organizational aspects of model structure. You may have studied or used the Java reflection package or runtime type identification in C++ which are implementations of structural reflection. A base-level object could be aware of changes at the meta-level environment using reflection concepts. Thus, we already have in place a methodology for reflecting structures of meta-level into base-level.

The act of making hidden information accessible is called reification. The architectural design of our model implies an enforced encapsulation not only of base-level object data, but also of its behavior. The client would be able to use the computational algorithms encapsulated in base-level objects only if the model implements the reification concept. Let us suppose that a client wants to obtain some results and it knows the location of a container satisfying its demands. The first thing the client has to do is to invoke a create() on the container's factory. The container will invoke a newInstance() method to create a new base-level object. The container will call further a setContext() function to define the computation environment in which the base-level object lives, followed by a createMetaObject(...); finally, it returns a meta-object reference to client. The meta-object exposes all the application-domain methods of the base-level object but not the interfaces that allow the container to manage and control the object. Thus, the client interacts only with a meta-object, the external representation of the base-level object whose behavior is encapsulated and reified at the meta-level.

The container may implement different methods on behalf of base-level objects like restoring their previously stored state or activating base-level objects and so forth. Many of these concepts enable a container to control the execution path regardless of the underlying operating system. All services are available to the client and more importantly to the base-level object whose main concern was only to solve the application domain-specific problem.

Unfortunately, in software engineering you cannot validate results purely by proving theorems. On the other hand, formalization still plays a fundamental role in software development because you must have a formal model of a domain before you can design effective software for that domain. Measuring the value of a model by its impact on practical software development is the only way to find out the viability of the model.

Server Component Model

As you know, in a traditional client/server relationship the client application contains graphical presentation/interaction with the user, algorithms for solving specific domain problems and data manipulations depending usually on an underlying operating system. It is natural for such an application, which has been labeled as fat-client, to be unreliable, difficult to maintain and to integrate in any changing computational environment. I suppose everyone knows that thin clients are in great demand nowadays, not as a matter of fashion but as a normal consequence of a Web-based computing environment. At least, at the theoretical level, the multitier concept has been around for almost a decade. The difficult part of the transition process is to define a framework in which the server components are reusable. The main reason I introduced to you the meta-level programming architecture was to create a framework for building reusable server components.

Let's consider our base-level object as a Bean [4]. This means that our base-level objects have attributes that affect their behavior. The properties of such base-level objects could be bound, constrained and, more importantly, customized. Our objects do not have java.awt.Component as an ancestor since they are what is called an invisible Bean. Based on a well-defined protocol between a Bean and its container, the model must specify protocol interfaces. Thus, a server component model should define not only the basic architecture of a component, but also should specify the structure of its interfaces and the mechanisms by which a component interacts with its container and with other components. A typical component representation will distinguish between components that are actively transforming data and passively storing data. The Container itself does not make service demands on the meta-objects. The calls a container makes provide a meta-object with access to container services and delivers notifications issued by the component. The container might also define the ContainerContext object which gives base-level access to its container and most importantly to its meta-object.

There should be an interface container that allows a client to do the following:

1. Obtain a factory object that allows a client to create a new meta-object in the container
2. Obtain a finder object that allows a client to look up an existing meta-object in the container
3. Destroy meta-objects

Eventually, the container insulates the base-level object from the specifics of an underlying component server providing a simple, standard protocol between base-level objects and a container. Thus, a client's view of a base-level object is unaffected by the container and server the object is deployed in. Therefore, a normal component server will provide a scaleable runtime environment for a large number of concurrently active base-level objects. An object cached can be shared by many clients and the performance can greatly improve for objects which are frequently read but seldom modified. Server components can be replicated and distributed across any number of servers to boost system availability and performance.

The model described has a current implementation in Enterprise JavaBeans [4], which defines a concrete component model to support multitier, distributed object applications.

Conclusion

The meta-level programming concept generates a simple and elegant server component container model. The model ensures that server components can be developed once and deployed to any system. Even though the container systems implement their runtime services differently, the interfaces ensure that a server component can rely on the underlying system to provide consistent life cycle, persistence, transaction, distribution and security services. In spite of using simple architectural primitives like reflection and reification, the model automates the use of complex infrastructure services such as transactions, thread management, and security checking.

The server component model built using a meta-level architecture has many advantages. Future articles will demonstrate how moving data manipulation logic to a server allows an application to take advantage of the power of multithreading in Java.

References

1. E. Gamma, R. Helm, R. Johnson, J. Vlissides, "*Design Patterns.*" Addison-Wesley, 1995
2. D. Lea, "*Concurrent Programming in Java Design Principles and Patterns.*" Addison-Wesley, 1996.
3. J. Gosling , B. Joy, G. Steele, "*The Java Language Specification.*" Addison-Wesley, 1996.
4 T. Lorentz "Making Enterprise Java a Reality." ***Java Developer's Journal***, Volume 2, Issue 12, www.javadevelopers-journal.com

About the Author

Jordan Anastasiade holds a BS in Architecture and an MS in Mathematics. He works for Hummingbird Communications Ltd., focusing on design patterns using object-oriented techniques in Java. Jordan can be reached at jordan@hummingbird.com

jordan@hummingbird.com

Bristol

1/2 Ad

Thread Pooling

Presenting a generic thread manager for thread pooling

by **Brian Maso**

I don't have to argue the point that the Java language's multi-threading capabilities are great. They're simple to use and generic enough to work on a variety of different implementations. Whether a VM is made to operate using a single operating system thread, like Microsoft's VM, or using native operating system threads for Java threads, like Sun's native threading VM for Solaris, your Java programs will work. I think that's a quiet but powerful feature of Java.

Perhaps it is almost too easy to create and run background threads in Java. Perhaps the designers of Java could have thrown a couple of snafus in there so we programmers didn't go quite so hog-wild sometimes. It can be a problem because a Java VM can't support infinite running threads. In fact, creating a Thread object and starting it running can be very costly in terms of memory.

Don't believe me? Take a look at Listing 1. It's a simple program that just creates as many background Threads as you tell it to on the command-line. Try typing in that program and running it with 1000, or even 10,000, threads. What you should see is an OutOfMemoryError appear after not too long. That's because each background thread requires a lot of memory to run. OutOfMemoryErrors are particularly nasty because there's no verifiable way to recover from them. A program that creates one too many background threads will find itself unable to run and basically will just have to quit.

I really like multi-threaded programs, to the point that I sometimes (not too often) run into this problem. For example, if I use three or four different multi-threaded packages I've created in the past, none of which had an OutOfMemoryError-type problem on their own, I may find the combination of the threads created by the different subsystems is just too much for a particular VM to handle.

The idea of thread pooling should immediately jump to mind. Thread pooling is when you create a fixed number of background threads and allow your program to use just those finite number of threads, instead of allowing my program objects to create an infinite number of background threads willy-nilly. Usually, thread pooling is used in a client/servant situation. That is, where a servant object (or collection of objects) fulfills any number of simultaneous client requests.

An unconstrained design would have each client request handled by an individual background thread. Listing 2 is an

Listing 1: UnconstrainedThreadCreator program creates as many background threads as indicated on the command-line.

```
public class UnconstrainedThreadCreator
  implements Runnable {
 private int m_id;
  private static int m_nextId = 1;

  public static void main(String[] astrArgs) {
    try {
      int nThreads = Integer.parseInt(astrArgs[1]);
      for(int ii=0 ; ii<nThreads ; ii++)
        new UnconstrainedThreadCreator();
      return;
    } catch (Exception e) {
      System.err.println(e);
      e.printStackTrace(System.err);
    }
  }

  public UnconstrainedThreadCreator() {
    synchronized(getClass()) {
      m_id = m_nextId++;
    }
      (new Thread(this)).start();
    }

    public void run() {
      Thread self = Thread.currentThread();
      try {
        while(true) {
          self.sleep(100);
          System.out.println("Thread " + m_id + " looping...");
        }
      } catch (InterruptedException ie) {
      }
    }
  }
```

Listing 2: Internet server class with unconstrained thread creation; one thread is created per client request. (ClientHandler class omited for brevity.)

```
public class INetServer {

  public static void main(String[] astrArgs) {
    try {
      ServerSocket ss = new ServerSocket(8888);
```

Java One

example of an Internet-based service. Each request comes in the form of a client connection to a particular port on the server machine (in this case, port 8888). Listing 2 shows what a main server thread would do: creating a new ClientHandler object and a new Thread for each client request. This main thread is susceptible to being overwhelmed by too many simultaneous client requests. Too many requests and the nasty OutOfMemoryError will appear, probably shutting down the entire server program.

Listing 3 is a ThreadPoolManager class. This class is used to manage a constrained number of threads. Each thread managed by the ThreadPoolManager has a potentially infinite lifetime. Each Runnable object passed to the ThreadPoolManager is queued, and when a thread becomes available to handle the Runnable's task, then that thread is used to run the Runnable's run() method. This way, a huge number of Runnable objects can each have their tasks eventually run without overwhelming the VM by having too many simultaneous running threads. Listing 4 is a replacement for the main thread routine of the server shown in Listing 2. The only change is that the new server uses a ThreadPoolManager's threads instead of creating new threads for each client request.

About the Author

Brian Maso is a programming consultant working out of California. He is the co-author of The Waite Group Press's, "The Java API SuperBible." Before Java, he spent five years corralled in the MS Windows branch of programming, working for such notables as the Hearst Corp., first DataBank, and Intel. Readers are encouraged to contact Brian via e-mail with any comments or questions at bmaso@developer.com.

bmaso@developer.com

```
      while(true) {
        Socket s = ss.accept();
        Client Handler ch = new ClientHandler(
            s.getInputStream(), s.getOutputStream());
        (new Thread(ch)).start();
      }
    } catch (Exception e) {
       System.err.println(e);
       e.printStackTrace(System.err);
    }
  }

}
```

Listing 3: ThreadPoolManager class and ManagedThread class.

```
public class ThreadPoolManager {
  private Vector m_runnableVector;

  public ThreadPoolManager(int threadPoolSize) {
    m_runnableVector = new Vector(1, 10);

    for(int ii=0 ; ii<threadPoolSize ; ii++)
      new ManagedThread(this);
  }

  synchronized void threadWaiting(ManagedThread mt)
      throws InterruptedException {
    while(0 == m_runnableVector.size())
      wait();

    Runnable r = (Runnable)m_runnableVector.elementAt(0);
    m_runnableVector.removeElementAt(0);
    mt.startRunnable(r);
  }

  public synchronized void start(Runnable r) {
    m_runnableVector.addElement(r);
    notify();
  }
}

class ManagedThread extends Thread {
  private ThreadPoolManager m_manager;

  ManagedThread(ThreadPoolManager manager) {
    m_manager = manager;
    start();
  }

  public void run() {
    try {
      while(true)
        m_manager.threadWaiting(this);
    } catch (InterruptedException ie) {
      // just quit.
    }
  }

  void startRunnable(Runnable r) {
    try {
      r.run();
    } catch (Exception e) {
      // Print out exceptions thrown by Runnables,
      // but return normally to allow thread to
      // continue handling cliuent requests.
      System.err.println("Error running " + r);
      System.err.println(e);
      e.printStackTrace(System.err);
    }
  }
}
```

Listing 4: The Internet server program using thread pooling to handle client requests.

```
public class INetServer {

  public static void main(String[] astrArgs) {
    try {
      ThreadPoolManager tpm = new ThreadPoolManager(100);
      ServerSocket ss = new ServerSocket(8888);

      while(true) {
        Socket s = ss.accept();
        Client Handler ch = new ClientHandler(
            s.getInputStream(), s.getOutputStream());
>
        /* HERE'S THE ONLY DIFFERENCE */
        tpm.start(ch);
      }
    } catch (Exception e) {
       System.err.println(e);
       e.printStackTrace(System.err);
    }
  }

}
```

Coriolius

Objective Grid/J 1.2

by Stingray Software, Inc.

Developers can use the Objective Grid Control anywhere they can use an AWT control, saving time and effort

by **Ed Zebrowski**

Objective Grid/J 1.2
Stingray Software Inc.
3000 Aerial Center, Suite 110
Morrisville, NC 27560
Phone: 800-924-4223
Fax: 919 461-9811
Web: http://www.stingray.com
Email: sales@stingray.com
Requirements: Sun's JDK 1.0 or 1.1
Price: $395

JAVA DEVELOPER'S JOURNAL JDJ WORLD CLASS AWARD

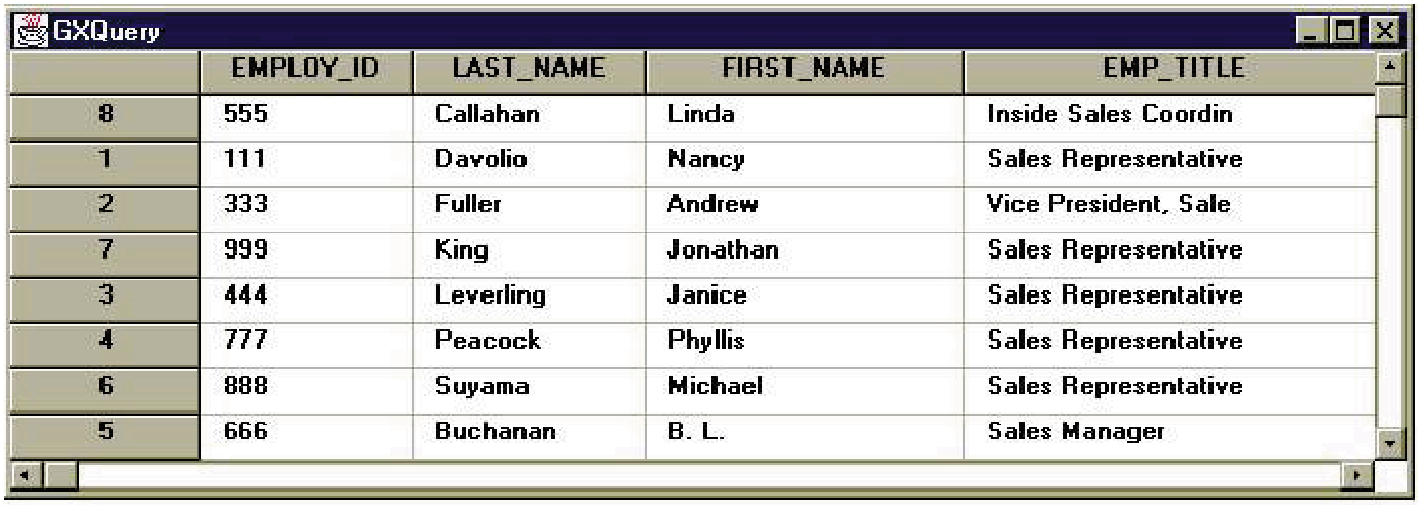
Many of the smaller tasks which Java developers are required to take on within a larger project can take on the air of a larger project all on their own. Developing a grid control can be one of those tasks. Many companies have responded to this need by creating their own "plug-and-play" grids for Java developers to implement in their own projects. Stingray's Objective Grid for Java is a fine example of tools which can enhance a Java application and shorten design time. Developers can use the Objective Grid control anywhere they can use an AWT control, saving time and effort.

Visually, Objective Grid tends to remind me of the typical spreadsheet, with rows and columns to hold the data (see Figure 1). I can remember when I first saw how Web pages were created using standard HTML, I thought to myself that what the Web needed was a way to use spreadsheets with dropdown combo boxes, listboxes, images, masked input and buttons. Along came Java and the door was opened, but it still had its limitations – the main one of which was the time it would take to include this functionality in an applet or application. Using Objective Grid, the programmer is able not only to include these features in their project, but also to include editing functions such as find and replace, undo and redo and, if the programmer is using the 1.1 version of the JDK, even the ability to print. The grid can also be bound to external data sources using the JDBC. So what exactly is Objective Grid/J?

Objective Grid/J is a package of Java extension classes that implements a user interface component which displays data in rows and columns. The user interface is known as a grid control and enables end-users to manipulate and edit the data which is displayed within it. Version 1.2 now enables developers to give their applets and applications an Excel-like look and feel. Objective Grid/J also allows developers using the latest releases of Visual J++, Symantec Café, Symantec Visual Café and Supercede to include grids in their applets and applications.

Objective Grid includes complete project files for the libraries as well as the demos, including source code for every class. The Objective Grid package is actually composed of several groups of Java extension classes which work together to create the grid effects mentioned above. These classes include:

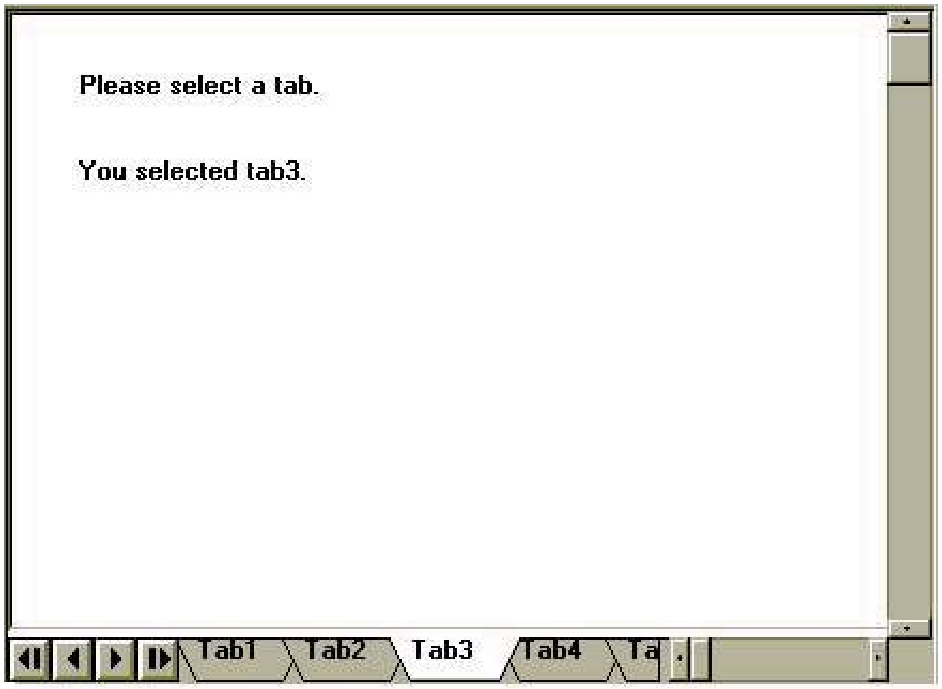
- Drawing Classes: The drawing classes actually perform the drawing and updating of cells that are displayed. The base GXGridCore class is derived from the


GXQuery

	EMPLOY_ID	LAST_NAME	FIRST_NAME	EMP_TITLE
8	555	Callahan	Linda	Inside Sales Coordin
1	111	Davolio	Nancy	Sales Representative
2	333	Fuller	Andrew	Vice President, Sale
7	999	King	Jonathan	Sales Representative
3	444	Leverling	Janice	Sales Representative
4	777	Peacock	Phyllis	Sales Representative
6	888	Suyama	Michael	Sales Representative
5	666	Buchanan	B. L.	Sales Manager

Figure 1: Objective Grid typically looks like a spreadsheet, with rows and columns to hold the data.

Sybex

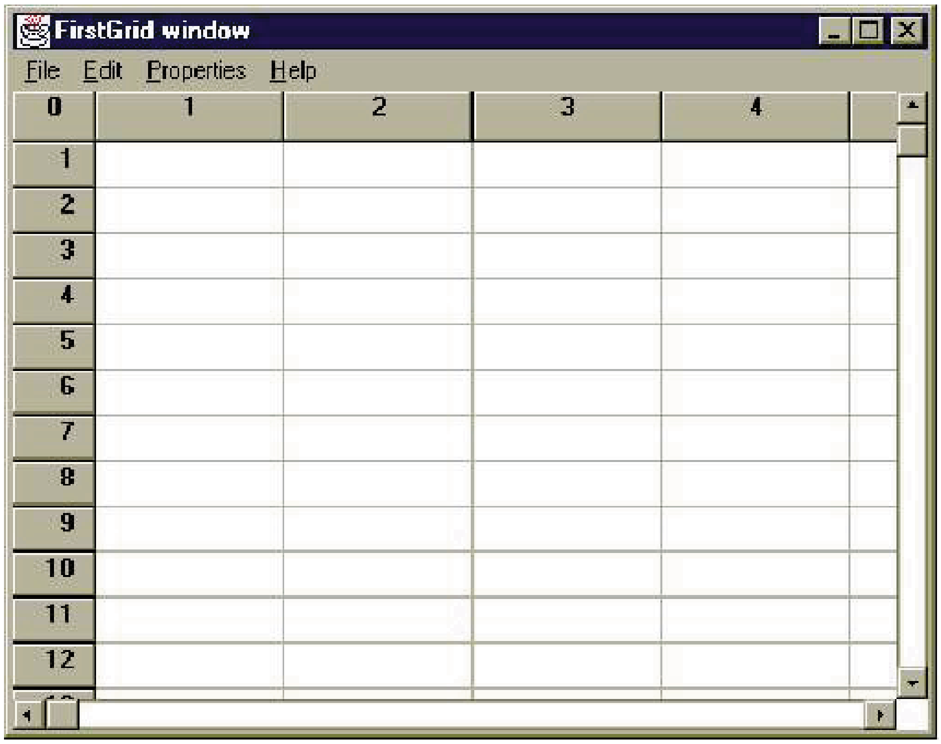

Please select a tab.

You selected tab3.

Figure 2: The tabbed panel class showcases a row of tab buttons with different Panels associated with each "tab."

AWT Panel class, and classes are provided to include a binary runtime of Stingray's Objective Blend product.

- Control Classes: Grid cells may be any of a variety of control types; several pre-built controls are ready to be placed in as cell types (you can also embed your own custom controls as cell types).
- Style Classes: The Objective Grid style classes manage attributes of a cell (or group of cells) and provide a pre-built dialog which is used to modify them.
- Browser Classes: The Browser classes enable the programmer to browse external data sources by overriding some of the virtual methods.
- JDBC Classes: The JDBC classes provide grid access to JDBC databases.
- Symantec dbAnywhere Classes: The Symantec dbAnywhere classes provide grid access to databases accessed when using Symantec's dbAnywhere software.
- Utility Classes: Objective Grid uses many Java extension classes for internal utility classes. The uses of the utility classes include tabbed windows, the undo/redo architecture, etc. The tabbed panel class (see Figure 2) showcases a row of tab buttons with different Panels associated with each "tab." They can be initialized to be placed at the top of the panel or at the bottom of the panel, allowing a variety of uses.



0	1	2	3	4
1				
2				
3				
4				
5				
6				
7				
8				
9				
10				
11				
12				

Figure 3: The FirstGrid example shows the developer the basics of using Objective Grid.

Objective Grid/J also includes an Objective Grid Guide and a FAQ in Microsoft Word format. In the Guide are tutorials that take you through each of the five included examples:

- FirstGrid (see Figure 3)
- GridApp
- GxdbAnyWhereSample
- GXQuery
- Print_Preview

When you are ready to examine the examples, you must build the complete OG/J package. OG/J comes with several pre-built packages and you have to set your CLASSPATH environment variable to include them.

Also included in the Guide is an Objective Grid and Objective Blend Reference that covers all the classes in detail. Additional chapters of the Guide show the programmer how to add their own controls, use the Undo and Redo feathers, and how to decrease the bytecode size of your applets and applications.

Objective Blend is a package of user interface components that OG/J utilizes. The Objective Blend package is included in OG/J as a runtime package; you can build your OG/J package with this package. It comes as a single .zip file which is included in the lib directory.

Objective Grid enables even the novice programmer to create full-featured spreadsheet applications. The majority of the functionality of Objective Grid is automatically provided directly by the package. Programmers who wish further customization can create classes from the base classes provided and extend their functionality even more. This is the type of functionality that programmers consume hours developing and even more hours debugging. Products like Objective Grid can enable an average programmer to turn out above average applets and applications. I'd rate this product as a must-have for serious programmers.

About the Author

Edward Zebrowski is a technical writer based in the Orlando, FL area. Ed runs his own Web development company, ZebraWeb, and can be reached on the net at zebra@rock-n-roll.co

zebra@rock-n-roll.co

DCI

ADVERTISER INDEX

Advertiser		Page
3D Graphics		27
www.threedgraphics.com	310 553-3313	
Bristol Technology		33
www.bristol.com	203 438-6969	
Coriolis		37
www.coriolis.com	800 410-0192	
DCI		41
www.DCIexpo.com/Internet	508 470-3880	
Greenbrier & Russel		19
www.gr.com/java	800 453-0347	
IAD		55
www.IADconf.com	508 652-1010	
ILOG		49
www.ILog.com	800 367-4564	
Imperial		77
www.Imperial.com	415 688-0200	
Installshield		15
www.installshield.com	800 269-5216	
Intuitive		50
www.intuitivesystems.com	408 245-8540	
JavaOne		35
http://java.sun.com	800 668-2741	
JavaWorld		67
www.javaworld.com	415 267-4527	

Advertiser		Page
KL Group Inc.		68
www.klg.com	800 663-4723	
Live Software		25
www.livesoftware.com		
Marimba		60
www.marimba.com/download	415 328-JAVA	
MindQ		29
www.mindq.com	800 847-0904	
Net-Developer '98		45
www.net-developer.com	612 368-7227	
Net Guru		42
www.ngt.com	800 know.ngt	
Net Guru		54
www.ngt.com	800 know.ngt	
Object Expo		61
www.sigs.com	212 242-7515	
Object Matter		42
www.objectmatter.com	305 718-9109	
Phaos Technology Corporation		21
www.Phaos.com	212 229-1450	
ProtoView		3
www.protoview.com	609 655-5000	
Roguewave		11
www.roguewave.com	800 487-3217	

Advertiser		Page
Sales Vision		23
www.salesvision.com	704 567-9111	
SofTech Computer Systems		29
www.scscompany.com	814 696-3715	
Stingray Software Inc.		2
www.stingsoft.com	800 924-4223	
SunTest		4
www.suntest.com	415 336-2005	
Sybex Books		39
www.sybex.com	510 523-8233	
Symantec		57
cafe.symantec.com	800 453-1059 ext. 9NE5	
SYS-CON Publications		27
www.sys.con.com	914 735-1900	
SYS-CON Publications		59
www.sys.con.com	914 735-1900	
SYS-CON Publications		62
www.sys.con.com	914 735-1900	
SYS-CON Publications		63
www.sys.con.com	914 735-1900	
Thought, Inc.		13
www.thought.com	415 836-9199	
Zero G Software		54
www.zerog.com	415 512-7771	

Net Guru 1/4 Ad

Object Matter 1/4 Ad

JavaBeans™ for the Service-Driven Network™

by Dave Hendricks

We are moving towards a world in which you can expect instant access to online shopping from a phone; your kids can always reach you with one number that bounces from your phone, to your pager, mail or car; and you pay one low price each month to your local telco provider that brings you combined fax, Internet, local, long distance and cellular services.

To help providers keep up with these demands, Sun recently announced the Service-Driven Network™, a model that helps telecommunications providers rapidly develop and deploy innovative network services, and manage the networks over which they deliver these services.

A key component of the Service-Driven Network is the Java Dynamic Management Kit, a combination of "push" and Java network management technologies for building self-managed networks. It is the first Java-based toolkit for building and distributing network management intelligence into system, application and network devices.

The Java Dynamic Management Kit is a Java agent toolkit for creating JavaBeans which allows rapid development of autonomous Java agents and will help telecommunications providers develop smart, autonomous Java agents that can be distributed throughout the network to act as invisible "assistants," alerting network managers of potential problems or fixing them on their own.

At the Heart of it All: JavaBeans for Management

The Java Dynamic Management Kit provides a library of core management services, implemented as JavaBeans components, called JavaBeans for Management. Developers can create their own JavaBeans for Management, which can be dynamically distributed to the network. They can be slotted in and out of an agent, allowing one to add, modify or cancel services, just as hardware elements can be slotted in and out of a rack.

Each JavaBean for Management has a set of properties, can perform a set of actions and can emit a set of notifications, all defined through standard JavaBeans design patterns. For example, a read-write property is defined for property "Foo" if the JavaBean for Management contains two methods with the following signatures:

```
public PropertyType getFoo();
public void setFoo(PropertyType value);
```

A JavaBean for Management is manageable as soon as it is registered with the agent's Core Management Framework (CMF). JavaBeans for Management can be registered and deleted from the CMF dynamically, which allows transient services to be implemented that can be pushed into the agent and then disappear once they have finished.

The representation of JavaBeans for Management as JavaBeans components enables the developer to access them using a JavaBeans application builder, such as Java Workshop or Java Studio. JavaBeans in an existing Java application can be made

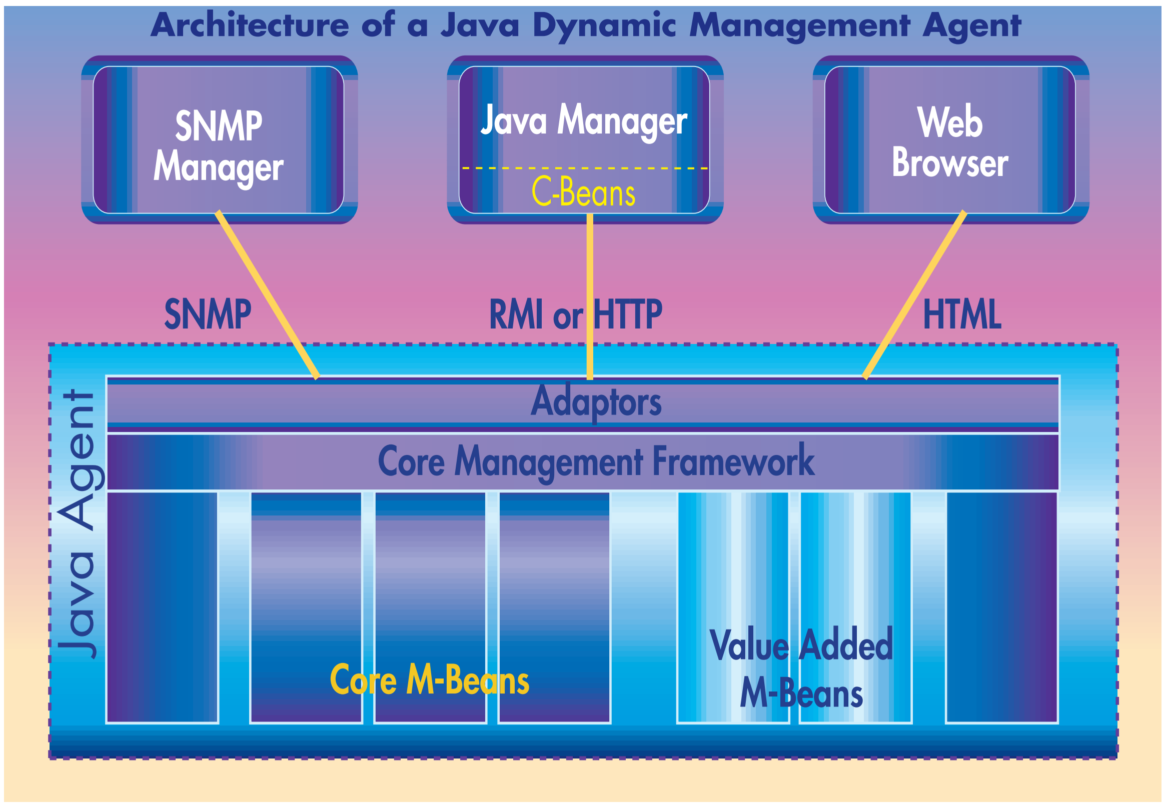
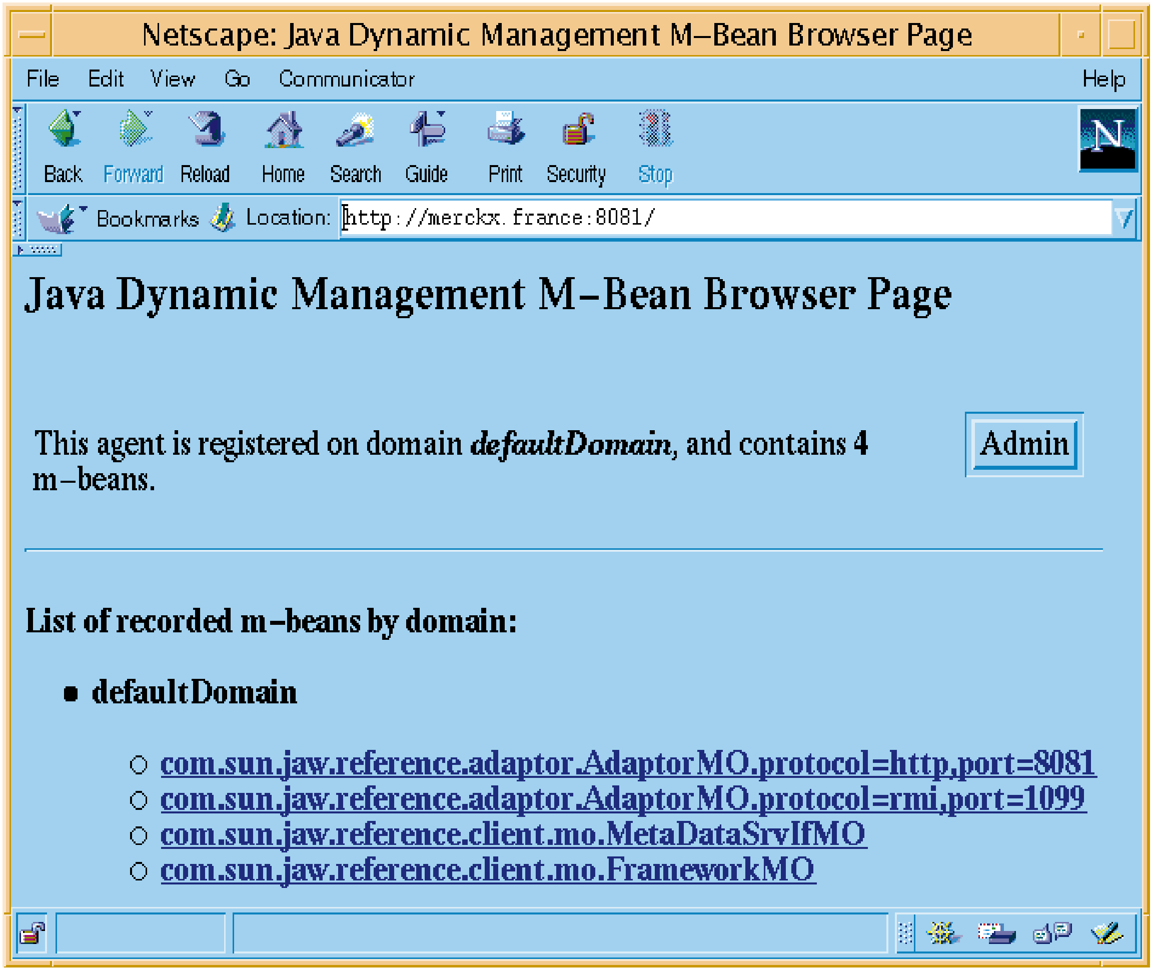


Figure 1: A Java Dynamic Management agent consists of these components, all running within a Java Virtual Machine (JVM): JavaBeans for Management, Core Management Framework (CMF), Adaptors for different protocols.

manageable simply by registering them with the CMF, with no change to the existing application. Note that the Java Dynamic Management Kit does not force a Java class inheritance scheme on the developer; the developers are free to develop JavaBeans as they like.

A number of core JavaBeans for Management which implement generic management services are provided in the Java Dynamic Management Kit. Only the core JavaBeans for Management needed by an agent must be registered with the CMF. The generic services provided include an object repository, basic notifications, filtering, monitoring, dynamic class loading, dynamic native library loading, relationships and MLET.

The MLET service provides a way for an agent to find JavaBeans for Management on the network, pull them into the agent and start executing them.


Netscape: Java Dynamic Management M–Bean Browser Page
File Edit View Go Communicator Help
Back Forward Reload Home Search Guide Print Security Stop
Bookmarks Location: http://merckx.france:8081/
Admin

Java Dynamic Management M–Bean Browser Page

This agent is registered on domain ***defaultDomain***, and contains **4** m–beans.

List of recorded m–beans by domain:

- **defaultDomain**
  - com.sun.jaw.reference.adaptor.AdaptorMO.protocol=http,port=8081
  - com.sun.jaw.reference.adaptor.AdaptorMO.protocol=rmi,port=1099
  - com.sun.jaw.reference.client.mo.MetaDataSrvIfMO
  - com.sun.jaw.reference.client.mo.FrameworkMO

Figure 2: Java Dynamic Management Kit agent's home page seen through a Web browser

Core Management Framework

The Core Management Framework (CMF) controls the JavaBeans for Management in an agent. Whenever an agent is requested to perform a management operation, the CMF calls the appropriate JavaBean for Management to perform the requested operation. A JavaBean for Management can query the CMF to obtain information on other JavaBeans for Management present in the CMF.

Adaptors

The JavaBeans for Management and the CMF are protocol-independent. This property frees the developer from having to work with communication protocols. For an agent to be manageable, it must include at least one adaptor which provides access to the JavaBeans for Management within the agent through a specific protocol. Adaptors are provided for HTML, HTTP, RMI and SNMP. An agent can include more than one adaptor, allowing a single agent to communicate via different protocols to different management applications.

The HTML adaptor includes a small "Web server" which provides an HTML view of the JavaBeans for Management within an agent. This allows the agent to be managed directly by a standard Web browser from which the user can see the JavaBeans for Management within an agent and perform operations on them. HTML pages are generated dynamically to show the requested M-Bean information. So, with no development required, the user has a manager application which can query their Java Dynamic Management Kit agent.

Service Creation Tools

The Java Dynamic Management Kit also includes some development tools. Mogen generates client stubs for JavaBeans for Management. These stubs are called C-Beans, or "Client Beans". C-Beans help developers build Java management applications that can talk to an agent through any protocol (RMI and HTTP are currently supported). The application developer manipulates the C-Beans as if they were local JavaBeans and the Java Dynamic Management Kit assures the communication with the agent.

An SNMP MIB compiler is provided. This takes an SNMP MIB as input and generates JavaBeans for Management that can be plugged into an agent. In essence, the Java Dynamic Management Kit provides a complete toolkit for the developer who wants to develop SNMP agents in Java. The developer can also put the HTML adaptor into their SNMP agent and browse the agent with a Web browser.

A Java Dynamic Management Kit agent can be thought of as a container to execute mobile code. If this container is distributed among the devices, systems and applications in a network, one can dynamically push intelligent services throughout the network and the Service-Driven Network is enabled.

For more information on the Java Dynamic Management Kit, see http://www.sun.com/software/java-dynamic/

About the Author

Dave Hendricks is the engineering manager of the TMN and Java Management Products group at Sun Microsystems' development site in Grenoble, France. Dave has been with Sun for 11 years and was one of the founding members of the Grenoble development site when it was created in 1990. Dave joined Sun after receiving BS and MS degrees in Computer Science from Stanford. At Sun, he initially worked on source code management tools before moving to Grenoble and working on X.25 and then network management.

Net
Developer
full

Java Reference Library
Java in a Nutshell, Deluxe Edition
by O'Reilly & Associates, Inc.

For seasoned code jockeys, but even beginners are singing its praises

by **Ed Zebrowski**

If you are a Java developer, you are no doubt familiar with O'Reilly and Associates and their line of books. One of the first Java books on the market was O'Reilly's "Java in a Nutshell". Although there had been a couple of "how-to" books published, there were no Java reference books available for the professional programmer. The Nutshell series is not for beginners, but rather for seasoned code-jockeys who will dog-ear the book in no time flat. But the funny thing was, even beginners started to sing the praises of the Nutshell book.

O'Reilly has taken the "Java in a Nutshell" book and made something even more valuable out of it for Java developers. The new Deluxe edition not only contains the complete printed book, but it also contains five additional books in CD-ROM format:

- Java in a Nutshell
- Exploring Java
- Java Language Reference
- Java Fundamental Classes Reference
- Java AWT Reference

The library also comes with a one-year subscription to the online version of the Java Reference Library, which may then be accessed throughout the year. After logging into the O'Reilly Web site – which requires you to go through a free registration process – you must log in further using the password that is provided on a card inside the book. In this manner, the books may be accessed online and the very latest material may always be accessed. The books may also be searched so finding information about a specific topic is a snap.

"Java in a Nutshell"

Part I: Introducing Java

Since the Java Reference Library is based on the "Java in a Nutshell" book, we'll take a detailed look at it here. Although it is not a book for beginners, it does have a couple of chapters to get C and C++ programmers up to speed. The first chapter, Getting Started With Java, is written for programmers who have been living in a cave – or so it seems. It covers exactly what Java is and where it came from; the usual introductory material. Most Java programmers will probably skip this chapter. Chapter 2, How Java Differs From C, and Chapter 3, Classes and Objects in Java, are chapters that C++ programmers will probably devour before moving on to the meat of the book.

Part II: Introducing Java 1.1

Chapter 4, What's New in Java 1.1, goes into a detailed discussion of the 23 packages that make up the core API for Java 1.1, explaining the packages and the changes which have been made. Chapter 5, Inner Classes and Other New Language Features explores how the programmer is to define and utilize each of the four new types of classes. These two chapters get the programmer up to date on the changes made in the latest release. The book approaches some of the problems that the programmer is likely to notice upon compilation and describes how to fix them.

Part III: Programming with the Java 1.1 API

Chapters 6 through 12 discuss the programming examples, which use the new features of Java 1.1. The examples are yours to do with as you wish and programmers are encouraged to adapt them for their own programs. The applets and applications themselves may be downloaded from the Web site as well. If you'd like to be able to read the source for these example programs, you'd better purchase this edition as author David Flanagan has indicated that he will probably be taking the example section out altogether in the next release of the book simply for space (and cost) reasons. He feels that the book is already too large and says that he may even produce another book specifically on the Java Enterprise APIs which will go into database connectivity, remote method invocation and the security features of Java 1.1.

Java Reference Library
Java in a Nutshell
O'Reilly & Associates, Inc.
101 Morris Street
Sebastopol, CA 95472
Phone: 800 998-9938
Fax: 707 829-0104
Web: http://www.ora.com
Email: info@ora.com
Platforms: Any platform for which there is a JVM
Price: $69.95 each

Part IV: Java Language Reference

The chapters which make up Part 4 focus on the very basic information you'll need to start coding and include basic reference material about the Java programming language. Chapter 13 contains summary tables of Java syntax. The rest of Part 4 is stuff that most programmers will already have down pat – Java system properties, HTML code for including applets on a Web page, compiler options. Always good to have around in case you wake up one morning after an all night code session and can't remember how to compile your source code.

Part V: API Quick Reference

This section is the core of this book and takes up the majority of the pages. It provides the programmer with a quick-reference guide to the Java API and goes into each of the packages in detail. Included are chapters on:

- The java.applet Package
- The java.awt Package
- The java.awt.datatransfer Package
- The java.awt.event Package
- The java.awt.image Package
- The java.awt.peer Package
- The java.beans Package
- The java.io Package
- The java.lang Package
- The java.lang.reflect Package
- The java.math Package
- The java.net Package

- The java.text Package
- The java.util Package
- The java.util.zip Package

Flanagan's writing style is both technical and personal at the same time; he introduces the packages by making the reader familiar with the basic features, as he does in this excerpt from Chapter 21:

"The java.awt.image package is, by any standard, a confusing one. The purpose of the package is to support image processing, and the classes in the package provide a powerful infrastructure for that purpose. Most of the classes are part of the infrastructure, however, and are not normally used by ordinary applications that have only simple image manipulation requirements.

To understand this package, it is first important to note that the Image class itself is part of the java.awt package, not the java.awt.image package. Furthermore, the java.awt.image classes are not the source of images; they simply serve to manipulate images that come from somewhere else. The Applet.getImage() method is perhaps the most common method for obtaining an image in Java – it downloads the image from a specified URL. In a stand-alone application, the URL.getContent() method can be used to obtain an ImageProducer object, which can then be passed to the Component.createImage() method to obtain an Image object."

One of the great things about the Deluxe Edition of "Java in a Nutshell" is that not only can you access the entire text and examples from the book online, but you can read the printed version while you're sipping your morning coffee. "Java in a Nutshell" is the best choice O'Reilly could have made for a printed reference book to go with the CD version of the other four books. The first of these online books, "Exploring Java", is an introductory book about the fundamentals of the Java programming language.

"Exploring Java"

This book, written by Patrick Niemeyer and Joshua Peck, breaks down the Java language, its class libraries, programming techniques and idioms into a readable, easy to understand format. It also provides realistic, though simplistic, examples that hint at what can be done with Java. Chapters include:

- Yet Another Language?
- A First Applet
- Tools of the Trade
- The Java Language
- Objects in Java
- Threads
- Basic Utility Classes
- Input/Output Facilities
- Network Programming
- Understand the Abstract Windowing Toolkit
- Using and Creating GUI Components
- Layout Managers
- Drawing With the AWT
- Working With Images

Chapter 3, for example, is about the tools which you need to be familiar with in order to program in Java. It covers the Java interpreter and discusses the manner in which it works, and also goes into how you'll need to set up your class path and how to use the JDK compiler. It also gives into a detailed explanation of how to compile Java source code using Netscape's own interpreter and how to include applets in your own web pages, along with all the possible parameters of such.

"Java Language Reference"

"Java Language Reference", written by Mark Grand, is a serious reference guide for Java programmers. It goes into detailed explanations of how the Java language works in particular situations. The book does cover some basic stuff, such as how to compile or run an applet, etc. It becomes a bit redundant when it's combined with the rest of this set. You'll probably find yourself referring to this book but it won't be to find out how to compile your first applet, or how to add an applet to a Web page. It's more likely to be when you are trying to find the answers to specific questions, such as how Java selects the method that's invoked by method call expressions, or how the multiplication operator works when using floating-point data.

"Java Language Reference" covers all aspects of the Java language and includes small examples where appropriate. It describes the syntax for all Java statements, exception handing, multithreaded programming and also contains reference material on the classes in the java.lang package. Consider this a backup for when you want even more material than you can instantly find in the "Java in a Nutshell" book.

"Java Fundamental Classes Reference"

"Java Fundamental Classes Reference", also written by Mark Grand and co-author Jonathan Knudsen, is the companion book for the Java Language Reference. It is a reference for the fundamental, or core, classes of the Java 1.1 API. That includes all the classes in the JDK that programmers are likely to need (not including the AWT, which is covered in a separate book, the Java AWT Reference). The following packages are covered, first with a detailed description of the class as a whole, then with a complete description of every variable, constructor and method which is defined by the class:

- java.io Package
- java.lang Package
- java.lang.reflect Package
- java.math Package
- java.net Package
- java.text Package
- java.util Package
- java.util.zip Package

Other topics include strings and related classes, threads, exception handling, I/O, networking and security. These are discussed in a tutorial fashion and include examples that show how to use these features. Since the entire text is searchable, it makes it very easy to locate specific information about a core Java class in a hurry.

"Java AWT Reference"

The "Java AWT (Abstract Window Toolkit) Reference", by John Zukowski, is a book you'll want to become familiar with, at least if you care about what your applet or application looks like. The AWT provides the application with its graphical user interface (GUI). This book covers the Java 1.0.2 to Java 1.1 AW, and discusses the differences between them, including the many changes which have been made to the method names. Again, as with the other books in this set, examples are given as appropriate. Among other things, the book covers basic graphics, fonts and colors, events, components and containers, image processing, errors, data transfer and printing.

"Java AWT Reference" also has an appendix section which may be very useful – especially the Platform-Specific Event Handling appendix. An appendix to image loading, which covers the hidden classes in the sun.awt.image package, is also included.

Conclusion

While owning the CD-ROM version of these books isn't quite the same as owning the printed copies, it's in many ways better. You are able to search the entire book for specific words and can have instant access to the specific information you are looking for. Also, the most up-to-date versions of the books are available online, so they never go out of date. "Java in a Nutshell' is by and large the best quick reference for Java you'll find. By combining it with the other four books which make up the Java Reference Library, O'Reilly has developed a reference set which will be well used by the serious Java programmer.

About the Author

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

zebra@rock-n-roll.com

Reinventing TCP Based Internet Protocols in CORBA

CORBA offers a solution to remedy TCP's inherent shortcomings in an elegant manner

by Kim Lau

This article proposes to reinvent select TCP based application layer Internet protocols and their client/server implementations in the framework of CORBA/IIOP [2]. Advantages of this approach will be exhibited from the perspectives of programming, deployment and protocol evolution. As an illustrative example, I will attempt a redefinition of the IRC protocol [1] in terms of OMG IDL (Interface Definition Language). I will implement its server object and client application in Java with Javasoft's JavaIDL [3] as the underlying ORB, paying attention to implementation issues different from those of its socket counterpart. I will then briefly go over various benefits the CORBA framework can bring to the deployment of the reinvented IRC 'protocol', with emphasis on the seamless incorporation of load balancing, security and transaction processing control.

Overview

CORBA has recently become the most talked-about distributed object framework for building client/server applications. However, few people have envisioned leveraging CORBA/IIOP on legacy client/server applications based on application layer Internet protocols such as SMTP, FTP, NNTP, IMAP, IRC, etc.

The unprecedented growth of the Internet as witnessed in recent years is mostly fueled by the widespread use of various Internet services, notable examples being e-mail and the World Wide Web. The underlying technology behind these Internet offerings is based on a client/server model with various application layer Internet protocols as the communication foundation. For years, these Internet protocols have been defined at the level of TCP socket streams.

The TCP socket framework, however, bears various inherent shortcomings. It turns out that CORBA offers a solution to remedy these shortcomings in an elegant manner.

The Socket Model

A TCP application layer Internet protocol engages two computer processes linked by a single or multiple (e.g., FTP) TCP connection(s), allowing one process (the client) to request services of a certain kind from the other (the server). The protocol governs the dialogs the client exchanges with the server over their TCP connection. The rule for the dialogs is usually expressed in Backus Naur Form (BNF), and serves as a contractual agreement for communication between the client and server implementations.

A TCP socket stream is a primitive communication infrastructure. Other than guaranteeing delivery of a sequence of bytes from one end of a TCP connection to the other in the same sequential order, it essentially offers no additional services. How a data object is marshaled for transportation over the wire is left as an exercise, often a tiresome one, to the design and implementation of the Internet protocol.

"Great care... must be taken in the design of the protocol upgrade to ensure that clients and servers of different versions are interoperable."

This month's article, by Kim Lau, presents an interesting and perhaps unorthodox view of CORBA as a tool. In essence, Mr. Lau argues that we can view CORBA as an automatic protocol generator, designing an Internet-based protocol based on a CORBA IDL description of an interface. And what interfaces does Mr. Lau propose to generate protocols for? Why, the very interfaces that are "implied" by the protocols that pervade today's Internet! Rather than spend time at the difficult-to-upgrade bit level, Mr. Lau argues that new (and upgraded) Internet services ought to be defined at a portable interface level, leaving CORBA to automatically generate protocols. A thought-provoking approach indeed, and one that has been discussed many times within the OMG process. Enjoy this concrete approach to the issue.

Richard Soley
Editor, CORBACORNER
President and Technical Director of the Object Management Group, Inc.

Real world deployment of an Internet protocol usually demands certain additional common functionality not related to the core application logic. For instance, a farm of Web servers is usually deployed to host a heavily hit Web site, giving rise to the need of uniformly brokering incoming HTTP requests among the HTTP servers, and for balancing their loads. E-commerce applications often necessitate encryption of network traffic, for instance, to prevent sensitive data from being eavesdropped.

Within the socket model, such additional requirements can only be tackled with ad hoc approaches. Load balancing of Web servers with identical content, for instance, is usually achieved through 'Round Robin DNS', in which the IP addresses of all participating Web servers are statically mapped to the hostname of the Web site. This ad hoc approach works because such HTTP servers seldom share state information among one another (cf. statelessness of

New House ad
full

HTTP), except possibly through a single backend DBMS. However, the Round Robin DNS approach is not robust, since the DNS server process has no way to find out the load of each participating Web server. It does not even have knowledge of whether a Web server is out of service or not. It is therefore not uncommon to come across popular Web pages with a certain percentage of broken in-line images, especially if the Web site is served from a lesser computing platform.

From time to time, an Internet protocol is due for upgrade to meet previously unanticipated demands, or to improve protocol performance. With a worldwide installed base, it is next to impossible to depreciate and retire implementations of the clients and servers conforming to an older version. Great care, therefore, must be taken in the design of the protocol upgrade to ensure that clients and servers of different versions are interoperable. This compatibility requirement imposes severe constraints in the design of the upgrade.

To summarize, the shortcomings of the socket model include:

- Application developers need to marshal structured data for communication over TCP streams. The data marshaling and unmarshaling complicates the coding effort, introducing unnecessary digression on software developers from the main application logic.
- Additional effort is required for load balancing, security and others. Usually ad hoc approaches result.
- Upgrade of an Internet protocol is constrained by compatibility baggage.

The CORBA Framework

CORBA (Common Object Request Broker Architecture) [2] is a specification for creating and using distributed objects in a platform-independent and language-neutral manner. CORBA specification is written and maintained by the Object Management Group (OMG) [4], an industry consortium of all major software vendors with the exception of Microsoft.

Within the CORBA framework, the type of object is described by OMG IDL (Interface Definition Language). The IDL definition of an object publicly exposes its services to its potential clients. Well-defined mappings have been established for translating IDL definitions into common programming languages. As of this writing, the Java language mapping has just been approved by the OMG.

Besides allowing distributed objects to interoperate, CORBA defines a rich set of preexisting services and facilities. A CORBA object can seamlessly incorporate such commonly demanded functionality as transaction processing control and distributed object security.

In addition to CORBA, there are other competing architectures available including RMI (Remote Method Invocation), and Microsoft's DCOM (Distributed Component Object Model). RMI is closely tied with Java, allowing a client written in Java to invoke methods carried by a remote Java object. DCOM is the answer from Microsoft for communications among distributed objects living in the Win32™ world. Since the Internet is a heterogeneous network linking up computers in diverse platforms, CORBA is currently the only architecture well suited for handling distributed objects over the Internet.

"...unlike HTTP, which in essence follows a simple request/response cycle, IRC is sufficiently complicated and feature rich to merit a CORBA approach."

The recipe for putting a legacy Internet protocol into the CORBA framework is:

1. Redefine the protocol in terms of OMG IDL.
2. Construct server objects on your targeted platform by implementing various interfaces defined in the protocol IDL.
3. Construct clients which interact with a server object by invoking its methods as defined in the protocol IDL.

Protocol upgrade is achieved through IDL inheritance. The upgraded server object will implement the inherited interfaces, which includes added attributes and methods pertaining to the newer protocol version. Appealing to the CORBA dynamic invocation interface, an upgraded client can determine at run-time which interfaces the server object exposes, thus ensuring interoperability with a server of older protocol version.

The advantages of the CORBA approach are:

- Communications between client and server happen as method calls, hiding programmers from explicit data marshaling.
- A rich set of preexisting CORBA object services and CORBA facilities can be utilized to seamlessly incorporate such commonly demanded functionality as load balancing, security and transaction processing monitoring.
- Protocol upgrade is achieved through IDL inheritance. Clients of newer protocol versions can discover at runtime whether the server supports the newer interface. Clients and servers of different protocol versions are thus automatically interoperable. The design of the newer protocol version is therefore not constrained by compatibility baggage.

Example: Internet Relay Chat

To illustrate the various ideas introduced here, I will attempt to recast the IRC protocol [1] into the CORBA framework. IRC is a text-based protocol allowing teleconferencing among connecting clients in a chat room (channel). An IRC server can hold multiple channels, each of which is comprised of a group of participating users. A user's level of participation in a channel can be characterized by the user mode. A user of the SPECTATOR mode can observe the chat in progress but cannot broadcast a message to all channel users. A user of the PARTICIPANT mode has the capability to broadcast a message to all channel users. A user of the HOST mode is a PARTICIPANT with the added privilege of changing the modes of other channel users, as well as various channel properties.

From a functional point of view, the IRC protocol can be categorized into 2 parts:

- Protocol governing interaction between an IRC server and its clients
- Protocol governing interaction among IRC servers for content replication

My attempt at IRC redefinition will focus on the first part here. If there is sufficient interest from the Internet community, I will exert future effort on the second part to complete the whole CORBA-tization process.

Why pick IRC for an exercise of CORBA-tization? Why not HTTP, for instance, which is better known among Internet users? The reasoning is threefold. First, unlike HTTP, which in essence follows a simple request/response cycle, IRC is sufficiently complicated and feature rich to merit a CORBA approach. Second, unlike HTTP, IRC requires its server object to maintain certain state information for each client connection; namely, what channels (chat rooms) the client has joined and what user mode the client carries for each subscribed channel. It would be less trivial to CORBA-tize IRC than the stateless HTTP. Third, IRC is one of a few Internet protocols with asynchronous elements. An IRC client can, at any time, receive channel messages from other clients, as well as notification of a new user joining a subscribed channel, for instance. Such asynchronous elements present an opportunity to illustrate call back methods within the CORBA framework.

Listing 1 exhibits my attempt to redefine IRC in terms of OMG IDL. The Ircd interface describes a CORBA object a client will interact with in the beginning, to inquire about available channels and to log in. Upon a login invocation from a client, the server process will create a CORBA object implementing the UserSession interface, and pass a reference of that UserSession object back to the client. From the server's perspective, a UserSession object tracks various state information of its corresponding client. The client invokes methods of its UserSession object to perform various IRC activities.

The UserNotifier interface describes a CORBA call back object residing at an IRC client for receiving asynchronous events from the server. A UserNotifier object needs to be created at the IRC client side, and passed to the IRC server at login. The interface defines methods for an IRC client to receive channel message posting, notifications of user arrival and departure, notification of a user mode change and server pings.

The IRC server and client objects are then implemented in Java with JavaIDL as the underlying ORB. Without going into coding details, I will merely point out that porting the implementation for another Java ORB should be relatively easy.

Since Java does not allow parameter passing by reference, the CORBA language mapping stipulates that passing an inout or out parameter requires the use of the Holder class. The UserNotifier call back object needs to be passed as an inout parameter. Passing it as an input parameter will trigger a CORBA exception for unknown reasons. It is probably a bug of the JavaIDL early release.

Due to its length, the complete Java source code for both the IRC server and client implementations will not be listed here. Interested readers may download it along with setup information from http://www.unique.net/~lau/CORBAirc/

Epilogue

Currently, the use of CORBA is mostly confined to the construction of n-tier client/server business applications. The forthcoming object Webs may bring CORBA closer to the mass. Much has been said in this article on the viability of leveraging CORBA on legacy Internet protocols. The Redmond school of thought has always proclaimed the installed base of COM/DCOM is an order of magnitude larger than that of CORBA. This claim may not be able to hold if CORBA-based Internet applications, such as IMAP-like e-mail clients/servers and NNTP-like Usenet news readers/servers are in widespread use.

References

1. J. Oikarinen, D. Reed. Network Working Group RFC 1459 (Internet Relay Chat)
2. CORBA/IIOP 2.1 Specification, http://www.omg.org/corba/corbiiop.htm
3. JavaIDL, http://www.javasoft.com/products/jdk/idl/
4. Object Management Group, http://www.omg.org/

About the Author

Kim M. Lau is a software engineer at Warner Bros. online. He holds a Ph.D. in Physics from UCLA. Kim can be reached at: lau@unique.net

lau@unique.net

Listing 1.

```
Redefinition of IRC in OMG IDL

module CORBAirc {

exception NickCollision {};
exception NickNotFound {};
exception ChannelNotFound {};

typedef sequence<string> StringSeq;

enum UserMode { SPECTATOR, PARTICIPANT, HOST };

interface UserSession {
  readonly attribute string nick;
  readonly attribute string realname;

  boolean join (in string channel);
  boolean part (in string channel);
  StringSeq get_channel_users (in string channel);
  UserMode get_my_usermode (in string channel)
    raises (ChannelNotFound, NickNotFound);
  UserMode get_usermode (in string channel,
    in string nick
  ) raises (NickNotFound, ChannelNotFound);
  boolean set_moderated (in string channel,
    in boolean moderated);
  boolean set_topic_anyone (in string channel,
    in boolean STA);
  boolean set_topic (in string channel,
    in string newTopic);
  boolean set_usermode (in string channel,
    in string nick, in UserMode mode);
  void send_msg (in string channel,
    in string msg);
  void whisper (in string nick,
    in string msg);
  void quit();
};

interface UserNotifier {
  void msg (in string channel,
    in string sender, in string msg);
  void whisper (in string sender, in string msg);
  void new_user (in string channel, in string nick);
  void user_left (in string channel, in string nick);
  void usermode_changed (in string channel,
    in string nick, in UserMode newMode);
  void ping();
};

interface Ircd {
  UserSession login(in string nick,
    in string realName,
    inout UserNotifier notifier
  ) raises (NickCollision);

  StringSeq list_channels();
  string get_realname (in string nick)
    raises (NickNotFound);
  string get_topic (in string channel)
    raises (ChannelNotFound);
  void get_channel_info (in string channel,
    out boolean moderated,
    out boolean setTopicAnyone
  ) raises (ChannelNotFound);
};

};
```

The Data Series

Storing and reading data using the INI file format

by **Alan Williamson**

Data

You just can't get away from it. No matter what you do, a certain amount of data is always generated. One of the more profound quotes of the day can be attributed to Peter Large of Information Anxiety, where he once said:

"More Information has been produced in the last 30 years than in the previous 5,000. About 1,000 books are published internationally every day, and the total of all printed knowledge doubles every eight years."

A sobering thought. Data is to the developer what heat is to the physicist. They share the same dilemma; for every operation a small amount of heat or energy is produced; sometimes welcomed, sometimes not. Fortunately, the majority of data generated is purely cosmetic and does not require permanent storage. However, a certain amount of data does require archiving.

The main issue isn't the fact that this large quantity of data is being generated; quite the contrary. But how we organize this information in such a way that we can find what we want, when we want, is the much bigger issue. Get this right and the volume of data being handled becomes academic. The famous Sioux chieftain, Sitting Bull, remarked, back in 1876:

"'The white man knows how to make everything, but he does not know how to distribute it."

A man obviously much ahead of his time; a remark so fitting in today's mass data producing world. So, what has all this got to do with us, and more importantly, what has it to do with Java?

We are all responsible for organizing our data in a more structured manner. If it makes sense to us, then the chances that it makes sense to the next person are very high. Therefore due care and attention must be taken to all data being produced, by whatever means.

This article kicks off a mini-series of articles focusing purely on data storage and exploring some of those options open to you as a Java developer. To start with, we will look at storing simple pieces of information using simple text files. Next month, we will take a close look at the Object Serialization that was introduced in the 1.1 release of the JDK and the following month look at storing large amounts of data using a fully compliant JDBC database. Finally, we will round off this mini-series by looking at how you can use Symantec's implementation of dbAnywhere to store data.

So, let's begin.

Small Volumes

Data comes in many shapes and sizes, and sometimes, paradoxically, the smaller the data, the more hassle it is to handle. How many of us have broken the golden rule of never hard coding values into programs because we knew it was too much work just to save one stupid value, and by the time the program was developed, we were left with many so called 'stupid values' all hard coded and awkward to change.

Well, fear no longer. Let me introduce you to a class that will make the saving and retrieval of such 'stupid values' so easy that hard coded values will be a thing of the past.

Before we go into the implementation details of the class, let's do a quick overview of what features we would like to see from such a class. First, it has to be generic and be able to handle any number of parameters. Second, it must be able to read a pure ASCII file – that way, we will manually edit the values if necessary. Finally, it must allow the saving of data as well as reading.

A solution that has been in use on the Microsoft Windows platform for many years is the concept of an INI, or information file. This is where data is stored, on a line-by-line basis, as key/value pair. For example:

```
UserLogin=ceri
Password=moran
```

So let's take this system and use it as the basis for our new data handling class. The only difference between our implementation and the one employed in Windows is that ours will not support the notion of categories – in other words, we won't have any [ ] sections.

INI class

This class will be constructed in such a way that a calling process will be able to request a particular parameter, using the same name as appears in the main INI file. We can then identify at least one parameter, the filename of the INI file. Using this parameter we will be able to locate the file and begin reading it.

There are two approaches that may be taken at this point. The first one, and most obvious, is to open and parse the file every time a particular parameter is called for. The advantage of this system lies in the fact that no memory is used in storing all the parameters. However, if a lot of parameters exist that are continually being read, then a significant amount of time is lost opening and reopening the file each time.

A better solution is to bring all the parameters into memory in such a way that it allows for easy retrieval. The most obvious data structure for this task is the java.util.Hashtable. This structure allows us to store data, based on a unique key, which in this case is the name of the parameter (see Listing 1).

From the user's point of view there are only two public functions: one for reading parameters and the other to set parameters. The class is created by passing in the full path of the INI file, which throws an exception if something goes wrong.

As soon as the class is created, the INI file is opened and parsed, which can be seen in Listing 2. This is a simple matter of creating a new Hashtable instance and then reading the file, line-by-line. Assuming the line isn't empty, or a comment field (denoted by a hash (#) symbol), it is parsed into a key and data pair (see Listing 2).

Once retrieved, the value is inserted into the Hashtable. If for some reason the value already exists, then the subsequent insert is ignored. Notice the inner try/catch block. This is to catch any parsing problems that may be associated with a particular line. If this wasn't here, then as soon as a rogue line is met, the first try/catch block would catch the problem and reading of the file would stop. This would flag an exception at the creation of the class and the whole file would be deemed useless.

Once created, reading parameters is a simple matter of calling the appropriate methods from the Hashtable. Listing 3 illustrates the rather simple get( ) method.

If the Hashtable does indeed contain that particular parameter, then it is retrieved; otherwise, null is returned.

The majority of INI files are used purely for reading values, but there are times when writing a value back out to the INI is desirable. This is the reason for the set( ) method of the INI class. However, whereas the read method didn't incur any additional file access time, the writing method must. When a value is set, it must be written straight out to file, in case of a system crash or some other undetermined state that may prevent the class from closing naturally.

Listing 4 shows the setting of a new value. Note that if the parameter already exists, then it is first removed and then reinserted as the new value. Having set the correct parameter in the Hashtable, it must be written out to the file, which can be seen in Listing 5.

Writing the data out to file is a trivial matter of running through the Hashtable, and formatting the data in a <key>=<value> format.

Summary

This article began the mini-series on data and shows how you, the developer, can best treat it using Java. We looked at the easiest way of storing and reading data using the INI file format found in the Microsoft Windows platform. A fully working class was developed that can be used to read and write values to such a file, with extreme ease.

The next article in this series will look at the next level up, which is Object Serialization as introduced in the 1.1 of the JDK.

About the Author

Alan Williamson is on the Board of Directors at N-ARY Limited, a UK-based Java software company, specializing in Java/JDBC/Servlets. He has recently completed his second book, focusing purely on Java Servlets, with his first book looking at using Java/JDBC/Servlets to provide a very efficient database solution. Alan can be reached at alan@n-ary.com (http://www.n-ary.com) and he welcomes all

alan@n-ary.com

Listing 1: Storing data based on parameter.

```
public class ini extends java.lang.Object {

    private String  FileName = null;
    private Hashtable   data = null;

    public ini( String _filename ) throws Exception;

    public String get( String _parameter );
    public void   set( String _parameter, String _value );

    private boolean reloadData( BufferedReader InFile );
    private boolean writeData( PrintWriter OutFile);
}
```

Listing 2: Opening & parsing the INI file.

```
private boolean reloadData(BufferedReader InFile){
    String LineIn="";
    String key, value;
    int c1;
    data = new Hashtable();

    try{
        while ( (LineIn=InFile.readLine()) != null ){

            if ( LineIn.length() == 0 || LineIn.charAt(0) ==
'#')
                continue;

            try{
                c1  = LineIn.indexOf("=");
                key = LineIn.substring(0,c1).toLowerCase();
                value = LineIn.substring(c1+1,LineIn.length()
);
                data.put(key,value);
            }catch(Exception E){}
        }

        return true;
    }catch(Exception E){}
    return false;
}
```

Listing 3: Get ( ) method.

```
public String get( String _parameter ){

    if ( data.containsKey( _parameter ) )
        return (String)data.get(_parameter);
    else
        return null;
}
```

Listing 4: Setting a new value.

```
public void set( String _parameter, String _value ){

    if ( data.containsKey( _parameter ) ){
        data.remove( _parameter );
    }

    data.put(_parameter,_value);
    try{
        writeData( new PrintWriter(new BufferedWriter(new
FileWriter(FileName))) );
    }catch(Exception E){}
}
```

Listing 5: Writing the parameter to the file.

```
private boolean writeData(PrintWriter _Out){
    Enumeration E = data.keys();
    String key;
    while (E.hasMoreElements()){
        key = (String)E.nextElement();
        _Out.println( key + "=" + (String)data.get(key) );
    }
    _Out.flush();
    return true;
}
```

Listing 6: Complete listing.

```
import java.util.*;
import java.io.*;

public class ini extends Object {

    private String  FileName = null;
    private Hashtable   data = null;
```

```
      public ini( String _filename ) throws Exception
      {
          FileName = _filename;
          BufferedReader InFile = new BufferedReader(new Fil-
eReader(_filename));

          if ( reloadData( InFile ) == false )
              throw new Exception("File created an error:" +
FileName );
      }

      public String get( String _parameter ){

          if ( data.containsKey( _parameter ) )
              return (String)data.get(_parameter);
          else
              return null;
      }

      public void set( String _parameter, String _value ){

          if ( data.containsKey( _parameter ) ){
              data.remove( _parameter );
          }

          data.put(_parameter,_value);
          try{
             writeData( new PrintWriter(new BufferedWriter(new
FileWriter(FileName))) );
          }catch(Exception E){}
      }

      private boolean writeData(PrintWriter _Out)
      {
          Enumeration E = data.keys();
          String key;
          while (E.hasMoreElements()){
              key = (String)E.nextElement();
              _Out.println( key + "=" + (String)data.get(key) );
          }
          _Out.flush();
          return true;
      }

      private boolean reloadData(BufferedReader InFile)
      {
          String LineIn="";
          String key, value;
          int c1;
          data    = new Hashtable();

          try{
              while ( (LineIn=InFile.readLine()) != null ){

                  if ( LineIn.length() == 0 || LineIn.charAt(0)
== '#')

                     continue;

                  c1  = LineIn.indexOf("=");
                  key = LineIn.substring(0,c1).toLowerCase();
                  value = LineIn.substring(c1+1,LineIn.length()
);

                  data.put(key,value);
              }

              return true;
          }catch(Exception E){}
          return false;
      }
```

Net Guru
1/4 Ad

Imperial
1/4 Ad

IAD

An Object Pool Using Remote Method Invocation

A welcome addition to your object-oriented Java arsenal

by **Steven Schwell**

Implementation of a fixed size pool of Objects in a distributed application must consider problems caused by the unpredictable nature of remote connections. An implementation is presented here for Java's Remote Method Invocation, which takes advantage of the Distributed Garbage Collector to solve those problems.

You're probably familiar with the mechanism of a fixed size pool of Objects, in the context of a memory management system, that keeps memory in a fixed size buffer pool. The idea is to manage the use of a scarce resource by requiring Objects to be checked out of and into a pool. When all Objects are checked out, the pool is empty; clients cannot check out any more, until a claimed one is checked back in. When a client is done with an Object, it is expected to check the Object back into the pool so that other clients may claim it.

Aside from memory management, you could use an Object Pool of this sort to manage any scarce resource, or limit the number of concurrent users of a service or data structure. For example, you could use an ObjectPool as the basis of a simple license enforcement scheme which would limit the number of concurrent users of a service. As another example, you could use it to limit the number of concurrent visitors in a chat room application.

In a simple, non-remote Java applet or application, implementation of a fixed size ObjectPool would be fairly straightforward, as long as you can follow the rule that clients must check objects back into the pool when they are done with them. The ObjectPool needs only to keep a fixed size Vector of Objects, and parcel them out as requested until no more are left.

The requirement that Objects be returned to the pool when a client is done with them could be problematic in large applications where the end of an Object's usefulness could occur in many different places, far removed from where it was checked out of the pool. The problem is analogous to memory management in systems without garbage collection. It quickly becomes onerous to live up to your responsibility of checking the memory, or in this case, the Object from the ObjectPool, back in.

One idea is to do garbage collection on Objects from the ObjectPool, leveraging off of Java's own garbage collection mechanism. You could override the finalize() method of the Objects in the ObjectPool to check the Object back in when the Object is slated for garbage collection. In this way, the client needs only to remove references to the Object and rely on the local garbage collector to check the Object back into the pool.

The problem with that solution is that garbage collection in Java is, with good reason, not guaranteed to be timely. Garbage collection is expensive; the VM is free to do it only when necessary. Since garbage collection of Objects from the ObjectPool would rely on Java's garbage collection, it would be subject to the same vagaries. In many applications, the ObjectPool cannot wait indefinitely to reclaim its lost Objects. If your program does not use much memory, your unreferenced Objects may never get garbage collected, and the Objects would never get checked back into the ObjectPool.

An additional concern arises in a distributed application where the client may be in a different VM. What if contact with the client is broken before the client is able to check its Object back in to the server's pool? How can the pool reclaim that lost Object?

I present here an elegant solution to these problems for Java's Remote Method Invocation, which relies on RMI's Distributed Garbage Collector. In contrast to the local garbage collector, the DGC's behavior is well defined and reliable with respect to its timing. The DGC employs a lease mechanism for remote references. All remote references are leased to clients for a default period of ten minutes (which may be overridden by setting the java.rmi.dgc.leaseValue property). The client VM must request a new lease before the period runs out. Otherwise the server considers the remote reference to be dead and releases the corresponding remote Object to the local garbage collector for potential collection.

RMI also provides the java.rmi.server.Unreferenced interface, which remote Objects may implement to be notified when all remote references to the remote Object have been released. This includes the case when the last remote reference to a remote Object is released due to expiration of the lease. So taken together, Distributed Garbage Collection and the Unreferenced interface, give us just what we need to reclaim lost Objects from an ObjectPool.

Listing 1 shows an implementation of an ObjectPool that works for both remote clients and local clients. Even non-remote applications could benefit from this implementation because of its reliable garbage collection and reclamation of unreferenced Objects.

There are a few subtleties in the implementation that are worthy of elaboration. First of all, notice that remote Objects are created only as needed. This avoids the overhead of pre-allocating the Objects which may not all be needed every time the application is run.

Remote Objects are never released for local garbage collection; they are reused. This avoids the overhead of constantly creating new remote Objects every time an Object needs to be doled out. RMI calls the unreferenced() method every time the last remote reference to the remote Object is released; even multiple times on the same remote Object.

When an Object is checked back in,

notice that the corresponding Object is taken from the 'out' Vector rather than the returned Object itself. This is because the returned Object may be only the stub for the remote Object, rather than the remote Object itself. Vector's indexOf()method uses the equals() method to find the Object in the array. Since the PoolObject is extended from UnicastRemoteObject, it inherits the implementation of equals() from RemoteObject which considers stubs to be equal to their corresponding remote Objects. In this way, I'm guaranteed to be reusing the remote Object itself rather than just its stub.

One other note: The PoolObject inner class is a remote Object and, as such, needs to be post-compiled by RMIC to create its stub and skeleton classes. Running RMIC on inner classes is a bit tricky because the name of the inner class is generated by javac and includes the '$'character. The '$' character must be quoted to make it to the RMIC compiler. On some platforms, it must be quoted more than once. For example, on Solaris, the command line looks something like:

```
rmic ObjectPool\\\$PoolObject
```

Anticipated Patterns of Reuse

The PoolObject in the ObjectPool has no substance aside from the unreferenced() method. This is not a very interesting Object aside from its characteristics as an Object in a fixed size pool. Even so, in some circumstances, that may be all you need. If so, you can use the ObjectPool as is, or perhaps write an ObjectPoolIfc Remote interface so that ObjectPool can be accessed directly from RMI clients.

It is more likely, though, that you will need more interesting Objects to be in the fixed size pool. The implementation in Listing 1 is not just an example of how you could write your own ObjectPools. It is intended for you to reuse the class exactly as it is to create your own ObjectPools. It is probably worth elaborating on this as an illustration of general techniques for class reuse.

"Garbage collection is expensive; the VM is free to do it only when necessary."

In the design of any RMI system, you will be faced with the choice of passing Objects between VMs by value or by reference. Passing by value is accomplished by implementing the Serializable interface, and passing by reference is accomplished by implementing the Remote interface. A discussion of the tradeoffs in this design decision is beyond the scope of this article. For our purposes here, let's consider how to reuse ObjectPool for both cases, when the Object in the ObjectPool is to be passed by value and by reference.

If the Objects in your fixed size pool are to be remote Objects, i.e. passed by reference, you can subclass PoolObject to create that Object and subclass ObjectPool, overriding the newPoolObject() factory method to instantiate your PoolObject subclass. This is an example of the Template Method design pattern. The code would look something like Listing 2.

If the Objects in your fixed size pool are to be Serializable, i.e. passed by value, you can reuse ObjectPool by composition. The PoolObject remote reference will be passed to the client as a remote Object when the PoolObject is serialized. That contained remote reference in the Serializable PoolObject is not visible to clients of the PoolObject. It is there solely to enable collection of lost PoolObjects by the DGC. The code would look something like Listing 3.

Of course these are just some of the ways that ObjectPool and PoolObject can be reused. This is a maximally reusable and generally powerful little class that should be a welcome addition to your object-oriented Java arsenal.

References

Gamma, E., Johnson, R. & Vlissides, J. "*Design Patterns: Elements of Object-Oriented Architecture*" Addison-Wesley, Reading, MA, 1995.

Java Remote Method Invocation Specification, JavaSoft

About the Author

Steven Schwell is a Senior Developer and Java Guru in the New York office of Micromuse Inc., a leading provider of Service Level Management software. Steve is currently developing a number of large distributed Java apps. He holds an M.S. in Computer Science from Columbia University. Steve can be reached at Steven.Schwell@micromuse.com

Steven.Schwell@micromuse.com

Listing 1.

```
public class ObjectPool {
 protected int size;
 protected Vector in;
 protected Vector out;
 /* member class: */
 public class PoolObject extends UnicastRemoteObject
      implements Remote, Unreferenced {
  public PoolObject() throws RemoteException { }
  public void unreferenced() {
   try { checkIn(this);     // reclaim lost Object
   } catch (Exception e) {}
  }
 }

 /** constructor */
 public ObjectPool(int size) {
  this.size = size;
  this.in = new Vector(size);
  this.out = new Vector(size);
 }

 /** factory method to create new Object for the pool */
 protected PoolObject newPoolObject() throws RemoteException {
  return new PoolObject();
 }

 /** check out an Object from the pool */
 public synchronized Object checkOut() {
  Object o = null;
  if (out.size() < size) {
   if (in.isEmpty()) {
    try {
     o = newPoolObject();
    }
    catch (RemoteException e) {
    }
   }
   else {
    o = in.elementAt(0);
    in.removeElementAt(0);
```

```
    }
    out.addElement(o);
   }
   return o;
  }

  /** check an Object back into the pool */
  public synchronized void checkIn(Object o) {
   int x = out.indexOf(o);
    /* rely on Vector's use of equals() method
    ** to equate stubs with remote objects if necessary
    */
   if (x < 0) throw new NoSuchElementException();
   Object oo = out.elementAt(x);
    /* get the real Object, not a stub */
   out.removeElementAt(x);
   in.addElement(oo);
  }
 }
```

Listing 2.

```
public XXPool extends ObjectPool {
 class XXObject extends ObjectPool.PoolObject implements Remote {
  ... // instance variables and methods for your Object
  XXObject() throws RemoteException {}
 }
 public XXPool(int size) {
  super(size);
 }
 protected PoolObject newPoolObject() throws RemoteException {
  return new XXObject();
 }
}
```

Listing 3.

```
public XXPool {
 static public class XXObject implements Serializable {
  private Object poolObject;
  ... // instance variables and methods for your Object
  XXObject(Object poolObject) {
   this.poolObject = poolObject;
  }
 }
 ObjectPool objectPool;
 public XXPool(int size) {
  this.objectPool = new ObjectPool(size);
 }
 public XXObject checkOut() {
  Object o = objectPool.checkOut();
  if (o == null) return null;
  return new XXObject(o);
 }
 public void checkIn(XXObject o) {
  objectPool.checkIn(o.poolObject);
 }
}
```

SIGS

Hous
Spre

se Ad

ead

DeNova Introduces J'Express

(Vancouver, BC) - DeNova has introduced J'Express, a pure Java installer and distributor. Developers build cross-platform Java installations with the click of a button. Or, they can customize their installation with Java or external programs. Customers download and install Java apps as one smooth procedure from the Web.

Java programmers can build a complete Java installation through easy to navigate tabbed panels or write their own wizard panels which integrate into the installation. The install program may be customized to do anything they can do from Java, including calling native code and running other programs.

J'Express helps developers stop wasting time trying to track down the exact classes their Java app or applet uses. End users don't complain about missing classes, distribution files are more compact and there's less wasted disk space. It also builds zip files and directory trees and reliably uploads each version to the Web. Or, J'Express may be configured to launch an app at the end of the installation.

The list price of J'Express is $499. For more information, see DeNova's Web site at www.denova.com, call 408 490-2852 or e-mail sales@denova.com.

Symantec's Just-in-Time Java™ Compiler 3.0 Runs Java Applets and Applications Faster

(Cupertino, CA) - Symantec Corp., the leading provider of Internet development tools, has announced the availability of Version 3 of its Just-in-Time Java™ bytecode compiler (JIT). Released test scores show that this JIT outperforms any other JIT available – it is over 50% faster than Microsoft's IE4 JIT.

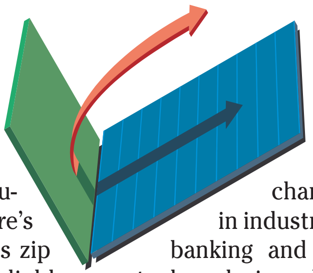
A JIT is a fundamental component of the Java Virtual Machine and boosts the execution speed of Java applets and applications by instantly converting Java bytecode to native code on the fly rather than by being interpreted as bytecode by the Java Virtual Machine. Symantec's JIT 3.0 is an integral part of their Visual Café for Java line of development tools and is integrated into Sun's Java Performance Runtime for Windows as well as into Netscape Communicator.

Symantec's JIT Version 3.0 is available now as an integral part of Visual Café for Java. Visual Café for Java Windows Version 2.1 customers should visit Symantec's update center to download a free upgrade from the Symantec Web site. The LiveUpdate feature will be available shortly. Future versions of Sun's Java Performance Runtime for Windows also will include the JIT 3.0.

For more information, please visit their Web site at www.cafe.symantec.com.

Rogue Wave's JChart 2.1 First to Offer Overlay Charts as JavaBean™ Components

(Corvallis, OR) - Rogue Wave Software has announced JChart 2.1, the first charting tool for Java™ to offer a palette of overlay charts built from JavaBeans™ components. Overlay charts, a composite of two or more chart types in a single chart, are commonly used in industries such as investment banking and securities to display stock analysis and financial information.



Also new with JChart 2.1 is the ability to mix and match JavaBeans components, combining them to create custom overlay charts. 39 JavaBeans charting components permit chart characteristics to be easily set and modified through individual property sheets.

JChart 2.1 includes JavaDocs, a User Guide, examples and online help. It is priced at $595 for a single use, source code license and is available immediately. JChart 2.1 supports Win95, Win NT 4.0 and Solaris. IDEs supported include Borland's JBuilder and IBM's VisualAge for Java. Browsers include versions of Netscape's Navigator and Microsoft's Internet Explorer that support the JDK 1.1.

For more information, see Rogue Wave's Web site at www.roguewave.com.

Phaos Creates Digital Certification Toolkit

(New York, NY) - Phaos Toolkit has introduced a toolkit that enables developers to easily add digital certificate technology into their Web applications. Dubbed the J/CA™ certification Toolkit, this package provides a turnkey certification solution for an array of applications – from electronic commerce to information technology – for functions such as access control, strong authentication, notarization and copyright protection.

The J/CA Certification Toolkit supports certificates based on the X509 version 3 standard. Applications built with the J/CA Toolkit can issue, parse, protect and validate certificates and interoperate with other certifying authorities. The J/CA Toolkit also includes technology for revoking certificates which have expired or are stolen.

When used with the Phaos SSLava™ Toolkit for secure transport, the J/CA Certification Toolkit gives developers a platform-independent, end-to-end solution for Internet security, including encryption, transport and authentication.

Phaos Technology delivers electronic commerce and security solutions for the Internet, including a complete suite of security toolkits for the Java platform. For more information, e-mail JCA@phaos.com or see their Web site at www.Phaos.com.

Rational Software Announces Visual Quantify for VB and Windows CE

(Las Vegas, NV) - Rational Software has announced the availability of Visual Quantify 4.0, a performance profiler that automatically pinpoints performance bottlenecks, extending the automatic performance profiling capabilities it has for Visual C++ and Java applications to Basic and Windows CE-based applications.

Visual Quantify 4.0 for Visual C++ for Visual C++, Java, Visual Basic and Windows CE is a scalable performance-profiling tool that helps developers create high-performance embedded and business enterprise software applications. It automatically identifies and pinpoints an application's performance bottlenecks and displays the performance data in an easy-to-read graphical format called the Quantify's Call Graph. This enables developers to focus on those parts of an application that will have the greatest impact on performance.

The patented Object Code Insertion (OCI) technology provides performance data for all parts of an applications, including components with or without source code. The PowerTune feature allows developers to specify, on a case-by-case basis, the amount of performance data to collect and the level of analysis to conduct. It also saves time by allowing developers to bypass the system-level data and go directly to the code through the annotated source.

Visual Quantify locates performance bottlenecks in applications built with

Microsoft VB, Visual C++, applications run using the Microsoft Java Virtual Machine and Visual C++ applications in Windows CE Emulation Mode on the Windows NT platform It requires an Intel, 486 or Pentium processor and supports Windows NT (versions 3.5-4.0), Visual Basic 5.0 and Visual C++ (versions 2.2 - 5.0) and is available now. US pricing is $748 per 1-year subscription (including upgrades and support for one year).

For more information, call 800 728-1212, e-mail info@rational.com or visit their Web site at www.rational.com.

Cosmo Software Brings Cosmo Code to Win 95/NT Platforms

(New York, NY) - Cosmo Software, a Silicon Graphics company, has announced the availability of Cosmo™ Code 2.5, an award-winning, professional Java™ development environment for the Windows® 95 and Windows NT® 4.0 platforms. Cosmo Code 2.5, which features integrated tools for developing, debugging and delivering applications, allows professional Java programmers to create media-rich applications.

Designed as an integrated set of visual tools for creating Java applications, applets and classes, Cosmo Code 2.5 offers a powerful and proven code base for easy and intuitive programming. The seamless integration between Cosmo Code's different components helps facilitate creation of sophisticated, cross-platform applications.

Cosmo Code 2.5 is shipping now and is competitively priced at $329. For more information, call 888 91-COSMO or visit their Web site at cosmo.sgi.com.

Sun and Visigenic Software Transition NEO Customers to Visigenic's ORB Technology

(San Mateo, CA) - Visigenic Software, Inc., the leading supplier of object request broker (ORB) technology, has announced a migration services agreement to transition users of Sun Microsystems' NEO technology who require multi-platform or middleware support to Visigenic's VisiBroker for Java and VisiBroker for C++ technology. To meet the unique, mission-critical needs of these users, Sun Microsystems and Visigenic will be providing consulting as well as documentation that maps key NEO features to VisiBroker features in order to streamline the transition between ORBs. Effective immediately, Visigenic will market and sell the VisiBroker ORB to established sun Microsystems customers.

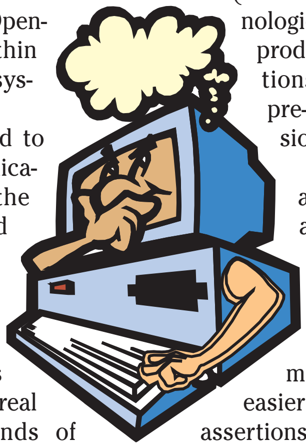
Sun Microsystems and Visigenic will offer existing NEO customers a portability/migration map, specialized consulting services and product discounts on Visigenic's ORB products. IIOP is an object messaging protocol for communication between network-based client/server software programs and enterprise applications based on CORBA standards.

For more information on both companies, see their Web sites at www.visigenic.com and www.sun.com.

ILOG JViews Chosen for HP OpenView Java User Interface

(Mountain View, CA) - ILOG S.A., a leading supplier of advanced software components, has announced that Hewlett-Packard Company will use ILOG JViews™ to develop a Java™-based user interface for HP OpenView network and systems management software.

HP plans to use ILOG JViews to broaden its support of the Internet for its OpenView family of network and systems management products. ILOG JViews' Web-based functionality will extend HP OpenView software's reach within large enterprise information systems.



ILOG JViews is well suited to the needs of the telecommunications market because of the product's scalability and robustness under load. For the first time, Java developers will be able to harness the power of quadtree algorithms and other graphics optimizations that facilitate real time displays with thousands of objects.

For more information, see their Web site at www.ilog.com.

Build Multithreaded Apps with O'Reilly's "Win32 Multithreaded Programming"

(Sebastopol, CA) - Multithreading, available for personal computers with the advent of Win32 operating systems, presents Windows programmers with both increased capabilities and challenges. O'Reilly & Associates' new book, *"Win 32 Multithreaded Programming"*, gives programmers the knowledge they need to skillfully construct efficient and complex multithreaded applications.

"Win32 Multithreaded Programming" by Aaron Cohen and Mike Woodring explains the concepts of multithreaded programs, from basic thread synchronization using mutexes and semaphores, to advanced topics like creating reusable thread pools or implementing a deferred processing queue. It illustrates these principles with real-world applications and carefully constructed examples and shows readers how to take full advantage of Win32's multithreading capabilities.

The CD-ROM accompanying the book features Mcl, the authors' C++ class library for multithreaded programming, which both wraps multithreaded API functions and easily supports more complex multithreaded scenarios. for programmers using MFC, an additional library, Mc14Mfc, is included for MFC compatibility.

For more information, see the O'Reilly Web site at www.oreilly.com.

Reliable Software Technologies Releases Java Assurance Tools for Free download

(Sterling, VA) - Reliable Software Technologies Corp. has released two new products for testing Java™ applications: AssertMate™ Version 1.0 pre-release and TotalMetric™ Version 1.0 Pre-release.

AssertMate is a system that aids Java engineers in safely and accurately placing software assertions within their Java programs. Software assertions assist in finding bugs earlier in the development process (when they are easier and cheaper to fix). Until now, assertions were missing from the Java development environment.

TotalMetric for Java is the world's first commercial software metrics tool to calculate and display cyclomatic complexity, level-of-effort and object-oriented metrics for the Java language.

RST's goal is to make it simple for developers to rigorously test their code as early in development as possible, without intrusion on their normal productions. Both AssertMate and TotalMetric are available for free download from RST's Web site for a limited time at www.rstcorp.com/tools.html. Information on RST's other tools can be found at www.rstcorp.com. You can also reach them by phone at 703 404-9293.

THE GRIND

New Year's Revelations

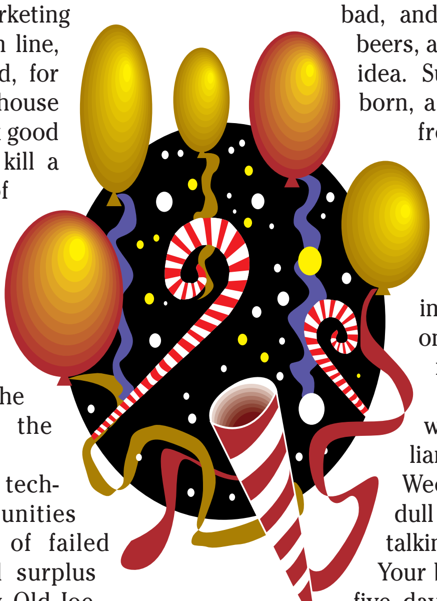
by **Joe S. Valley**

"Just Joe with his cheap gin sitting at the bar, thinking about the past year, pondering the upcoming year"

OK, so I'm a couple of weeks late on the New Year. Hey, I am using a Java-powered watch, so what do you expect? Maybe when the watch gets a JIT, I can get the correct time and date.

Went out for a couple of drinks last night to my hang-out in The Valley... the Bucket-O-Bits. In its past life, this sleazy dive was the high-brow Ely McFly's, the center of everything cool about Silicon Valley. Ely's was next door to Apple and near a dozen start-ups. Hot marketing managers would stand in line, gold AMEX card in hand, for the honor of buying the house a round of drinks. I drank good gin at Ely's. Enough to kill a normal man; but of course, old Joe isn't normal. I am a survivor.

The crowd wasn't there, yet. Just Joe, working on his second gin and tonic, sitting at the bar, thinking about the past year, pondering the upcoming year.



Each year brings new technologies, new opportunities and fresh inventories of failed products into the local surplus stores here in The Valley. Old Joe keeps his perspective by regularly visiting such stores as Haltek and Weird Stuff. In those places, staring at me, are pallets full of products that some engineers designed on the usual rush schedules, some marketing guys promoted even though they were way underpowered and six months late and some sales guys sold even though the customers didn't think they wanted them.

That software I worked through Christmas one year to get into distribution? Over in the "$5 and less" bin. The Digital Video cards that needed superhuman ASIC development schedules to make it in time for MacWorld? Look in the "Free! Just haul them out of here" section. Go to places such as Haltek and Weird Stuff. Look around there, soak it up. It keeps you humble. I see lots of engineers walking around in these places, thinking. But there are few marketing or sales people. They move on, quickly. I think there is a message in it all. Try to do good stuff, but don't leave any of your blood where you work. It isn't worth it.

Sometimes, the planets line up and the product is a hit. Most of the time, something happens. Apple was a hit, Fortune Systems wasn't. Small changes at any time could have reversed the order. As the bottom of the glass is fast approaching, I hold the hope that some history can repeat itself. The project that renewed the spark in a dying company, remember? Yea, Joe was there, in the middle of it all. Things were bad, and over one too many beers, a sales guy gave you an idea. Suddenly, a team was born, a goal was set and the freight train was in motion.

Running on all cylinders, the project came together. Your software worked in beta, the ASIC worked on the first spin. The marketing guy, who you thought was worthless, became brilliant in his pitch to PC Week. At an otherwise dull Comdex, people were talking about your product. Your booth is packed for all five days. Suddenly, the supply-base manager is having fits, trying to get enough parts. The sales guys are stuffing the channel full of the new product, and selling through the junk in inventory as well. The Western Region is 150% of quota. The stock blows through $10 and hits $18 in a month. Higher than it had been since the IPO days. Your stock options are in the money. This year, you can go to Europe instead of Santa Cruz. The president of the company buys everyone on the team a weekend at a spa in Napa. Life is good.

It happens sometimes. When it doesn't, your labor of love ends up in the bargain bin at Weird Stuff.

Round three of the gin and tonic is heading my way. People are starting to show up in this dump, and they are a lot younger than old Joe. The Bucket-O-Bits is their Ely's. They will toast the success or lament the failure of their year's work.

New year is here. Look back quickly. Order a round of drinks and toast to new hopes. Such is life in The Valley.

Joe S. Valley is a scarred veteran of the Silicon Valley wars. It was either writing this column or heading back into therapy. His company can't afford mental health care coverage anymore, so writing is the only option. There are a million stories in the Valley and Joe knows lots of them. Got a good story? E-mail him at Joe@sys-con.com

Joe@sys-con.com

JavaWorld
Ad

KL Group
Full Page Ad