EARN BIG BUCK$ WITH JAVA

JAVA™ DEVELOPER'S JOURNAL

JavaDevelopersJournal.com

Volume: 3 Issue: 7, 1998

SYS-CON PUBLICATIONS

Full Page Ad

Full Page Ad

Full Page Ad

JAVA DEVELOPER'S JOURNAL

EDITORIAL ADVISORY BOARD

Ted Coombs, Bill Dunlap, David Gee, Arthur van Hoff, Brian Maso, Sean Rhody, Kim Polese, Rick Ross, Richard Soley, George Paolini

Editor-in-Chief: Sean Rhody
Art Director: Jim Morgan
Executive Editor: Scott Davison
Managing Editor: Anita Hartzfeld
Senior Editor: M'lou Pinkham
Editorial Assistant: Brian Christensen
Technical Editor: Bahadir Karuv
Visual J++ Editor: Ed Zebrowski
Visual Café Pro Editor: Alan Williamson
Product Review Editor: Jim Mathis
Games & Graphics Editor: Eric Ries
Tips & Techniques Editor: Brian Maso

WRITERS IN THIS ISSUE

Syed Q, Abbas, Juergen Brendel, David Cittadini, Maros Cunderlik, Mike Fichtelman, Eric Greenfeder, George Kassabgi, Sean Rhody, Ajit Sagar, Mukul Sood, Alan Williamson, Ed Zebrowski

SUBSCRIPTIONS

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

Subscription Hotline: 800 513-7111

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

Publisher, President and CEO: Fuat A. Kircaali
Vice President, Production: Jim Morgan
Vice President, Marketing: Carmen Gonzalez
Advertising Manager: Claudia Jung
Advertising Assistant: Erin O'Gorman
Advertising Intern Jaclyn Redmond
Accounting: Ignacio Arellano
Senior Designer: Robin Groves
Designer: Alex Botero
Webmaster: Robert Diamond
Senior Web Designer: Corey Low
Customer Service: Rae Miranda
Sian O'Gorman
Paula Horowitz

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.

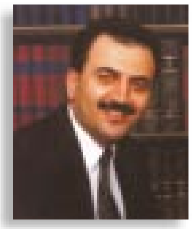
ISSN # 1087-6944

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

BPA Membership Applied For

FROM THE PUBLISHER

Fuat A. Kircaali



Big Money in Java

In this month's cover story we focus on how to earn big money with Java on three levels. ***JDJ***'s feature brings you practical tips for the Java programmer by both Ed Zebrowski, ***JDJ***'s product review editor, and Sean Rhody. We also have a somewhat technical discussion by Juergen Brendel covering the software engineering issues in startup companies. We conclude the cover story with a feature on the "Java Electronic Commerce Framework," which concerns Sun's JECF architecture for established businesses.

New JDJ Editor

I'm pleased to announce that Sean Rhody has joined ***Java Developer's Journal*** as editor-in-chief. Sean has been the chief editor of ***JDJ***'s sister publication, ***PowerBuilder Developer's Journal***, for the past four years. You've been reading his multitier series in ***JDJ***, volume 4, issues 4 and 6. Starting with this issue you'll see his editorial direction and influence in ***JDJ***.

What's New at JDJ?

With this issue you're likely to come across ***JDJ*** in many more newsstands than before. This is the first issue of ***JDJ*** that's being distributed by Curtis Circulation Company. Curtis, the world's largest magazine distributor, can take most of the credit for our expanded newsstand distribution. Over the past twelve months our newsstand sales have increased more than tenfold, making ***JDJ*** one of the fastest growing technology publications anywhere. You'll also notice that we've redesigned our cover.

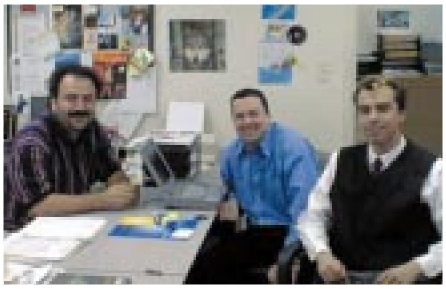
Industry Meetings at JDJ Offices

Since JavaOne we've been keeping busy meeting with Java vendors to bring you the latest developments from the Java industry. Last month we had visitors from several companies who presented new versions of their Java products. We'll be bringing more information on these new and exciting products in our upcoming issues.

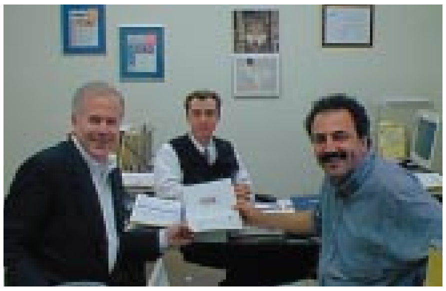
Among several other new products, on July 16 we got a sneak preview of IBM's Visual Age 2.0 Enterprise Edition. The product is scheduled to ship in August and we will have an extended feature story on this new release in our next issue.

*On July 16 IBM's Rick Weaver visited **JDJ** offices to present a sneak preview of IBM's Visual Age 2.0 Enterprise Edition. Photo Left to right: Rick Weaver (IBM), Corey Low (**JDJ**), Stephanie Clark (IBM) and Fuat Kircaali (**JDJ**).*

Meredith Fuller, and Michelle Guile, SCM product manager of MKS, announced Source Integrity 3.1 online in JDJ offices on July 14.



*Novera Software cofounder Michael Frey visited **JDJ** to update our readers.*



Bob Gleason, president and CEO of Riverton Software, presented HOW for Java.

Full Ad

FROM THE EDITOR

Sean Rhody

Beginnings

Almost invariably, when I write an article I know pretty much what I want to say, and the part I have the most difficulty with is the introduction. This is my first column as editor-in-chief of ***Java Developer's Journal***, so while most of you are familiar with our magazine, many of you may be less familiar with me. I'd like to take this opportunity to introduce myself, and also describe where I think we'll be taking ***JDJ*** over the next few months.

First of all, this is the second magazine I've edited – the first being our sister publication, ***PowerBuilder Developer's Journal (PBDJ)***. I've been writing about technology for approximately five years now, starting with several articles in ***PBDJ***, then a regular column and finally as editor for the past two years.

During that period I think we've seen the software version of Moore's Law, the one that states that the number of transistors on a chip will double, roughly, every eighteen months. That software version is of course the programming paradigm under which we work and it takes longer than eighteen months, but it changes just the same. I started working in the industry as client/server was enduring its growing pains, and relational databases were just beginning to penetrate smaller companies. I spent a good deal of time developing PowerBuilder applications and frameworks to do various tasks.

Then, two or three years ago, the paradigm began to shift again. We started talking about application partitioning and three tier architecture. Eventually the horsepower (hardware and software) was available at the right price to physically make three tier design architecturally feasible.

Right along with that came the explosive growth of the Internet, and HTML as the default language. Java stepped up as a language that was suitable for any number of purposes, particularly for platform-independent networked applications and for simple but powerful multitasking.

As a technical architect for a large consulting company, I have been involved with Java and Internet programming for several years now. I'm also happy to have the opportunity to move to this magazine and help guide its course as we discuss all things Java.

In particular, I find Java a challenging language to write about and describe. That's because no one vendor really owns Java in the same sense that a single vendor owned PowerBuilder or Visual Basic. We've got Java development products from a dozen companies, large and small, each with its own take on what is the right mix of visual assistance to the programmer. We've got hundreds of smaller ISVs creating JavaBeans that we can use. We've got companies developing hardware that runs a Java OS and companies developing VMs for an ever-increasing number of operating systems. The best way to describe this is to call it a "movement."

What we'll try to do over the course of my tenure here is to structure our information and content in such a way that it will help you make sense of the various issues and choices you have in the Java world. We'll be doing our ***JDJ*** Reader's Choice Awards shortly to select some of the best and brightest products and services available for Java.

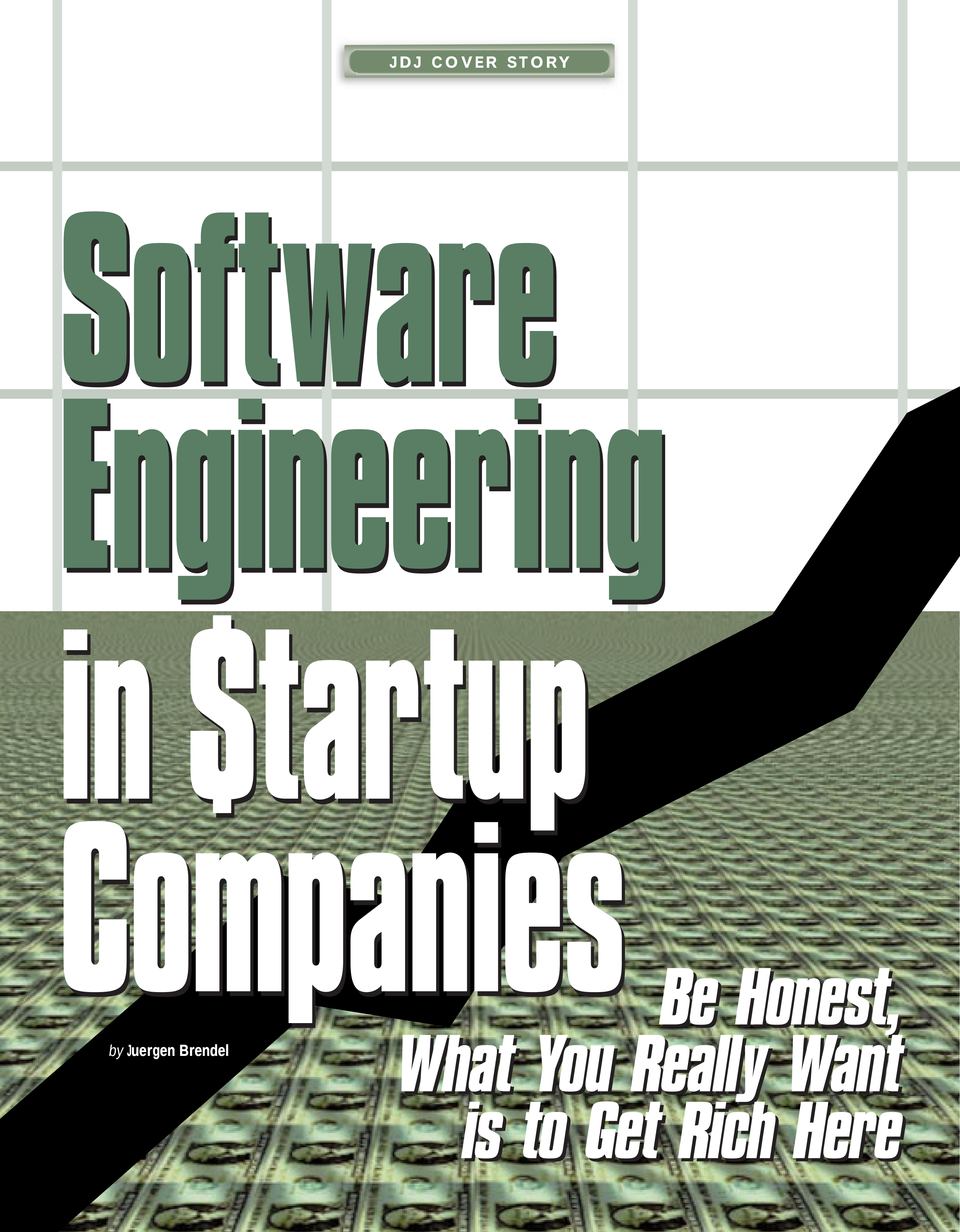
One of my current focuses is on distributed architectures, which is a natural area to employ Java. We'll try to pass along informative articles on building such systems with Java and CORBA, and point out alternatives (see my series of articles in ***JDJ*** on Jaguar CTS, for example).

We'll also focus on general techniques for programming in Java, with articles that explain how to take on specific tasks that Java provides particularly elegant solutions for, or just plain interesting code. And we'll try to keep you abreast of the various developments in the industry so you can make informed decisions on whose tools to use.

If I've left out some of your favorite topics, if you just want to see something in particular or if you have a proposal for the magazine, you can contact me at roadhog@nac.net. I do try to answer all e-mails, including any constructive criticisms you may have. I hope you'll enjoy reading ***JDJ*** as much as we enjoy publishing it; I'm glad to be on board.

About the Author

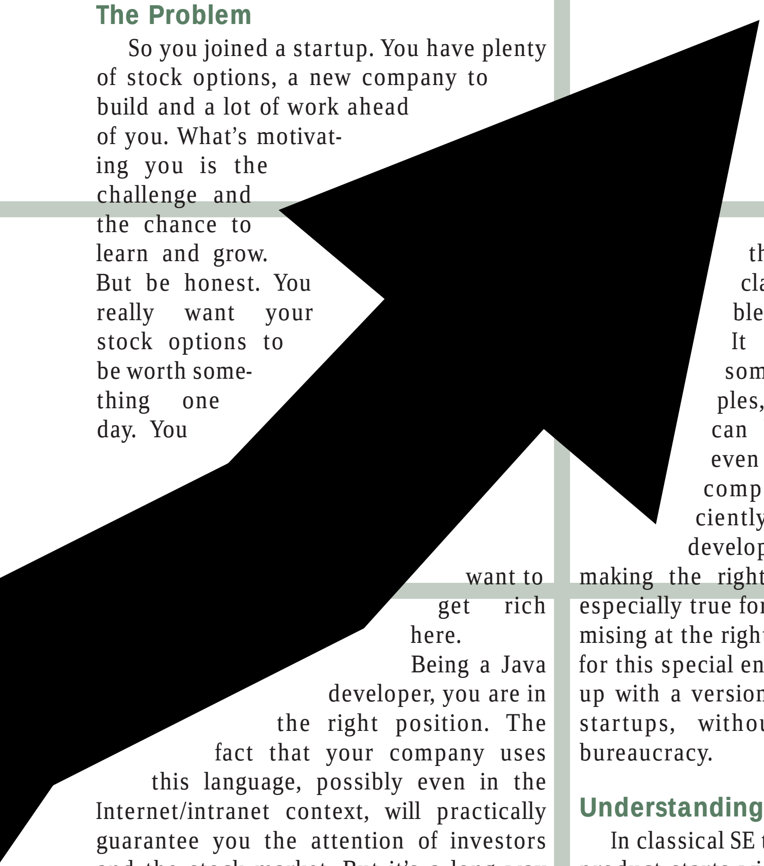
In addition to being editor-in-chief of ***Java Developer's Journal****, Sean Rhody is a senior consultant with Computer Sciences Corporation where he specializes in application architecture, particularly distributed systems. You can contact Sean at roadhog@nac.net.*



Software Engineering in $tartup Companies

Be Honest, What You Really Want is to Get Rich Here

by Juergen Brendel



The Problem

So you joined a startup. You have plenty of stock options, a new company to build and a lot of work ahead of you. What's motivating you is the challenge and the chance to learn and grow. But be honest. You really want your stock options to be worth something one day. You want to get rich here.

Being a Java developer, you are in the right position. The fact that your company uses this language, possibly even in the Internet/intranet context, will practically guarantee you the attention of investors and the stock market. But it's a long way from early hype to long-term business success. There are a few exceptions, but most companies will need a solid product and many happy customers to have any kind of success. Building a quality product, however, is where the problem begins. Classical approaches to software engineering (SE) do not work in a startup environment, but without SE your product will lack quality.

Classical Software Engineering

Software engineering is the application of engineering principles to the building of software. Classical SE suggests that development takes place in stages: analysis, design, implementation and testing. It defines the techniques and processes used to set up a system of control, testing and approval at every one of these stages. The SE processes require each stage to be completed before the next stage begins, because the output of a stage must first be understood, tested and approved. As you can easily imagine, SE requires some bureaucracy. However, when properly applied, SE holds the key to timely delivery of quality software.

The Startup Difference

Startups face several unique hurdles that must be overcome to ensure success. Therefore, classical SE techniques and principles need to be modified for this special environment:

1. Short time to market
2. Immature, undefined market
3. Fast growth in the number of employees

It is often argued that the deliberate processes of classical SE are not applicable under these conditions. It is my contention that some of the basic SE principles, processes and methods can be utilized successfully, even by startups, to make the company operate more efficiently. As one experienced developer put it, "Engineering is making the right compromises." This is especially true for any startup. By compromising at the right places and modifying SE for this special environment, you can come up with a version of SE that will work for startups, without creating too much bureaucracy.

Understanding the Requirements

In classical SE the life cycle of a software product starts with requirements analysis. The goal is to identify and quantify all customer requirements through close interaction with the customer and market research. It is important to quantify requirements. For example, a requirement could either read "the data packets should not be too big" or it could state that "the data packets should not exceed 500 bytes in size." The first requirement is not quantified; its satisfaction is left to interpretation. The second one is quantifiable and therefore testable. Since all requirements are listed, and each one will later have a test assigned to it, a successful pass of all tests indicates that the development effort has been completed.

In startups, the analysis phase does not proceed in this manner. There are no customers in the beginning from whom you can get feedback. Also, the market is still so new, no one can say what is really needed. How do you identify all customer requirements? You don't. You use your common sense, experience and creativity to come up with an initial set of requirements. Should they be formulated in a quantifiable manner? That depends. You may be more certain about some aspects of your product than others. Unless you have an explicit statement from customers, it doesn't make sense for management or marketing to make up an artificial number that is not based on real-world input. It would be a waste of engineering time and money to try

Big Bucks in Java

by **Ed Zebrowski**

Three years ago I was earning a meager living by working at a local resort hotel. In my spare time, I found a nice diversion playing around with my 486. Soon thereafter, I was making good side money by developing Web pages, and occasionally finding some technical consultation work. It was right about then that I became aware of a new programming language. It had been used, I was told, to run the microprocessors in small appliances such as toasters and microwave ovens. Because of the way this new language was structured, it lent itself perfectly to Internet applications. I was both fascinated and challenged by this strange and wonderful new brew, so I decided to learn as much about it as I could.

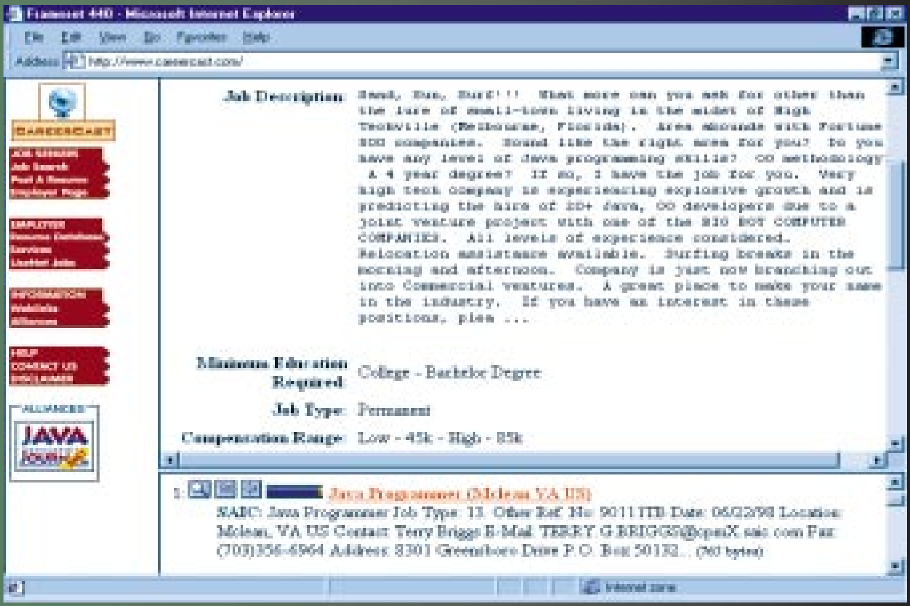
I didn't realize it at the time, but I was standing in the filter basket of what was to be the largest "percolation" of business and technology the world had ever seen. By sipping from this cup of Java, the caffeine rush would carry me well into the next century.

Maybe you're also ready to take this ride. Perhaps you are familiar with another programming language and are just now starting to realize the potential of Java, or maybe you're just bored with your job down at the ice cream parlor. Whatever the case may be, Java now provides unprecedented opportunities, if you know how to look for them.

Step One: Sharpening Your Skills

We all like to think we know everything about Java. The truth of the matter is that Java, like most programming languages, requires constant updating and upgrading of skills to stay current. Nothing is more awkward than taking a contract or job and then discovering that you're not as up-on-things as you thought. This can be avoided by brushing up on your Java skills. The method you chose to do this will vary, depending on how much "brushing up" you think you'll require. If it's only been a little while since your last Java project, a visit to one of many Usenet groups may be all you need. I was able to find hundreds of these by running a Usenet search on my server. They cover broad topics, such as CORBA development, and narrower ones, such as the use of a specific IDE. This would

Sand, Sun, Surf!!! Earn up to 85k


Frameset 440 - Microsoft Internet Explorer
File Edit View Go Favorites Help
Address http://www.careercast.com/
CAREERCAST
ALLIANCES
JAVA
Internet zone

Job Description: Sand, Sun, Surf!!! What more can you ask for other than the lure of small-town living in the midst of High Techville (Melbourne, Florida). Area abounds with Fortune 500 companies. Sound like the right area for you? Do you have any level of Java programming skills? OO methodology A 4 year degree? If so, I have the job for you. Very high tech company is experiencing explosive growth and is predicting the hire of 20+ Java, OO developers due to a joint venture project with one of the BIG HOT COMPUTER COMPANIES. All levels of experience considered. Relocation assistance available. Surfing breaks in the morning and afternoon. Company is just now branching out into Commercial ventures. A great place to make your name in the industry. If you have an interest in these positions, plea ...

Minimum Education Required: College - Bachelor Degree

Job Type: Permanent

Compensation Range: Low - 45k - High - 85k

1. Java Programmer (Mclean VA US)
SAIC: Java Programmer Job Type: 13 Other Ref. No: 9011TTB Date: 06/22/98 Location: Mclean, VA US Contact: Terry Briggs E-Mail: TERRY.G.BRIGGS@cpmX.saic.com Fax: (703)356-6964 Address: 8301 Greensboro Drive P.O. Box 50132... (760 bytes)

Many online placement services provide access to updated, open Java positions

also be very useful if you've just taken a job that requires the use of an IDE that you're not familiar with.

If you haven't developed a Java project recently or if you're new to the language, more thorough instruction may be in order. There are several good books that can bring you up to speed, and a visit to your local library or favorite bookstore can fetch the material you need. If you think you may need a more structured learning environment, there are a couple of suggestions I can make. One excellent place for online training is ZD University. ZDU offers a full line of courses on everything from Java development to Internet business strategy. Drop by their Web site at www.zdu.com for more information. For those of you who prefer a classroom atmosphere, check your local newspaper. Most cities have a good selection of technical colleges that now offer great Java instruction. I was able to find at least twelve of these, and they offered everything from "Java as a First Programming Language" to some quick brush-up classes. Call them for more information. Financial aid and job placement is usually available.

Finding Regular Full-Time Employment

Who hasn't dreamed of landing that perfect job? A great schedule, excellent pay and health insurance are usually the attractive features that lure Java programmers into these positions. The type of salary you'll earn depends on many factors. Entry level Java programmers can expect between $30k - $40k a year. I've noticed that some companies will pay the more experienced and proficient Java developer as much as $70k-$100k. Bear in mind, to earn that kind of money you'll need to be proficient at other software development languages besides Java. Companies love to hear that you're good at C++, C, Visual Basic and UNIX as well. Many firms hire Java developers for both Internet and intranet applications. Some knowledge of Database management can be a huge plus here.

Before applying for a particular job, it's a good idea to think about whether or not you will relocate. Some software-orientated positions require a good deal of moving around. One friend of mine had to move from San Antonio to Houston to New Orleans and then to Los Angeles inside of two years. Although the position paid extremely well, he left the job so he could put down some solid roots. Some companies operate from one city, but send their employees on assignments all over the country. If you're going to take a position like that make sure you can handle the jet lag.

If you're seeking full-time employment with a company, why not look on the Internet? There are several good Web sites to get you started. One of my favorites is the Monster Board. The Monster Board offers an updated, indexed listing of hundreds of great jobs. You can search for the job you want either by location, category or entering a keyword. Using their search engine, I was able to find openings for software development in dozens of disciplines, and in all corners of the country. Openings can be applied for imme-

to satisfy artificial targets. List all the requirements you can think of and quantify those that you can. Don't quantify requirements with values that have no basis, although you should list a requirement even if you cannot quantify it. By listing requirements and making sure they are addressed during development, you ensure that no requirements are missed. Later, when you do finally get some customer feedback, you can try to fill in the blanks in the requirement list. Even though this goes against some of the foundations of classical SE, it is one of the compromises you have to make in a startup.

New requirements that have not been addressed in the initial analysis and design will be set aside for the next release of the software whenever possible. Because of the uncertainties that go with developing a brand new product for a brand new market, most of the compromises will be made in the analysis phase of the SE process when you work in a startup company.

Designing a System

Once the requirements have been stated, you can begin to design a system that meets them. Since a startup operates in a new market with potentially new technology, your work is likely to contain a significant research component. A prototype may be developed even during analysis or design. Some assumptions may turn out to be technically infeasible. This may force you to change the requirements and modify the design. The analysis and design stages, so nicely sequential in classical SE, are likely to pass through multiple iterations of the analysis, research and design phases in startups.

The design stage is important for any product in any company, which is especially true in a startup. Markets and requirements change constantly. Hence, your product will need to change a lot over time. In fact, the first version of your software will probably have to be redone after its release. It would be great if you could just rearrange the product at this point rather than rewrite it. To a large extent, this is possible if you design your system out of small, independent objects. Well-designed objects exhibit high cohesion and low coupling.

High cohesion means that the object's data and methods remain focused on one task or functionality. The object is an expert at doing one thing. If you have to modify this object's functionality, the changes should not affect other objects.

Ad

diately, or you can post your “electronic resume.” Give the Monster Board a try at www.monsterboard.com.

Another good site I’ve stumbled upon is JVsearch. This is a search engine linked to an extensive database of Java professionals who are looking for work, and employers who are seeking the expertise of the Java developer. JV also allows the submission of electronic resumes. They can be accessed at www.jvsearch.com.

JavaDevelopersJournal.com offers a “Java Jobs” link to Careercast’s online job search engine

Working as an Independent Contractor

If you’re like me, you’ve had it up to here with the “corporate grind.” Office politics, traffic jams and fear of “downsizing” have driven thousands of working professionals out of the cubicle and into the ranks of the self-employed. Although it seemed a little frightening at first, today’s Java market offers fairlygood opportunities to those who know how to find them.

Getting started as a contractor can be a bit tricky at first. I was able to find some work from a small ISP in my area. He was later helpful by recommending me to a few of his clients who were looking to develop some Java applications. They tossed my name around, and before I knew it things began to snowball. Things don’t always go that well, however. I’ve known a few people who had some good-looking accounts set up for themselves only to have it all fall through. Before long they were back to the grind.

If you still think you want to give the wonderful world of independent contracting a try, here are a few pointers you may want to remember:

- Be careful not to over/under charge for your services. No one in their right mind is going to pay a thousand dollars an hour for someone who obviously isn’t that highly skilled. By the same token, if you consistently charge too little, you may become stereotyped as a “nickel and dime” guy. I’ve noticed that after this happens it is very difficult to charge more for your work. Rather than pay you more, some companies or clients will go after some one else. I’ve worked for as little as $50.00 an hour and for as much as $500.00 an hour. Try to estimate what the going market rate is for what you’re doing. For example, if you’re doing some work that involves C++, obviously going the charge will be considerably more than if you’re just doing an applet for a basic Web page. If you’re going to be doing some UNIX work, that should be at least another 10-25% more.
- Take everything into consideration when charging clients. When I was just starting out I did a couple of foolish things. One mistake was forgetting that I no longer had health insurance included in my wages. I thought I was making a killing, but I woke up after I caught a nasty flu. Not only did I have to dig deep into my own pocket for the doctor and a prescription, but it then dawned on me that I was going to have to purchase a health plan of my own. You might want to keep this in mind when deciding what to charge. Another big boo-boo was not putting cash aside at the end of the year for good old Uncle Sam. It’s very difficult at first to try and anticipate what your taxes might be, but try to put some cash away for it. You don’t want to end up owing an entire year of taxes to the IRS. Take my word on that!
- Get the word out! The best contract work comes from word-of-mouth advertising. The best way to ensure this is to deliver speedy, efficient results while maintaining a good relationship with the client. Post your qualifications all over the Internet, as well as newspapers and trade magazines. If you do good work, offers may start coming to you rather than your having to look for them.

The explosion of Java as a sought-after development language has created opportunities that only a few years ago we could not even fathom. Don’t be afraid! Grab your surfboard! Ride this wave into the new millenium!

Low coupling means that the interfaces between objects are small, few and without side effects (e.g., no global variables). The fewer interfaces an object uses, the fewer dependencies it has. Code changes and rearrangements affect a smaller portion of the total system. You can therefore make such changes faster and safer.

Such modular design also facilitates the rapid growth of a startup. When interfaces to small objects are well defined, you can easily assign well-bounded tasks to new employees. Also, since smaller modules are less complex, new employees can learn and understand their tasks by focusing on the relevant object. As they work on more and more objects over time, they will develop a broader understanding of the system. They can also be productive while climbing the learning curve. If you have few monolithic modules, however, new employees have to work inside such a module with no clearly defined interfaces between their work and that done by other developers. Not only is this an error-prone environment, it will also increase the communication overhead among employees. Unit-testing or at least a code review is easier with a small module.

Several methodologies are appropriate for high-level design. They usually involve drawing bubbles (objects) connected by arrows (messages) on paper. All of the methodologies are quite effective and help you see the system as a whole. If you are using Java to implement the software, you can use the actual implementation language rather than pseudo code to do your low-level design. Use interfaces for specifying APIs so that classes that implement those APIs can be written later. Object definitions can be specified as abstract classes, which can then be subclassed and filled in with specific implementation details.

The Implementation

Implementation of the design should be one of the smallest tasks in software development. If requirements analysis and design have been done thoroughly, you will simply fill in the blanks during implementation. You should create proper source code documentation, check return values and follow a suitable coding and error-handling standard. Such standards should be defined before implementation ever starts. Java’s exceptions provide a good foundation for implementing error-handling, but they do not define a complete error-handling standard.

Ad

The error-handling standard determines how user errors, program exceptions (for example, I/O errors) and internal system errors are handled. Setting a standard once will save developers from having to make these decisions again and again, in a possibly inconsistent way.

Standards will make life easier not only for new employees, but also for those who do maintenance. Reviews can be used to test for compliance with the standards.

Testing

Testing does not start only when coding is complete. Analysis and design documents can be tested through reviews. In a review, check that all requirements have been addressed. In general, reviews are an important part of testing at all stages. During implementation they can even substitute for more resource-intensive kinds of testing, such as unit testing. Once you have finished coding subcomponents or the whole system, run automated test cases. You can start designing the test cases as early as the analysis stage.

"Since money is already a point of motivation, it can be used to convince dubious developers or management."

Convincing People

SE techniques are effective only if they are used. Unfortunately, many developers and managers resist SE practices. This is not so much an engineering issue as it is a psychological and organizational one.

Many individuals who are attracted to startups have had bad experiences with the design-by-committee style of SE practiced in large companies. They come to startups because they don't want to deal with bureaucratic overhead. When you suggest SE practices to these individuals, the answer will often be, "Sure, we need some SE, but I've seen how much overhead it requires. We don't have time for that now...." The practices discussed here, however, don't require much overhead.

These practices do help you build a quality product that is easier to maintain and grow as the company grows and as markets change. Less time will be spent on debugging and fire drills. This is important for everyone who wishes to have a life outside the company. The higher quality of your product and the responsiveness of your company will result in satisfied customers, good press, and higher sales. This will keep everyone motivated, and the value of your stock options will increase. Since money is already a point of motivation, it can be used to convince dubious developers or management.

As an early member of a new company, you have both the opportunity and the obligation to shape its culture. You need to create an environment in which SE is practiced naturally. New employees, immersed in this environment, will adopt SE without even questioning its application. Rather than forcing your point on people, guide them toward seeing that SE can be a powerful solution for the software development problems encountered in startups.

This brief overview of how to use SE in startup companies will be followed in upcoming issues by details on how SE practices make the software development process more efficient in startups.

About the Author

Juergen Brendel is a software architect at Resonate Inc. He welcomes your feedback via e-mail at jbrendel@resonate.com.

jbrendel@resonate.com

How to Win Friends, Influence People and Make a lot of Money – the Java Way

by **Sean Rhody**

In medieval times alchemists searched for the philosopher's stone, the secret to converting lead into gold. Fortunately, our search to convert code into cash is slightly easier. The focus of this month's issue is on making money – a great deal of money, we hope – with Java. I won't pretend to know the exact path to wealth and fame, but I have a few suggestions about how to be successful with Java.

One of the easiest ways is to get to know the language really well. This sounds simplistic, but it isn't. Until you know the full capabilities of a language, you can't really gauge where and when it is appropriate to use it or what tools to use it with. That's important knowledge. Most companies today are wrestling with two important information technology issues – Year 2000 and the shift to an Internet paradigm.

I'll just pray that Java is Y2K-compliant. In the meantime, we've got a host of companies out there who are wrestling with the conversion from client/server to Internet, or even from host-based systems to the Internet. Most of these companies have heard that Java is the way to go, but in truth there are a number of options, depending on business needs. If you're a corporate developer who wants to use Java, you need to know how to highlight its suitability to this task. If you're a consultant who has just recommended Java, you need to be able to defend your recommendations against other solutions, such as DHTML, ASP or XML. If you don't understand the language and its capabilities, you'll have a difficult task.

Another thing that can drive your salary up is understanding and applying a robust methodology. Using a modeling tool is not enough. A tool just makes it easier to create garbage faster. Without a methodology that covers the details of how you will partition your application, you're bound to make costly mistakes. You add tremendous value to your company or client by knowing how to design the application, not just how to build it. And that's bound to add to your own bottom line.

Another tenet of success is "Buy, Partner, Build," in just that order. Java was designed to be a language that enables code reuse and extension. You shouldn't be out trying to build a data-bound grid control – there are too many out there already. Buy one. If the tool or widget you are looking for is not quite what you want, try to partner with the developer to extend the tool. Build it only when you have no choice. This philosophy helps you concentrate on the logic that differentiates your particular client or employer's business from its competitors – and that's the best way to add business value. Most of us should be tool users, not tool builders. A good tool user can find the right tool for the job and concentrate on the business at hand.

We've seen tremendous growth in demand for Java in the past few years, and as more and more products begin to adopt Java as a part of their environment (e.g., Java-based stored procedures in relational databases), that trend is likely to continue. Java can be your philosopher's stone. If you know how to use it well, it can turn your lead into gold. Otherwise, all you've got is a rock.

Ad

Java Electronic Commerce Framework

JECF is Sun's Java architecture for supporting secure electronic commerce transactions on both the Internet and within intranets

by **Mukul Sood**

The JECF provides services (application programmer interfaces) on top of Java for creating payment and financial applications. The current version (beta release 0.8) provides services such as graphical user interface, secure encrypted database, capability mechanism, access to cryptography, applets and infrastructure for purchasing.

JECF Architecture

JECF has a layered architecture; each layer has predefined responsibilities and uses the services of the layer below it. The layers (see Figure 1) are:

- Merchant Applet layer uses Java applets to present an interface, e.g., a shopping mall, store front, etc. These applets do not require a long-term customer-to-merchant relationship; they are appropriate for implementing short-term customer relationships such as the shopping experience.
- The Cassette layer implements long-term customer relationships such as credit cards, home banking and brokerages. A cassette is a jar archive file that contains resources (commerce beans, graphics, etc.) and is digitally signed with one or more roles. Each role provides specific capabilities to the contents of a cassette. Similar to applets, cassettes are downloaded from servers to client computers. Unlike applets, which disappear when users quit their browsers, cassettes are retained on the customer's system. Cassettes store information in a database provided by JECF. They may safely store valuable information such as public key certificates and transaction records, since the entire database is encrypted. Examples of cassettes include SET

certificates and protocols, home banking and credit cards.

- The Java Commerce Package layer implements the infrastructure needed by the merchant and the cassette layers. Features of this layer include a user interface, an application model, a database and access to strong cryptography. This layer consists of three main service layers.
- The GUI services layer: provides a graphical interface similar to a wallet. The JECF user interface components consist of generic widgets, such as panels, scrolling panels, scroll bars, tree controls, buttons, labels and so on. The services are provided through the classes and interfaces in the package javax.commerce.gui. A JECF UI (Wallet) consists of a combination of these widgets and is contained in a UI Cassette. Creating a new user interface for a JECF wallet involves creation or modification of a user interface cassette.
- The Application services layer: can be used to implement common business operations such as sales and trading. The central classes in this layer are JECF, JCM, OperationContext, OperationThread, UICContext and WalletContext (package javax.commerce.base). I'll revisit these classes later when I walk through the JECF application. Before going on to the security model, I should also mention some of the interfaces defined in this layer, namely, Operation, Instrument, CommerceContext.
  Operation is an action, transaction or unit of work, for example, PurchaseOperation. Instrument facilitates an operation; a common instrument is a credit card. It could also be communication protocols, authentication mechanisms and so on. A CommerceContext object encapsulates information relevant to an operation. This includes what frame to show status information into, the UIFactory relevant for the operation, and the methods used to show a Web document.
- The Foundation services layer: includes the database classes, access to strong cryptography, smartcard device access and various common utility classes such as Money. The database classes (javax.commerce.database) provide an interface to an embedded database, which is small and lightweight, and provides a subset of functionality of a relational database.

JECF Security Model

The Java Sandbox security model supports trusted applets and untrusted applets, but not partially trusted applets. Commercial entities (e.g., businesses, trading houses, banks) operate on relationships, that have defined limited trust among themselves. To get around the restrictions of the sandbox model, JECF provides the Gateway security model which supplies the means to implement contractual trust relationships. This model uses the safety features of the Java language as well as the infrastructure of the Java Sandbox security model. It also provides the ability for developers to create arbitrary trust relationships.

The two main concepts in the Gateway model are Principals and Roles, and Capabilities. A Principal can be a person, a bank, an application program. A Role represents a right (admin, user...) and is used in the Gateway model to authenticate Tickets in a Gate. "Role" is basically a container for a public key. The Role interface, defined in the package javax.commerce.cassette, could be implemented in different ways depending on the source of the public key. The Ticket is a use-once authentication token used by a Gate. The class Ticket defines a method stampTicket (Role role) that checks that the ticket for the role is verified by the public key. The Capabilities model has four components: the client code, the gate, a credential checker and the capability object. The client code passes a token (this is of class Ticket) to the gate, which authenticates the client using the role and returns a capability object, also referred to as a Permit. (See Listing 1 to view the Permit User side code. Listing 2 shows the Gate side code.)

The gate is implemented using a pattern called a Gate, which is a specialized form of factory pattern method [GHJV 95].

The permit is based on a delegate pattern (see Design Patterns [GHJV 95]). It forwards calls to the actual implementation object. Both the permits and implementation are in the same package. The implementation, however, has only package private constructors. The permit objects need to be obtained by the client code to be useful.

The interface javax.commerce.base.WalletGate specifies methods for obtaining access to Wallet and Cassette Permits. The method getWalletUserPermit(Ticket tix) returns a WalletUserPermit if the ticket was created for a Role W_USER. The method getWalletAdminPermit(Ticket tix) returns a WalletAdminPermit if the ticket was created for a Role W_OWNER.

The interface javax.commerce.database.WalletUserPermit specifies methods for opening selected databases in OWNER or USER Roles. The method openDatabaseUserPermit(Ticket tix, String location, String dbName, String password, boolean makeBackup) returns a DatabaseUserPermit if tix was created for a role DATABASE_USER.

This should give you a good overview of the Gateway model. The purpose here is not to delve deeply into the internal mechanisms, but to give an idea of the main concepts involved and needed for this model.

I've mentioned Commerce beans previously in this article; let's take a quick look at what they are.

Commerce Beans

A commerce bean is a reusable commerce component that meets specific interface requirements. A commerce bean can be:

- An operation (purchase, ATM transfer, financial planning)
- A protocol (post, SET, Mondex...)
- An instrument (a credit card, a coupon, a voucher...)
- A service (account management, cassette management...)
- A preference (user preference configuration)

Commerce beans are contained within cassettes. When a cassette is installed, the JECF can make use of the commerce bean(s) it contains in order to perform commerce operations. For example, the JECF could use a purchase operation bean in conjunction with an instrument and protocol bean to perform an online purchase. Currently, commerce beans do not meet the JavaBean interface requirements; JavaSoft plans to bring them in conformance in a future release.

At this stage we've covered most of the concepts in JECF, but how do the various layers communicate with each other, i.e., what is the format for communication between servers, commerce beans and WalletUI?

The format is JCM, and I'll cover that in detail, as it is ubiquitous in JECF.

Java Commerce Messages (JCM)

JCM is a format for communication between Web servers and JECF and within the JECF. Communication in the JECF takes place between servers and JECF, between JECF and commerce beans, between commerce beans themselves and between commerce beans and the Java Wallet UI. JCMs supply the JECF with the information necessary to execute JECF operations. The JECF needs to know what operation the server is requesting. The operation bean might need to know what kind of instruments could be used for a particular operation, or which protocols it can use in conjunction with an instrument. On top of this, a user can select between different instruments accepted on a site. All of this information is communicated via JCM.

Structure of JCM

The JCM format is the language of operations within the JECF; a JCM is a list of name value pairs that describes some kind of transaction. Listing 3 shows part of a purchase JCM code.

The field operation=purchase in Figure 2 causes the JECF to look for a commerce bean that contains the purchase operation.

The field offer indicates that both of these items are presented as part of a single offer that expires 02/27/98 and bears id number ....

The lineItem fields describe individual items that are part of the offer.

A JCM is parsed by the JECF in the form of a labeled tree. Each branch has a name or number. Each leaf has a value named by the path from trunk to leaf. For each value a branch is created. Figure 3 shows a parsed JCM.

A JCM is organized into this structure by the JCM parser in the JECF. For example, if a transaction in the JECF is a purchase operation, then the JCM is sent to the purchase bean. The purchase bean (operation bean) contains a JCM parser, which reads the JCM as a tree.

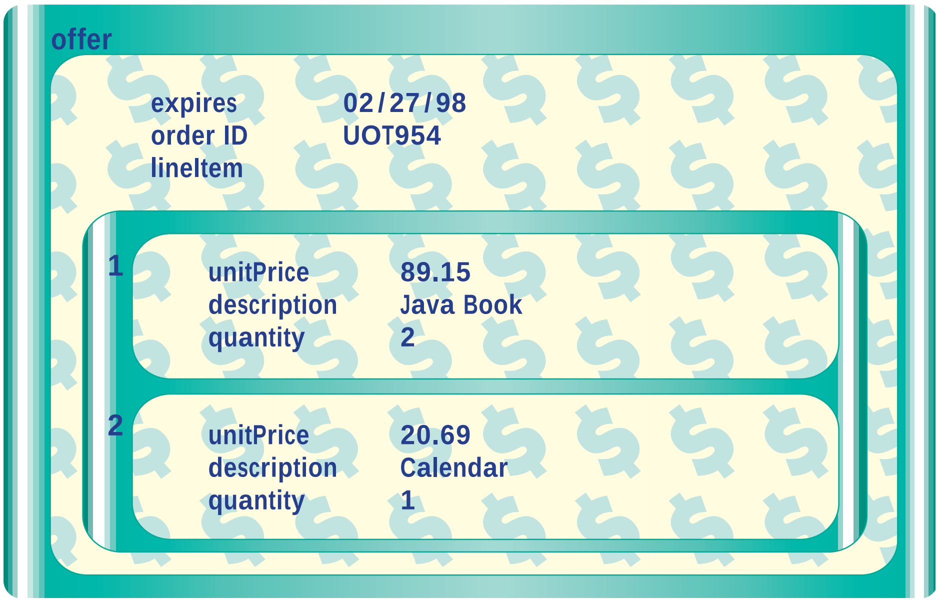


Figure 1: The purchase JCM as nested subdocuments

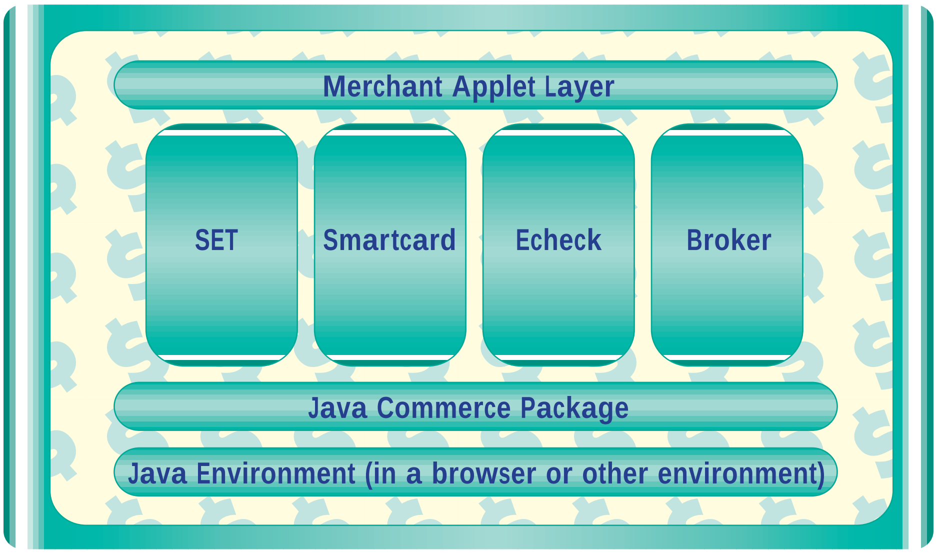


Figure 2: The layers in JECF

Operation, Protocol, Instrument

The key field in any JCM is the operation = field. This field dictates what other fields are required in a JCM. In many cases an operation requires that the JECF use instruments and protocols to carry out the operation. These instruments and protocols also dictate, farther down the tree, what fields are required in a JCM. When instruments and protocols are required, the valid field must be present followed by a subfield of either instruments or protocols:

```
valid.instruments = VISA
```

Dependencies for a given operation are identified through the requires field.

```
requires.cassette = buy
requires.protocol = Mondex
requires.instrument = VISA
```

This field is the complement of the valid field, which establishes the instruments that will be accepted on a site. The requires field establishes the cassettes the user must have installed to complete the transaction.

JCM Delivery

Servers that intend to send JCMs and browsers that intend to receive JCMs must, respectively, send and receive the appropriate MIME type header in order to open the necessary JCM handler. With respect to JECF, the handler is the Java WalletUI. The MIME type for a JCM is application/x-java-commerce. The file extension for JCM is .jcm.

Any Web site, Web server or application that is intended for use with the JECF must be capable of generating JCMs. JCMs can be generated in a number of ways, either statically (html hyperlink references a .jcm file embedded in a Web page) or dynamically (cgi scripts, applets and servelets).

Walk Through a JECF Application

Let's walk through what happens when a shopper makes a purchase from an online store. We assume both the server and the client browser are configured to handle JCMs.

We'll examine the interaction between a merchant server, The Online Computer Store, and a shopper, "Billy," who has a Web browser and a system configured with JECF, and is looking for a new computer. Billy visits the Online Store's Web site, which presents him with the various models offered. Each model has an image and a hyperlink that, when selected, loads a .jcm file. Listing 4 shows the .jcm file.

When Billy selects a link, the server sends a MIME header Billy's browser receives the MIME type .jcm, finds the handler (the Java Wallet), and invokes the handler by passing in the JCM. The JECF starts an operation thread and passes the JCM to this thread. This happens through a call to the method JECF.startOperation (JCM, AppletContext, TransactionListener). This call creates an instance of class OperationThread and invokes its run method. The operation thread performs a series of lookups. It looks into the JCM for the operation = field. It then looks in the JECF database for the operation cassette. If the database doesn't contain the cassette, then the thread checks the requires and locator fields in the JCM to see what cassette is required and where on the Web it can be found.

Let's say, in our case, the thread finds the operation cassette in the JECF database. Next, an instance of WalletCContext is created which holds context for that wallet. The thread then creates an instance of UICContext, passing it the WalletCContext, and then instantiates OperationCContext, passing it the UICContext. OperationCContext carries context relevant to that operation. It then instantiates the operation, passing it the OperationCContext object, and sets the JCM for this operation.

Next, the operation gets the UIFactory value from its context. If it's set to some preferred value, a check is made on all installed UI cassettes to see if any of them provide those services. If not found, JECF sets the UIFactory to DefaultUI and looks for a user interface associated with this operation. In this case the purchase operation has a ServiceUI consisting of a panel with an image inside it. The purchase operation implements Operation interface and ServiceUI interface. When the purchase operation was installed on Billy's computer, the method boolean canUseOperation(Operation op) was invoked on the installed WalletUI. This method checks the interfaces of the Operation and returns true if it can accommodate the Operation's UI requirements. In this case it returned a true value. The method addOperation(Operation op) was, subsequently, called, which in turn called addSelector(ServiceUI service). A resulting Selector Button was added to WalletUI, which, when clicked, shows the image panel.

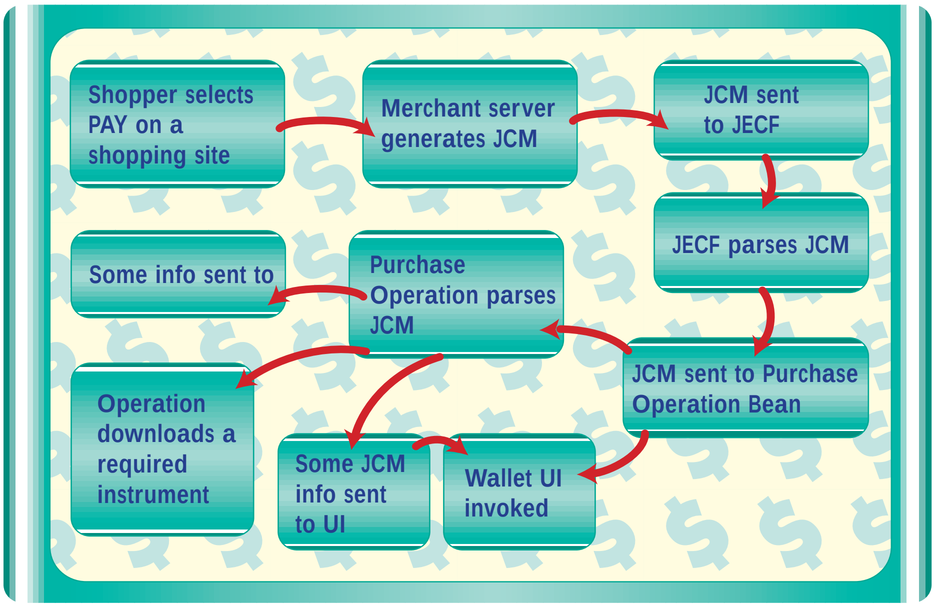

Shopper selects PAY on a shopping site
Merchant server generates JCM
JCM sent to JECF
JECF parses JCM
JCM sent to Purchase Operation Bean
Purchase Operation parses JCM
Some info sent to
Operation downloads a required instrument
Some JCM info sent to UI
Wallet UI invoked

Figure 3: The process involved in a purchase operation

The operation first makes a secure call into the Wallet to ascertain what instruments and protocols are accepted by the merchant site. The JECF takes the valid instruments and protocols listed in the JCM and compares them with the instruments and protocols registered in Billy's JECF database. The valid.instruments field in the JCM lists Mondex Card and Visa Cash. Billy's database contains Mondex card and Visa card. The same process takes place for protocols. The valid protocols on site are Mondex and Visa Cash. Billy's database contains Visa Cash, Post and Mondex.

Billy's wallet now appears on his screen. It shows a Pay button on the right side, and the instruments appear on the middle top of the screen. The protocols appear below the instruments, and the ServiceUI panel appears on the left side of the screen.

Billy selects the icon for Visa Card (instrument), chooses Mondex for protocol and then selects Pay in the WalletUI. The JECF gives the instrument to the protocol along with the portion of the JCM relevant to the protocol. In this example the Mondex protocol sends Billy's Visa information to the value acquirer. The amount of the purchase is deducted from Billy's card and placed in the Online Store's account. If everything works, the electronic receipt in Billy's Wallet UI is stamped PAID and he can dismiss the Wallet or continue shopping. If an applet sent the JCM, the applet is notified that the transaction is complete. Control is not returned to the caller until the user closes the Wallet or dismisses the purchase operation before completion.

Electronic commerce has its unique challenges, one of them the need to interoperate with a plethora of technologies, protocols and applications. JECF solves some of these problems through its concepts of cassettes. Since it's Java-based, a JECF-based application is portable (a significant benefit). With the diverse services that JECF provides, coupled with Java's strength as a portable, object-oriented, multithreaded and well-suited for Internet-based applications, this platform from JavaSoft will form the basis for the next wave of electronic commerce products, applications and frameworks.

Acknowledgements

JavaSoft Web site was the source of information on the JCM, Security Model and Architecture Layers.

References

The JavaSoft Web site, www.javasoft.com/products/commerce has the latest information on JECF, the class api documentation, papers on jcm, security model, WalletUIs and links to related sites.

CODE LISTING

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

About the Author

Mukul Sood is a systems architect with an e-commerce consulting firm. He has over six years of experience in designing and architecting 3-tier and n-tier applications. Currently he is working on a business-to-business e-commerce solution for a big computer retail chain. He can be reached at mukuls@digitalfocus.com.

mukuls@digitalfocus.com

Ad

Java at Fault?…Surely Not!

Assumptions and trust

by **Alan Williamson**

Come, friends, family and passersby, welcome to the start of a new column, from the good old keyboard of Alan Williamson. Some of you may have read my previous column under the banner name of 'Visual Café.' That column looked at various aspects of the Java language, including such goodies as POP and SMTP. This column is going to be somewhat different. I intend to strip away all of the media hype and marketing stories surrounding Java, and present you with a monthly look at the real Java: Java at the frontline. We will look at the problems facing developers on a daily basis: things like playing around with classpath's, shipping releases to other platforms and database drivers. If any of you have particular problems you have come across and successfully resolved, and you feel would be worth sharing, please e-mail me. Or even if you haven't solved them yet, e-mail me anyway. I am always interested in hearing about other people's problems…there is a strange comfort to be had knowing others are suffering equally as much!

Before we go ahead into our first subject, let me take the opportunity here to thank Dolly Parton. While the Managing Editor and myself where thrashing out the idea for this column we came up with several names for it; we eventually agreed on "Straight Talking," meaning to tell it like it is. A few days later, I remembered that dear Miss Parton took the lead role in a movie of the same name. In the movie, if memory serves me right, she was a radio deejay answering listener's problems. So you can think of me as your new Java-Dolly ready to tackle the problems others dare not address. With that, I think I have now professionally peaked! Miss Parton, I salute you.

On that note, on to this month's address: trust.

Trust is a marvelous thing. It is the one thing that makes living that much easier. In order to survive, we trust things will work. When we pick up the phone, we trust it will have a dial tone. When we get into our car, we trust it will start. When we develop code, we trust it will work (well, I know some pray as well, but theology is beyond the scope of this article). The point is the majority of us take the approach that if something isn't working quite right it must be our fault, and we must have missed something. It is perfectly natural to do this, especially if you're new to the language.

But I will let you in on a wee secret…Java has one or two bugs! Well to be precise, Java is harboring at least 5000 bugs. At least! To be honest, it's expected. Let's take a quick look at the history of Java. It was first released approximately 3 years ago with a handfull of classes, compared to the 1000 odd classes that are proposed in the new JDK1.2 specification. This is a phenomenal growth rate. So, some teething problems are to be expected. But should we tolerate them? If it where any other company, then the answer would be no; but, credit where credit is due, Sun is making the effort (looking past the fact that they created the problem in the first place with their eagerness in having Java become accepted in the mainstream development).

Let me illustrate this with a simple problem that had one of our engineers here at N-ARY tearing his hair out for several days. Lawrence had spent some time developing a complete servlet database solution, using HTML. This servlet hooked into an MS-Access database and basically allowed the insertion and viewing of a database that evolved around chronological dates and rooms. For example, a user could book a room for a given day, say the 9th of December, for 2 hours beginning at 1400 hrs. However, a problem was discovered at the end of the procedure when the final booking was e-mailed to the room coordinator; instead of the 9th of December, it booked for the day before, the 8th. A strange error indeed.

The first natural thing to do was to look at the way the dates where being stored in the database. In this particular instance we weren't actually storing the date but the number of milliseconds since 1970. This ensured that no date mangling would have occurred on the database side. So onto the next stage of debugging.

Lawrence thought it might have to do with the servlet processing and the actual storing of the data. He worked on the assumption that somewhere in the chain the day was starting from 0, as opposed to 1. But again, nothing unusual came up.

Upon closer inspection we discovered that the problem was one of the classes from the core JDK. The code snippet shown below illustrates the formatting function we used.

```
public static String getDate(long _date){
  SimpleDateFormat formatter  = new Simple-
DateFormat( "HH:mm:ss dd/MM/yy" );
  formatter.setTimeZone( TimeZone.getDe-
fault() );
  return formatter.format( new
java.util.Date(_date) );
}
```

The class, SimpleDateFormat, was the crux of the whole problem. Passing in our date, it incorrectly formatted the date, dropping a digit from the day value. This was a very difficult problem to find. Why?

Well, for starters, it is a common to assume/trust that the core JDK classes actually do what they say, and therefore you go off looking for the fault elsewhere. Only after a comical comment was made about the class was it investigated further, and sure enough it caught the culprit.

But is this an isolated case? Well, I am not going to advocate that you suddenly assume the JDK is wrong every time your class doesn't behave itself. The chances that a problem does exist with the core classes is rare. That said, a little known resource is available within the main Java Web site called the "Bug Parade." This forms part of the Java Developers Connection (JDC) located at http://java.sun.com/jdc. This section requires registration, but the good news is it's free.

This is the main area where bugs are submitted, and their present status can be viewed. For example, the particular bug number for the problem that we discovered is 4040985. The status for this bug can be viewed by directly entering the number and searching. If you do this you will see it has indeed been fixed, and has been updated in the 1.1.4 release of JDK. This information is particularly useful if you're experiencing problems and are unsure if upgrading your virtual machine will make any difference.

As a side point, when you install your new

JDK version, remember to have a quick scan over the "changes" file that is deposited in the route directory of the installation. This file is often overlooked but contains all the bug fixes that this release promises to fix. So keep an eye out for it.

So what should you do if you believe a problem does indeed exist? First of all, you should check to see if you're the first one to have experienced the problem. You can do this by first posting a question on the Java Usenet group, comp.lang.java.programmer. You might even find me lurking around there. Generally somebody will have experienced something similar. If not, then go to the JDC and extensively search the bug database to see if your problem is the same as or similar to an existing bug.

If you find a similar one, then add your experience to the bottom of it (assuming it hasn't already been fixed in a later release). The more help you can give the engineers the quicker the problem can be reproduced, and hopefully rectified.

If however you discover that no one else has had a similar experience, then fill out a new bug form. It will ask you a series of questions, and ask you to describe the nature of the problem in as much depth as possible. This is an extremely important step. Give them as much information as possible. But before you do, make sure you can repeat the bug yourself. If you can't repeat it then the chances of an engineer repeating it are slim. Also, include sample code where necessary.

One of the more interesting aspects of the bug parade is the ability to vote for bugs. I am not convinced of the value of this concept. Basically, you can vote for the bugs that you would like fixed first. This is meant to be a general guide for the engineers at Sun to determine which bugs are causing the most grief. So it's important to check for the existence of the bug before posting a new one. It is better to have 1 bug with 10 votes, than to have to 2 bugs that are the same with 5 votes.

Conceptually it's a great idea; let the users decide which bugs are annoying them the most. But the system fails for the bugs that only a few people discover. These are placed last on the list of priorities and it may be several releases later that they are addressed. Sun indicates that they only use the votes as a general guide, and they don't always fix the ones with the most votes first. I am not so sure. But that said, there have been a good amount of bugs highlighted and fixed.

It's an interesting approach to take: having nearly all of the Java community become your beta testers, and reporting back their problems and experiences. It's a bold maneuver that appears to be working. However, Java still has a lot of problems that need attention before we see it controlling medical equipment. But I think I would prefer to take my chances with a Java controlled heart monitor as oppose to a Microsoft NT solution.

Sun has a completely different approach to their developer community than say the likes of Microsoft. When we here at N-ARY had discovered a problem with the JDK we sent an email to Sun indicating our concern. They acted fantastically. They took our problem and worked with us to a successful conclusion. We can't ask for much more than that.

So the moral of the story? Don't believe everything you read on the computer screen. If a problem is still persisting and you've exhausted all avenues, then take a look at the classes your utilizing. Don't assume/trust that just because they are part of the core JDK that they work. Sometimes they don't. Don't be afraid to challenge the classes. Ask around, see if anyone else has experienced the problems. You may be surprised.

About the Author

Alan Williamson is on the board of directors at N-ARY Limited, a UK based Java software company specializing solely in JDBC and Java Servlets. He has recently completed his second book, focusing purely on Java Servlets. His first book looks at using Java/JDBC/Servlets to provide an efficient database solution. He can be reached at alan@n-ary.com. He welcomes all suggestions and comments.

alan@n-ary.com

Bristol 1/2 Ad

Create a Distributed Intranet Search Mechanism

Using Java Servlets

How to glue together Web and legacy systems across the corporate intranet using CGI-based applications

by **Eric Greenfeder**

Corporate intranets are heterogeneous environments comprised of Web servers and search engines from numerous vendors. In such a disparate environment, how do you create a corporate collection of indexed documents for use by a single search facility? One method is to use a catalog or index server, such as Netscape's Compass Server or Microsoft's Index Server. These products employ robots or agents that build collections of indexed documents by crawling through your company's intranet via URLs. While this method is very effective, it requires careful planning and administration. An alternate method is to write a Java servlet that connects to several search engines, compiling the results into a single document.

What Is a Java Servlet?

The Java servlet API was developed by Sun Microsystems to provide a mechanism for implementing Web server-side logic using the Java programming language. Java servlets are similar to CGI (Common Gateway Interface) programs in that they provide an HTTP-based mechanism for receiving user input and producing output in the form of HTML. Servlets offer a significant performance boost over CGI programs due to their architecture and implementation. CGI programs require the creation of a separate process to handle each client request. This approach consumes a significant amount of system resources and processing time for each client connection. Web servers implement the Java servlet specification load and instantiate any registered Java servlets upon start-up. Client HTTP requests are handled by creating a new thread within the Server's process space. Thus, servlets perform as if they were developed with native interface APIs such as NSAPI (Netscape Web Server) or ISAPI (Microsoft Web Server). Java servlets can be created easily by sub-classing the HttpServlet (javax.servlet.http.HttpServlet) class and overriding one of the HTTP processing methods.

The Java servlet architecture can be used to enhance the performance of database applications by implementing database connection pools. In traditional Web-based database applications, the overhead of connecting to a database is incurred during each client request. Servlets that implement database connection pools create a collection of instantiated database connection objects during servlet initialization. Therefore, the time required to create the objects and initialize the connections is incurred before any client HTTP requests are handled. Subsequent HTTP client requests are assigned one of the pooled database connection objects, reducing the required processing time. The size of the pool is set to the number of anticipated concurrent users. If that number exceeds the predetermined threshold, a new connection object is instantiated and added to the pool.

Architecture Overview

The search servlet combines the output from registered search engines by coordinating multiple threads that work in tandem to create a consolidated list of search results. This multithreaded approach increases the overall performance of the application by allowing the search engines to run concurrently. Each engine in the integrated search is registered via a URL entry in a configuration file (see Figure 1). The servlet strips the results from each search engine, ensuring that the most relevant results appear at the top of the final document (see Figure 2). To achieve a consistent look and feel across diverse search engine results, the output from each search engine is parsed to extract the HTML anchor tags. An HTML template file determines the final HTML representation of the results. Using a simple meta tag substitution, the template file alleviates the need to edit the servlet's Java code.

Servlet Design

The search servlet is comprised of three classes, each encapsulating a specific category of functionality into a reusable module. The HTMLAnchorParser class parses an input stream which extracts the HTML anchor tags

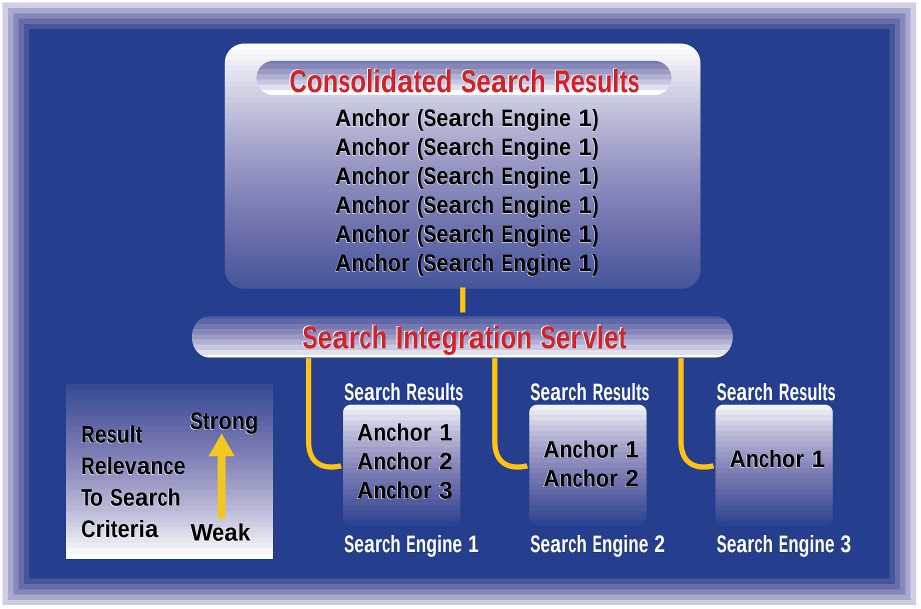


Figure 1: Search servlet architecture

DYNAMIC WEB APPS
COLD FUSION
DEVELOPER'S JOURNAL
Volume:1 Issue:1 November/October 1998
U.S. $6.99 (Canada $7.99)

From the Publisher
Fuat A. Kircaali pg. 5

Visual Programming
Advanced Programming with Studio
pg. 40

Advanced HTML
How to Use Drop-in Java Form Controls
by Joe Valley pg. 48

Case History
Booz Allen & Hamilton Deploys Cold Fusion Apps
by Bill Williams pg. 39

Tips & Techniques
Heavyweight Components for Thin Clients
by Brent Master pg. 50

Cold Fusion News
Allaire Acquires ICVERIFY Components for E-Commerce
pg. 45

Cold Fusion 3.1 –New Features: Corey Low
How to Build a Top Notch E-Commerce Application 8
Using Java forms, Mail, Transaction, HTTP, FTP and Schedule Tags

Fine Tuning Data Access in Cold Fusion Michael Green
Dynamic SQL Statements and Stored Procedures 14
Faster, more intelligent data access in performing database lookups

CFML: Cold Fusion Markup Language Steve Benfield 18
Tips, techniques and tricks in server-side programming

Cover Story: The Anatomy of a Startup Scott Davison
An Interview with J.J. and Jeremy Allaire 28
"Java was our competitors' worst distraction in Silicon Valley..."

HomeSite 3.0 Corner: Web Craftsmanship... David Bell 36
How to fine-tune your hand-crafted Web site across platforms

Extending Cold Fusion with ICVERIFY: Robert Diamond
Exploring the Allaire Tag Gallery 42
Creating a sample application with tag-based programming

SYS-CON PUBLICATIONS

ADVERTISER INDEX

Advertiser			Page
Accusoft	www.accusoft.com	206 329-7460	19
Bristol Technology	www.bristol.com	203 438-6969	23
Cold Fusion Developer's Journal	www.sys-con.com	914 735-1900	26
Data Representations	www.datarepresentations.com	888 307-9550	29
Halcyon	www.halcyonsoft.com	888 333-8820	35
IBM	www.ibm.com	800 426-5900	58&59
IEC-EXPO	www.iec-expo.com	888 222-8734	73
ILOG	www.ilog.com	415 688-0200	17
Installshield	www.installshield.com	800 374-4353	13
Inno Val	www.innoval.com	914 835-3838	38
Keo Group	www.keo.com	978 463-5900	22&37
KL Group Inc.	www.klg.com	800 663-4723	B/C
Live Software	info@livesoftware.com	619 643-1919	41
Net Dynamics	www.netdynamics.com	650 462-7600	79
ObjectShare	www.objectshare.com	800 973-4777	43
Object Matter	www.objectmatter.com	305 718-9101	50
ObjectSpace	www.objectspace.com	972 726-4100	4
Object Management Group	www.omg.org	508 820-4300	53
Progress/Cohn & Godly	www.apptivity.com	800 477-6473	21
ProtoView	www.protoview.com	609 655-5000	3
Roguewave	www.roguewave.com	800-487-3217	15
Sales Vision	www.salesvision.com	704 567-9111	47
Silverstream	www.silverstream.com	888 823-9700	83
Sockem Software	www.sockem.com	814 696-3715	65
Stingray Software Inc.	www.stingsoft.com	800 924-4223	2
SunTest	www.suntest.com	415 336-2005	11
Sybex Books	www.sybex.com	510 523-8233	63
The Object People	www.objectpeople.com	919 852-2200	23
SYS-CON Publications	www.sys-con.com	800 513-7111	71
Thought, Inc.	www.thought.com	415 836-9199	48
Visionary Solutions, Inc.	www.visolu.com	215 342-7185	50
WebMethod	www.wbmethods.com	888 831-0808	33
Zero G. Software	www.zerog.com	415 512-7771	6

and stores them in an internal data structure. The SearchEngine class connects to a search engine via a URL, initiates a search request and stores the results. It extends HTMLAnchorParser to provide parsing functionality for the search engine results. This class also implements the java.lang.Runnable interface to support concurrent processing via threads. Lastly, the SearchManager class coordinates all search engine threads and handles the HTTP requests from the Web Server by extending the HttpServlet class in the Java servlet package (javax.servlet.http). The doGet method is overridden to implement the necessary HTTP request handling. Figure 3 depicts the class diagram for the servlet designed using Rational Rose for Java. Rational Rose is an object-oriented analysis and design tool that provides powerful design and code generation functionality for Java development. Designing the system in a modular fashion such as this promotes reuse and decreases the time required for development and testing.

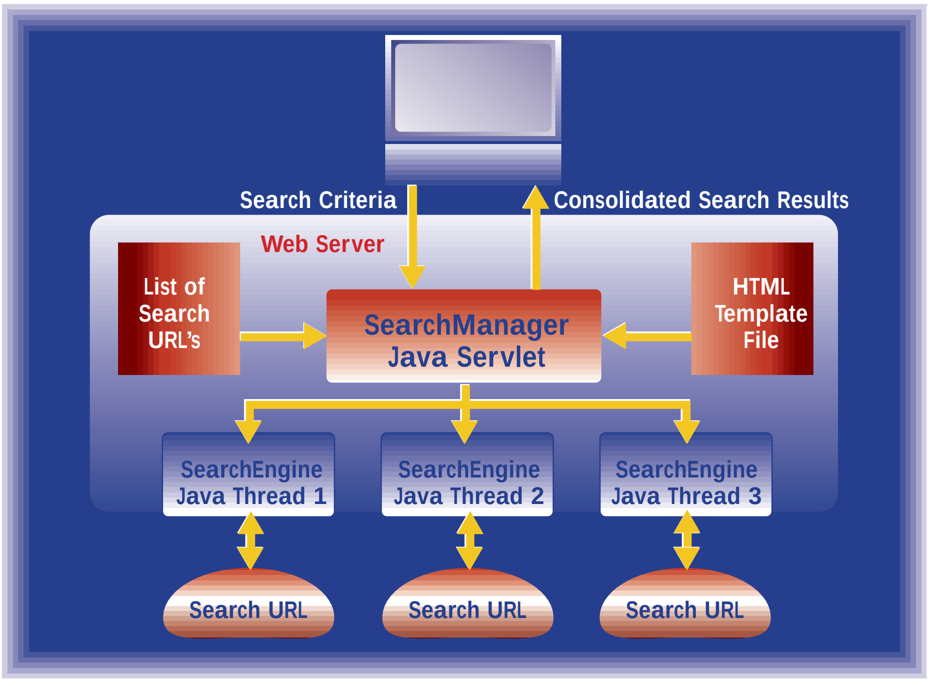

Search Criteria
Consolidated Search Results
Web Server
List of Search URL's
SearchManager Java Servlet
HTML Template File
SearchEngine Java Thread 1
SearchEngine Java Thread 2
SearchEngine Java Thread 3
Search URL
Search URL
Search URL

Figure 2: Search results integration

Implementing the Servlet

Listing 1 contains the source code for the SearchManager class. HTTP GET requests are handled by overriding the doGet method of the inherited HttpServlet class. When this method is invoked, it stores the output stream from the HttpServletRequest object using the getOutputStream method. Sending output back to the client's browser is simplified by redirecting System.out and System.err to this output stream. This is accomplished by invoking the setOut and setErr methods of the System class. Subsequent calls to System.out.println and System.err.println will cause the output to be sent to the client's browser. The HTTP Response header's content type field is set to "text/html" via a call to the setContentType method. Without this call, the browser would not know how to interpret the information being sent back from the servlet. The search servlet accepts a single CGI parameter that contains the criteria for the integrated search. The getParameter method of the HttpServletRequest class is used to access this variable by name. In Listing 1 you will notice that the doGet method invokes an initialize method. This method reads the search engine configuration file and creates a thread for each entry. A thread group is used to monitor the status of the search engine threads and determine when they have finished processing. This is implemented using a busy-wait loop and the ThreadGroup.activeCount and Thread.sleep methods. The rest of the doGet method processes the HTML template file and outputs the search engine results when the "<<results>>" meta tag is encountered. The results from the search engines are stripped by outputting a single line from each search engine and removing the search engine when the results have been exhausted.

Listing 2 contains the source code for the HTMLAnchorParser

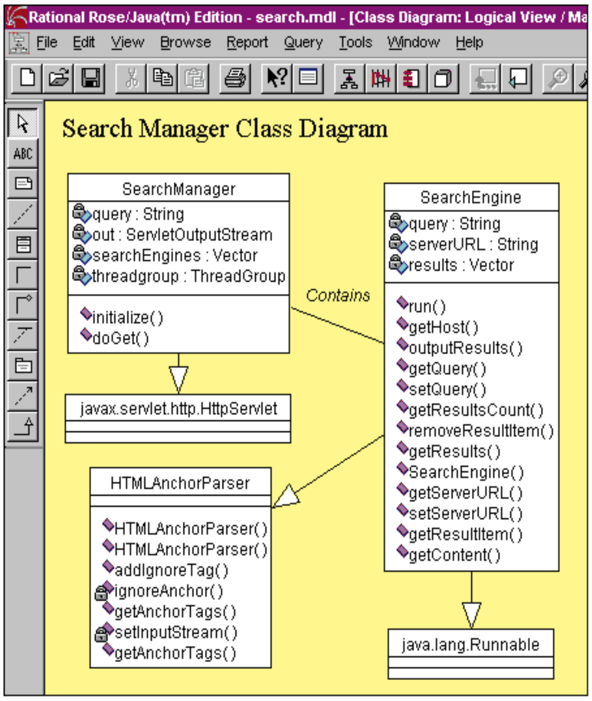


Figure 3: Search servlet class diagram

Listing 1: Search Manager class.

```
import java.io.*;
import java.util.*;
import java.lang.*;
import javax.servlet.*;
import javax.servlet.http.*;

// THIS CLASS IS USED TO MANAGE THE SearchEngine THREADS
// AND PROVIDE JAVA SERVLET FUNCTIONALITY

public class SearchManager extends HttpServlet {
    private ServletOutputStream out; // THE OUTPUTSTREAM BACK TO THE
CLIENT'S BROWSER
    private String query=null;     // THE QUERY CRITERIA TO USE IN THE
SEARCH
    private Vector searchEngines = new Vector(); // A LIST OF
SearchEngine OBJECTS
    // A ThreadGroup TO MANAGE THE SearchEngine THREADS
    private ThreadGroup threadgroup = new ThreadGroup("searchEngines");

    // doGet PROCESSES HTTP GET REQUESTS FROM CLIENT CONNECTIONS
    public void doGet (HttpServletRequest req,
                       HttpServletResponse res)
                       throws ServletException, IOException {
      int cnt = 1,index=0;
      String link,inline; // BUFFER VARIABLES
      String currenthost=null; // THE HOST NAME OF THE CURRENT
SearchEngine
      SearchEngine currentse=null; // CURRENT SearchEngine OBJECT
```

class, which provides methods for reading and parsing the HTML anchor tags from a supplied java.io.InputStream object. The ignoreAnchor method provides a mechanism for ignoring HTML anchor tags that contain certain string patterns. This enables the servlet to ignore irrelevant and/or ornamental anchor tags that are generated by various search engines. A call is made to Thread.currentThread().yield(), while this class reads data from the given input stream to yield the CPU to other competing threads. Without this method call, a single thread would dominate the CPU.

The SearchEngine class (see Listing 3) inherits the HTML anchor parsing capabilities of the HTMLAnchorParser class while implementing the java.lang.Runnable interface to provide multithreaded support. This class creates a URLConnection instance from a specified URL, passing the associated InputStream to the HTMLAnchorParser.getContent method to return and store a list of HTML anchor tags.

The SearchManager strips the output from the SearchEngine instances using calls to the getResultItem and removeResultItem methods. Once a result is read it is removed from the associated SearchEngine object, making the next result available for the subsequent pass. The getResultsCount method is used to determine whether all the results have been read from a specific SearchEngine object and whether it can be ignored in subsequent stripping passes (see Figure 2).

Conclusion

The java.net package provides a feature-rich set of classes for interfacing and controlling HTTP based resources. Coupled with the Java servlet API, developers can easily create traditional CGI-based applications that support socket programming, HTTP handling and multithreaded capabilities that glue together Web and legacy systems across the corporate intranet.

Resources

- Sun Microsystem's Servlet Tutorial: http://java.sun.com/products/jdk/1.2/docs/ext/servlet/servlet-tutorial.html.
- Sun Microsystem's "Inside the Java Web Server:" http://java.sun.com/features/1997/aug/jws1.html.

About the Author

Eric Greenfeder currently works for the BASF Corporation as a Senior Internet Architect. He specializes in Java development, Web security, CORBA and object-oriented analysis and design. Eric has been programming in Java since 1996 and currently teaches an internal Java course to BASF developers. You can reach him by e-mail at greenfe@basf.com.

greenfe@basf.com

```
      Enumeration se=null;

      // GET THE RESPONE OUTPUT STREAM AND REDIRECT THE out
      // AND err OUTPUTSTREAMS TO RETURN RESULTS TO THE CLIENT
      // CONNECTIONS
      out = res.getOutputStream();

      // REDIRECT ERRORS TO THE CLIENT'S BROWSER
      System.setErr(new PrintStream(out));

      // REDIRECT STANDARD OUTPUT TO THE CLIENT'S BROWSER
      System.setOut(new PrintStream(out));

      // SET THE RESPONSE CONTENT TO HTML TEXT
      res.setContentType("text/html");

      // GET THE QUERY PARAMETER
      try {
        query = req.getParameter("query");
       if(query == null || query.length() == 0) {
          out.println("<H1 ALIGN=CENTER>Please input a "+
                      "search string !</H1>\n");
          out.println("</BODY></HTML>\n");
          out.flush();
          return;
        }
      }
      catch (Exception e) {
        System.err.println("SearchManager (doGet): "+e);
        System.err.flush(); }

      // INITIALIZE THE SEARCH ENGINE THREADS
      if(!initialize()) return;

      try {
      // READ THE TEMPLATE USED TO FORMAT THE SEARCH RESULTS
      BufferedReader in =
          new BufferedReader(new
FileReader(System.getProperty("user.dir")+
                                            "/searchservlet.pat"));
      // READ IN THE TEMPLATE FILE UNTIL THE <<results>>
      // TAG IS FOUND, THEN WRITE OUT THE RESULTS. LOOP ACCROSS
      // EACH SEARCH ENGINE GATHERING THE RESULTS FROM EACH SEARCH
      // ENGINE ONE RESULT AT A TIME. THIS WAY THE MORE PERTINENT
      // SEARCH RESULTS FROM EACH ENGINE WILL APPEAR AT THE TOP OF
      // THE DOCUMENT.
      while((inline=in.readLine()) != null) {
        if((index=inline.toLowerCase().indexOf("<<results>>")) > -1) {
          System.out.println(inline.substring(0,index));
```

```
int exhaustedEngines=0;
while(searchEngines.size() > exhaustedEngines) {
  exhaustedEngines=0;
  // if all the engines have been exhausted then this
  // variable will = searchEngines.size()
  for(int i=0;i<searchEngines.size();i++) {
    currentse = (SearchEngine)searchEngines.elementAt(i);
    currenthost = currentse.getHost();
    // THIS SEARCH ENGINES RESULTS HAVE BEEN EXHAUSTED
    if(currentse.getResultsCount() == 0) {
      ++exhaustedEngines;
      continue;
    }
    try {
      // get the first element
      link = currentse.getResultItem(0);
      ((SearchEngine) searchEngines.elementAt(
        searchEngines.indexOf(currentse))).removeResultItem(0);
    }
    catch (ArrayIndexOutOfBoundsException e) {
        ++exhaustedEngines;
        continue;
    }

    // WE DO NOT WANT IMAGES AND BLANK ANCHORS
    if(link.toLowerCase().indexOf("http://") == -1) {
      int idx; // SCRATCH VARIABLE
      if((idx = link.toLowerCase().indexOf("href=\"")) > -1) {
        if(link.toLowerCase().indexOf("href=\"/") > -1)
          link = link.substring(0,idx+6)+currenthost+
                 link.substring(idx+6);
        else
          link = link.substring(0,idx+6)+currenthost+"/"+
                 link.substring(idx+6);
      }
      else {
        idx = link.toLowerCase().indexOf("href=");
        if(link.charAt(idx+6) == '/')
          link = link.substring(0,idx+5)+currenthost+
                 link.substring(idx+5);
        else
          link = link.substring(0,idx+5)+currenthost+"/"+
                 link.substring(idx+5);
      }
    }
    out.println(String.valueOf(cnt++)+". "+link+"<BR>");
  }
}
// PRINT OUT THE REST OF THE LINE AFTER THE
```

Ad

```
        // <<results>> META TAG
        System.out.println(inline.substring(index+11));
      }
      else {
        System.out.println(inline);
      }
     }
   } catch (Exception e) { System.err.println("SearchManager (doGet): "+e); }
    out.flush();
    out.close();
  }
  /* THIS METHOD INITIALIZES AND RUNS THE SEARCH ENGINES
   AND WAITS FOR ALL SEARCH ENGINE THREADS TO COMPLETE.
   THE FILE searchurls.conf IS READ FROM THE CURRENT WORKING
   DIRECTORY TO PROVIDE A LIST OF SEARCH ENGINE URL'S. THE
   URLS LISTED IN THE CONFIGURATION FILE MUST END WITH THE
   SEARCH ENGINE'S QUERY PARAMETER FOLLOWED BY AN EQUAL SIGN.
   example:  www.search.com/cgi-bin/search?query=</B> */

  public boolean initialize() {
    String inline;             // A SCRATCH VARIABLE
    String servlet_dir=null;  // THE SERVLET HOME DIRECTORY
    SearchEngine se=null;     // A SearchEngine OBJECT

    try {
      // GET THE SERVLET HOME DIRECTORY
      servlet_dir = System.getProperty("user.dir");

      // OPEN AN INPUT STREAM TO THE CONFIGURATION FILE THAT
      // LISTS THE URL'S OF THE SEARCH ENGINES TO BE INTEGRATED
      BufferedReader in =
        new BufferedReader(new FileReader(servlet_dir.replace('\\','/')+
        (servlet_dir.charAt(servlet_dir.length()-1) ==
        '/'?"searchurls.conf":"/searchurls.conf")));

      // INITIALIZE THE LIST OF SEARCH ENGINES
      searchEngines.removeAllElements();

      // READ IN THE URL'S FROM THE searchurls.conf FILE
      while((inline=in.readLine()) != null) {
        // CREATE A SEARCH ENGINE INSTANCE USING THE URL
        // READ IN FROM THE searchurls.conf FILE AND
        // CONCATENATE THE USER SUPPLIED SEARCH CRITERIA
        // NOTE: THE URL LISTED IN THE CONFIGURATION FILE
        // MUST END WITH THE SEARCH ENGINES QUERY PARAMETER
        // FOLLOWED BY AN EQUAL SIGN.
        // example:  www.search.com/cgi-bin/search?query=

        se = new SearchEngine(inline.trim()+query);
        se.addIgnoreTag("<img");
        se.addIgnoreTag("<IMG");
        se.addIgnoreTag(">_<");
        searchEngines.addElement(se);
        // START THE SEARCH ENGINE THREAD
        new Thread(threadgroup,se).start();
      }

      // WAIT FOR ALL OF THE SEARCH THREADS TO FINISH PROCESSING
      while(threadgroup.activeCount() > 0)
        Thread.currentThread().sleep(50);

    } catch (Exception e) {
      System.err.println("initialize: "+e); System.err.flush();
      return false; }
    return true;
  }
}
```

CODE LISTING

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

1/4 Ad

1/4 Ad

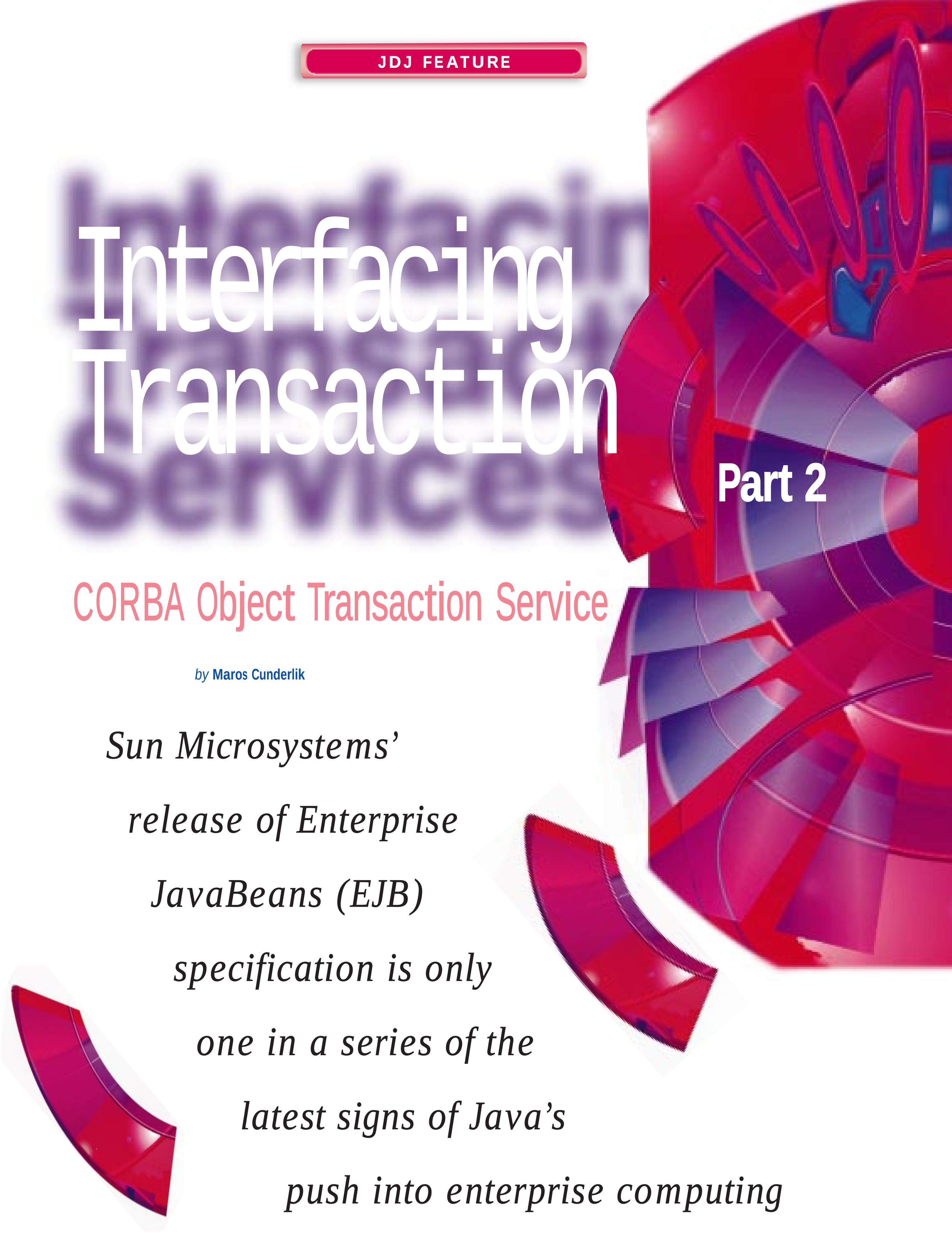
Interfacing Transaction Services

Part 2

CORBA Object Transaction Service

by Maros Cunderlik

Sun Microsystems' release of Enterprise JavaBeans (EJB) specification is only one in a series of the latest signs of Java's push into enterprise computing



Undoubtedly, the support for distributed transactions is a part of any enterprise system. Part One of this series (JDJ, March 1998) explored the X/Open Distributed Transaction Processing (DTP) Model – a common model for distributed transaction processing. We also explained the concepts of Microsoft Transaction Server (MTS) that quickly become de facto standard for developing business components on the Windows platform. We demonstrated how to write "first-class" MTS COM components in Java. The described process of creating these components has been greatly simplified by the release of Microsoft Visual J++ 6.0.

In this second part we will take a closer look at CORBA Object Transaction Service (OTS). As Java gains wider acceptance in enterprise computing, we are likely to see a number of simplified object models and API layers that will rely on OTS to provide transaction support (e.g., EJB, Java Transaction Service, IBM's San Francisco Project, etc.). These models greatly reduce the amount of code necessary to create applications with full transaction support. Nevertheless, as the number of components in the system increases, the complexities of real-world systems are often encountered. These include managing object interdependencies, state and context management, and dealing with various hardware and network limitations. To make the situation worse, the problems often go undetected until the deployment time because of insufficient prototype evaluation and stress testing. In solving these issues, an understanding of the underlying CORBA specification and implementations is critical. Therefore, we will describe in this article the fundamentals of CORBA OTS specification as defined by Object Management Group (OMG). We will also show some details of CORBA OTS commercial implementations and discuss the relationship between CORBA OTS and EJB.

CORBA Object Transaction Service Specification

CORBA OTS specification is loosely based on X/Open DTP Model. The specification defines a model that complies with ACID transaction properties: atomicity, consistency, isolation and durability. The specification also describes the set of standard interfaces and behaviors that define how a client, the Transaction Service and the transactional objects participate in transaction processing. Figure 1 illustrates the main entities of CORBA OTS.

In simple view, the client starts a transaction by contacting the Transaction Service. The service creates a transaction context for the new transaction. The client then makes a series of requests on transactional and nontransactional objects. The transactional objects that maintain durable data and state are called recoverable objects. These objects must register with the Transaction Service using a resource object. The resource object participates in coordinating a two-phase commit process, and is used to access the underlying resources consumed by the objects. When finished, the client commits or rolls back the changes by instructing the Transactional Service to commit or roll back, respectively. The service then coordinates the two-phase commit process with all resources registered within the scope of the current transaction. Listing 1 shows the definitions of all OTS-defined data types and interfaces in Interface Definition Language (IDL).

The CORBA OTS specification defines two basic approaches to carrying out transaction processing. These are based on the level of involvement in the context management and transaction propagation. The context management refers to the process of the transaction context creation, initialization and control. In general, the participants can choose the indirect context management and rely on the Transaction Service to perform all context-related responsibilities. On the other hand, the client can elect to manage the transaction context directly by using the set of standard OTS interfaces. Similarly, when creating transactional objects, the client can use the Transaction Service implicit propagation, or explicitly pass the transaction context to the objects. Clearly, there are four possible combinations of how to perform the context management and transaction propagation:

- Indirect context management and implicit propagation
- Indirect context management and explicit propagation
- Direct context management and implicit propagation
- Direct context management and explicit propagation

While all of these combinations are valid, the ones described in the following sections are the most common. These two combinations reflect the fundamental design tradeoff between ease of use and flexibility.

Indirect Context Management and Implicit Transaction Propagation

When using the indirect context management and the implicit transaction propagation, we rely on the Transaction Service to perform all context-related responsibilities. This model requires the simplest code by utilizing a predefined standard pseudo object Current. The client uses the Current object to begin, commit or roll back the transaction. The client also instantiates and

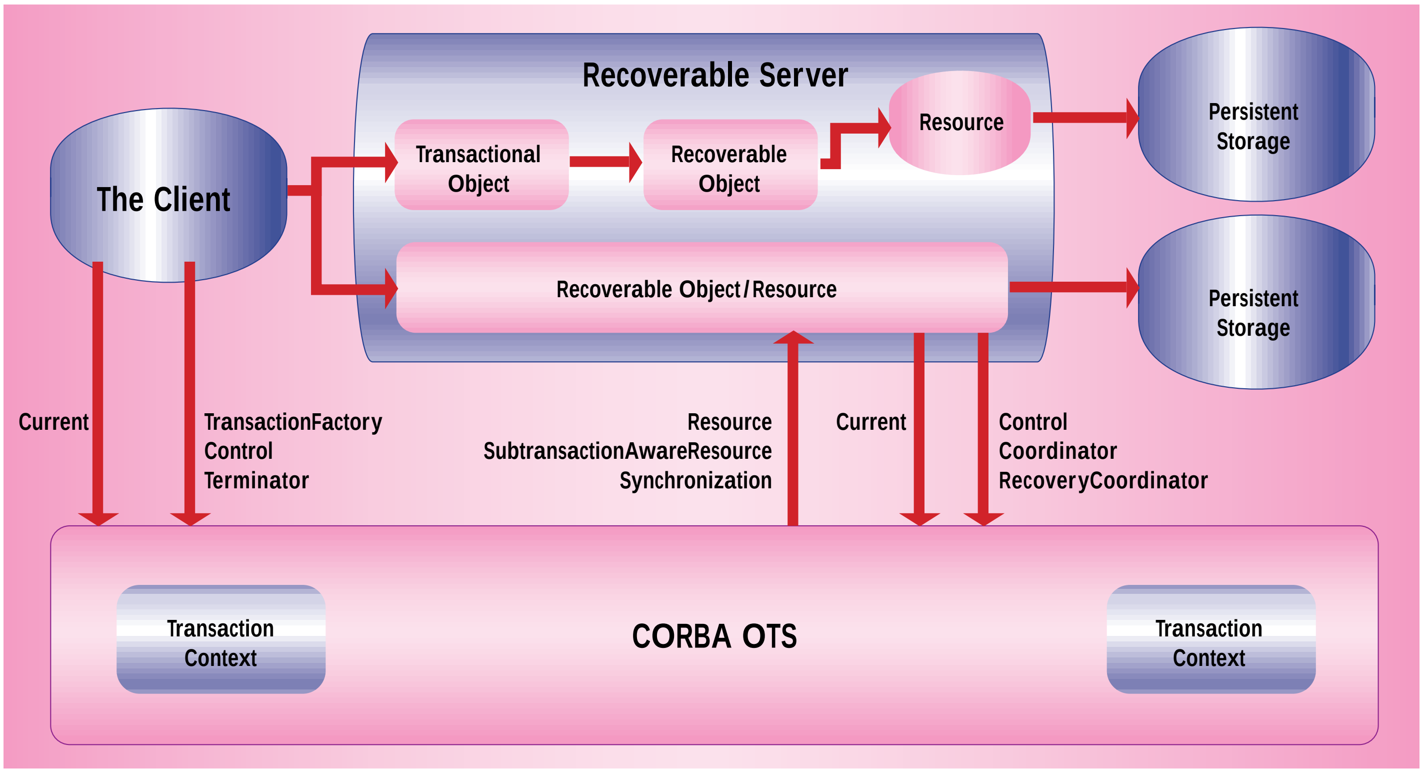


Figure 1

uses the recoverable objects as any other CORBA objects. Upon object creation, the transaction context is automatically associated with the server thread by the Transaction Service. The recoverable objects can be marked as "transactional" at design time by implementing the TransactionalObject interface. This interface does not have any methods -- it simply signals that all of the object's methods are transactional, and the service should propagate the transaction context to the object implicitly. The server object can also implement the Resource interface. This indicates that the object is recoverable and it is capable of participating in the two-phase commit process.

Figure 2 depicts the typical scenario in which both the client and the recoverable object use the Current pseudo objects. The client starts the transaction by calling Current.begin() method. It then performs the series of operations on the server objects. When finished, the client issues Current.commit() or Current.rollback() commands to commit or roll back the current transaction, respectively. On the server side, the recoverable object retrieves the Control object by calling Current.get_control(). The Control object in turn is used to obtain the Coordinator object. The Coordinator object provides an access to the single transaction information such as the transaction name, transaction status, parent transaction and the propagation context. The recoverable objects use the Coordinator object to register its resources with the current transaction by calling Coordinator.register_resource(), which returns the RecoveryCoordinator object. The resources use this object when recovering from the failure after being previously prepared. Finally, the recoverable object carries out the client's request.

At the client's request, the Transaction Service commits the changes through the two-phase commit process with all registered resources. When the client requests to commit the transaction, the Coordinator issues the Resource.prepare() command. The resources then prepare to commit and return the Vote object accordingly: VoteCommit, VoteRollback or VoteReadOnly. The VoteReadOnly indicates that no changes were made to the underlying persistent data managed by the resource. If all registered resources voted VoteReadOnly, the Coordinator informs the client about the successful commit. If at least one resource replied VoteCommit and the rest of the registered resources voted VoteReadOnly, the Coordinator first saves the current transaction information in case of later failure, and then completes the two-phase commit process by calling Resource.commit().

Direct Context Management and Explicit Transaction Propagation

The direct context management allows the clients to access the transaction context directly by using TransactionFactory, Control and Terminator interfaces. The client starts the new transaction by calling TransactionFactory.create() method. This call returns the Control object, which is used to retrieve references to the Terminator and Coordinator objects. The client subsequently performs calls on transactional objects. The client explicitly passes the reference to the Control object as a method parameter. To finish the current transaction, the client uses the Terminator.commit() method. Upon the request to commit, the OTS will initiate the two-phase commit process with all of the registered resources as described in the previous section.

The server objects do not have to implement TransactionalObject interface. Instead, these objects receive the reference to the Control object as an explicit parameter for each method. This allows the same object to have both transactional and non-transactional methods. All transactional methods must use the passed-in Control object to retrieve the reference to the Coordinator object and register all their resources. Figure 3 illustrates how the client and the recoverable server engage in transaction processing using the direct context management and explicit transaction propagation.

Nested Transactions and X/Open Interoperability

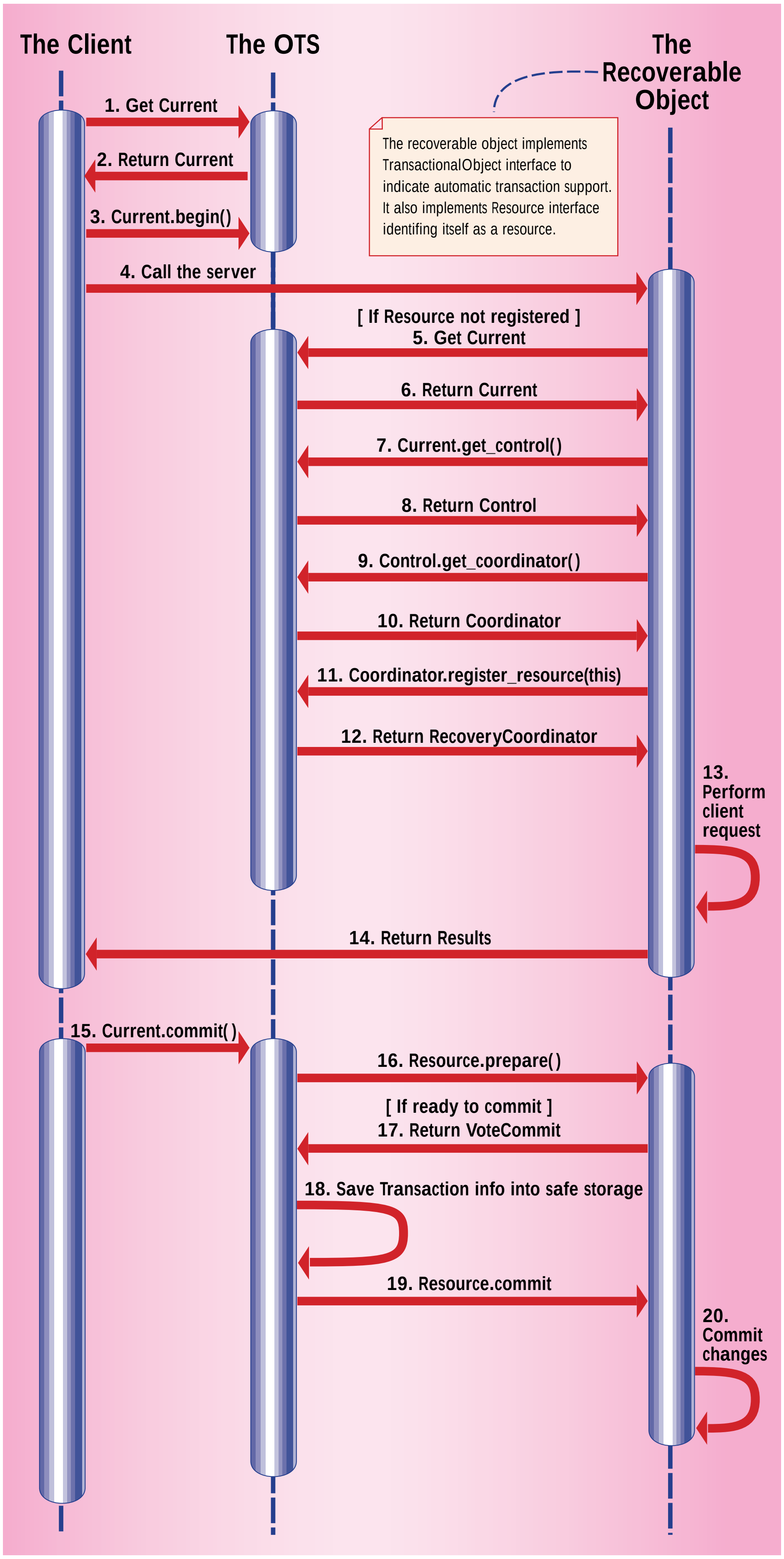
Although the X/Open DTP model does not define the nested transactions, the Transaction Service supports this model of transaction processing. The nested transactions are implemented by establishing a parent-child relationship among transactions and thus building so-called transaction families. The transaction family comprises one top-level transaction and one or more subtransactions. The subtransactions can be nested. A transaction will not commit until all subtransactions are committed. Similarly, the changes committed by the subtransactions could be rolled back by ancestor transactions. The nested transactions prove useful in optimizing system performance. By isolating failures in subtransactions, rollbacks and resource recovery are kept to minimum.

The specification also establishes guidelines for integrating with X/Open compliant systems. In particular, transactions can be imported and exported across the two models. The OTS implementations should allow XA-compliant resource managers to participate directly in OTS transactions. The OTS specification explicitly allows a single transaction service to support both X/Open DTP and OTS models. This decision not to engage in “purist” arguments will certainly prove critical to the specification's success with vendors and customers. The transaction monitor vendors can leverage their experience with X/Open and incrementally add OTS support onto the existing products (e.g., IBM TXSeries Transaction Server). On the other hand, users can easily integrate their “legacy” systems with the new CORBA-compliant applications.

CORBA OTS Implementations

While the service specification defines a set of standard interfaces and scenarios, ORB vendors provide the actual implementations of the service. Contrary to popular belief, the common specification and language mapping do not necessarily lead to interchangeable implementations. While this allows vendors to differentiate their products, it also causes significant problems when porting server objects across different ORBs. Object Management Group recognized the problem and defined the set of interfaces for porting the Transaction Service across various ORBs (CosTSPortability module). The service portability is achieved by defining specific interfaces that lie between the ORB and the Transaction Service. While this model virtually guarantees the portability of the service, using the same vendor’s ORB and CORBAservices is still the safest and most practical decision.

From the user’s standpoint, implementing CORBA solutions can be greatly simplified by leveraging a complete CORBA “suite” that usually allows for easy integration of CORBAservices, transaction monitor, server utilities, common IDE and possibly a set of classes for simpler client-side programming. For example, in addition to the implementation of standard CORBAser-



The Client
The OTS
The Recoverable Object
The recoverable object implements TransactionalObject interface to indicate automatic transaction support. It also implements Resource interface identifing itself as a resource.
1. Get Current
2. Return Current
3. Current.begin()
4. Call the server
[If Resource not registered] 5. Get Current
6. Return Current
7. Current.get_control()
8. Return Control
9. Control.get_coordinator()
10. Return Coordinator
11. Coordinator.register_resource(this)
12. Return RecoveryCoordinator
13. Perform client request
14. Return Results
15. Current.commit( )
16. Resource.prepare()
[If ready to commit] 17. Return VoteCommit
18. Save Transaction info into safe storage
19. Resource.commit
20. Commit changes

Figure: 2

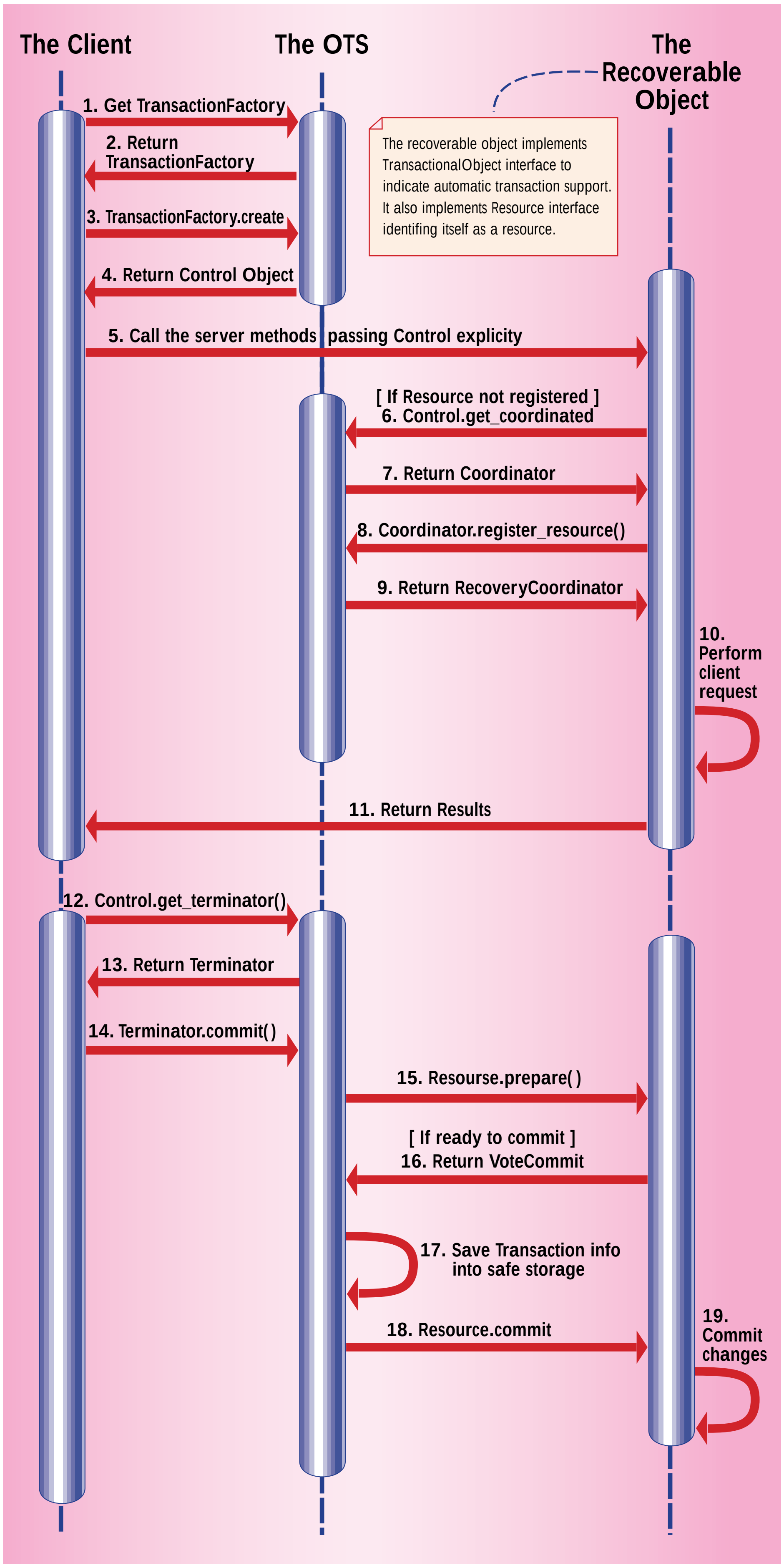

The Client
The OTS
The Recoverable Object
The recoverable object implements TransactionalObject interface to indicate automatic transaction support. It also implements Resource interface identifing itself as a resource.
1. Get TransactionFactory
2. Return TransactionFactory
3. TransactionFactory.create
4. Return Control Object
5. Call the server methods passing Control explicity
[If Resource not registered]
6. Control.get_coordinated
7. Return Coordinator
8. Coordinator.register_resource()
9. Return RecoveryCoordinator
10. Perform client request
11. Return Results
12. Control.get_terminator()
13. Return Terminator
14. Terminator.commit()
15. Resourse.prepare()
[If ready to commit]
16. Return VoteCommit
17. Save Transaction info into safe storage
18. Resource.commit
19. Commit changes

Figure: 3

vices, Iona's Orbix Object Transaction Monitor (OrbixOTM) also offers a set of GUI development tools, server-side manager with SMTP support, fault "resilience" and server load balancing. As CORBA matures, we are likely to see more of these complete-suite products, which will help to push CORBA further into the enterprise mainstream.

On the Java side, OMG provides a standard Java mapping of IDL data types, modules and other constructs. It also offers details on server-side mapping of transient and persistent objects, Java ORB portability and mapping of CORBA pseudo objects into Java. JDK 1.2 contains a great example of this mapping -- an implementation of CORBA 2.0-compliant ORB and CORBA Naming service.

In addition to JDK 1.2, the release of the Java Transaction Service (JTS) and Enterprise JavaBeans (EJB) specifications further strengthened the bond between CORBA and Java. These technologies also helped to clear up some of the confusion over the relationship of CORBA and the Java platform.

EJB was designed to provide a high-level component architecture for writing business applications. In general, the enterprise bean object can use automatic declarative transaction management by specifying a transaction attribute. The bean container interposes all client requests, and the container is then responsible for managing the transaction according to the transaction attribute. EJB defines valid values for the bean transaction attribute (TX_NOT_SUPPORTED, TX_BEAN_MANAGED, TX_REQUIRED, TX_SUPPORTS, and TX_REQUIRES_NEW) that are very similar to Microsoft MTS transaction attributes. For example, the bean marked TX_REQUIRED will execute in the client's transaction context, if the client has one. Otherwise, the new transaction is automatically created. The bean can also choose to use manual transaction management by specifying TX_BEAN_MANAGED attribute. Under this scenario, the clients and the beans use the JTS javax.jts.UserTransaction interface (see Listing 2) to directly manage the transaction. In fact, UserTransaction interface is the only JTS interface that the bean container must provide to be EJB compliant.

The JTS specification defines a standard transaction management API for the Java platform. It includes org.omg.CosTransactions and org.omg.CosTSPortability packages that contain the standard Java mapping of the CORBA OTS modules. As mentioned before, the JTS also defines a high-level application transaction "demarcation" API (javax.jts package). Since this API provides an isolation layer between the enterprise beans and the service implementation, in theory any transaction manager exposing this API can participate in the EJB model.

Conclusion

In order to develop enterprise business components in Java, we need to provide support for distributed transaction processing. The X/Open DTP model defines the set of standard interfaces and behaviors for implementing distributed transactions. On the Windows platform, it is Microsoft Transaction Server that provides COM components with common services, such as resource management, object pooling and the DTP-like transaction model.

The CORBA Object Transaction Service offers an alternative model for two-phase commit transactions. The OTS specification describes how the clients, transactional objects, resources and the OTS participate in transaction creation, context management, transaction propagation and two-phase commit protocol. In addition to the flat transaction model, the Transaction Service provides support for nested transactions. While a variety of OTS commercial implementations are available, their success will often be determined by features like transaction monitor, CORBA server management and productivity tools, and ease of integration with "legacy" X/Open DTP-based applications.

With the release of JDK 1.2, Java Transaction Service and Enterprise JavaBeans specifications, CORBA and Java platform integration has never been better. EJB provides necessary higher-level abstractions for creating business components in Java. At the same time, EJB relies on technologies like CORBA to provide the underlying service implementation. As a result, understanding of CORBA, CORBAservices and CORBA OTS in particular will prove to be essential for building large business systems in Java.

References and Resources

Arnold, K., and Gosling, J., "The Java Programming Language," Addison-Wesley, Reading, MA, 1996.
"Distributed TP: XA Specification, X /Open Document C193," X/Open Company Ltd., Reading, UK, 1992.
"Enterprise JavaBeans, Version 1.0," Sun Microsystems, Palo Alto, CA, March 1998.
"OrbixOTS Administrator's and Programmer's Guide," IONA Technologies PLC, Cambridge, MA, 1997.
"OrbixOTM Guide," IONA Technologies PLC, Cambridge, MA, 1997.
"Transaction Service Specification: Version 1.1," Object Management Group, November 1997.
"White paper - The Object Transaction Service," IONA Technologies PLC, Cambridge, MA, 1997.

About the Author

Maros Cunderlik is a lead consultant with 3M. He focuses on applications design and distributed object architectures. He can be reached at mcunderlik@mmm.com.

mcunderlik@mmm.com

Listing 1: CosTransactions Module (OMG IDL).

```
//The CosTransactions Module

#include <Corba.idl>
module CosTransactions
{
  //DATATYPES
  enum Status
  {
    StatusActive,
    StatusMarkedRollback,
    StatusPrepared,
    StatusCommitted,
    StatusRolledBack,
    StatusUnknown,
    StatusNoTransaction,
    StatusPreparing,
    StatusCommitting,
    StatusRollingBack
  };
  enum Vote
  {
    VoteCommit,
    VoteRollback,
    VoteReadOnly
  };

  //Structure definitions
  struct otid_t
  {
    long formatID; /*format identifier. 0 is OSI TP */
    long bqual_length;
    sequence <octet> tid;
  };
  struct TransIdentity
  {
    Coordinator coord;
    Terminator term;
    otid_t otid;
  };
  struct PropagationContext
  {
    unsigned long timeout;
    TransIdentity current;
```

▼▼▼▼▼▼ CODE LISTING ▼▼▼▼▼▼

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

Wall Street 1/4 Ad

SIGS S

SPREAD

gpharrison@compuserve.com

A String Bean: *Making JavaBeans with VAJ*

The holy grail of object-oriented development

by **Mike Fichtelman**

Visual Age for Java™ provides a good IDE for object oriented development. One of the strong points of Visual Age for Java is its support for JavaBeans. As everyone in the world knows by now JavaBeans are components, or software parts, from which applications can be built. This has been the holy grail of object-oriented development, and Visual Age for Java supports it well. It provides a convenient, connect-the-dots metaphor for assembling an application from parts.

One of the obvious questions one might ask, of course, is, "Where do the parts come from?" IBM does supply some JavaBeans with Visual Age for Java but, naturally, they won't do everything. If you need parts, and they're not available, Visual Age for Java makes the process easy. Once you've made the new parts they are reusable in any application. They can even be exported for use outside VAJ.

I'll illustrate this with the String Bean. Let's say you've developed a number of string handling functions that could be used in various situations or programs. You could simply include them when they're needed in a new application but JavaBeans provides a better alternative. You can reuse Beans without changing, or even knowing about, the code itself. You can then expose the methods or properties of the Bean and connect your application to them.

For example, you might need to parse an input string into its component words. If a method for this was included in a String Bean, you could design the input applet or application frame visually in Visual Age for Java using the supplied design beans, adding an instance of the String Bean to your frame and then connecting the JavaBeans. These are the steps you need to take to create a simple application that will allow input of a text string, and then display the individual words of the string in a list box at the click of a button.

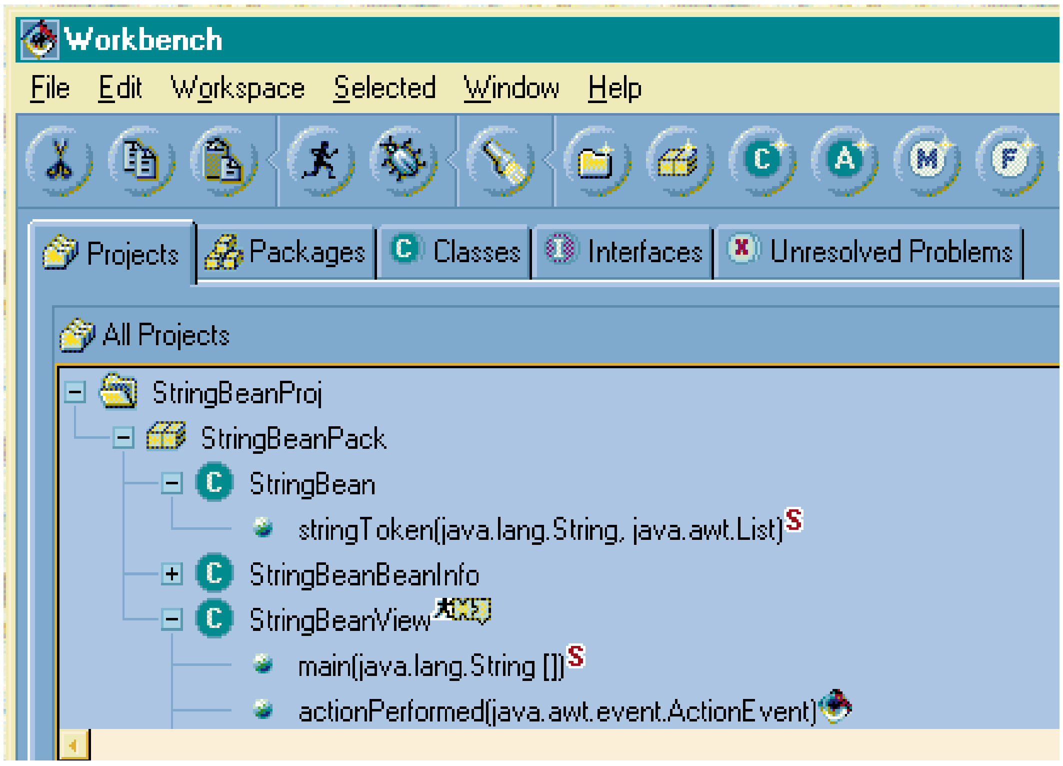
First, open Visual Age for Java and create a new applet. When prompted for project and package names, use StringBeanProj and StringBeanPack. The hierarchy should look something like that in Figure1.

Next, create a new class and name it StringBean, but choose not to design it visually. Be sure to import the java.awt.* and java.util.* packages when prompted. Then, create a new method for the StringBean class called stringToken with a signature of public static void stringToken (String Instring, List fillList). Add the code block in Listing 1 and save it.

Now, if you select the BeanInfo tab you see a list of all available features and properties for the new StringBean bean. It should look like Figure 2.

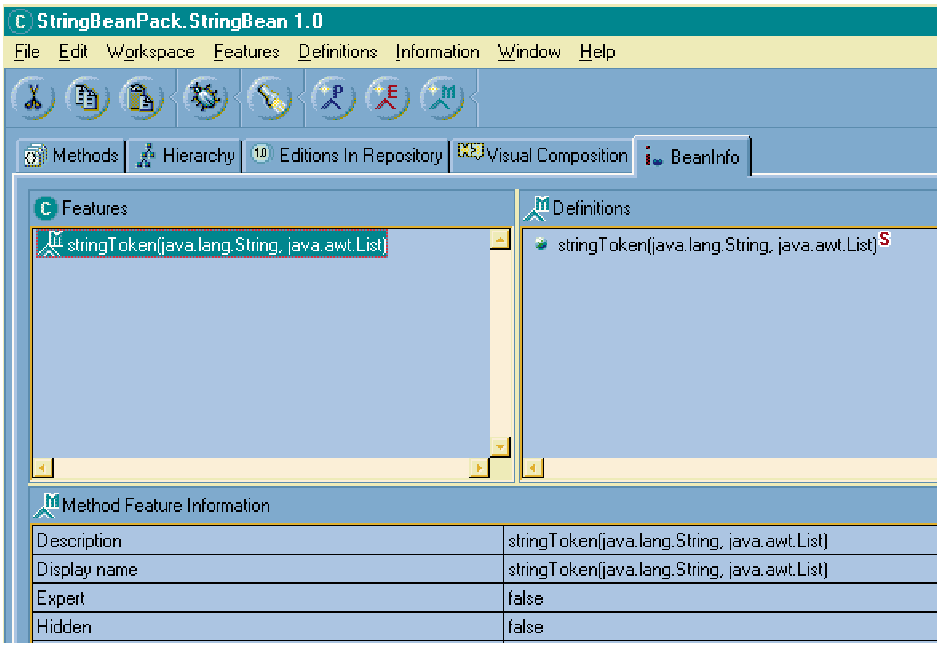
Next, select the StringBeanPack package in the hierarchy and click the C button to build a new class. Call this class StringBeanView and select the choice to design it visually, or go to the Visual Composition editor. After the blank applet panel is displayed, you'll need to add the appropriate beans from those supplied with Visual Age for Java to complete the application/applet. First, add a TextField bean. Then, add a List bean. Finally, add a Button bean and change the label property of the bean to String Bean.

You're ready to add the StringBean JavaBean we just created. From the main



```
StringBeanProj
  StringBeanPack
    StringBean
      stringToken(java.lang.String, java.awt.List)
    StringBeanBeanInfo
    StringBeanView
      main(java.lang.String [])
      actionPerformed(java.awt.event.ActionEvent)
```

Figure1


StringBeanPack.StringBean 1.0
File Edit Workspace Features Definitions Information Window Help
Methods Hierarchy Editions In Repository Visual Composition BeanInfo
Features
stringToken(java.lang.String, java.awt.List)
Definitions
stringToken(java.lang.String, java.awt.List)
Method Feature Information

Description	stringToken(java.lang.String, java.awt.List)
Display name	stringToken(java.lang.String, java.awt.List)
Expert	false
Hidden	false

Figure 2

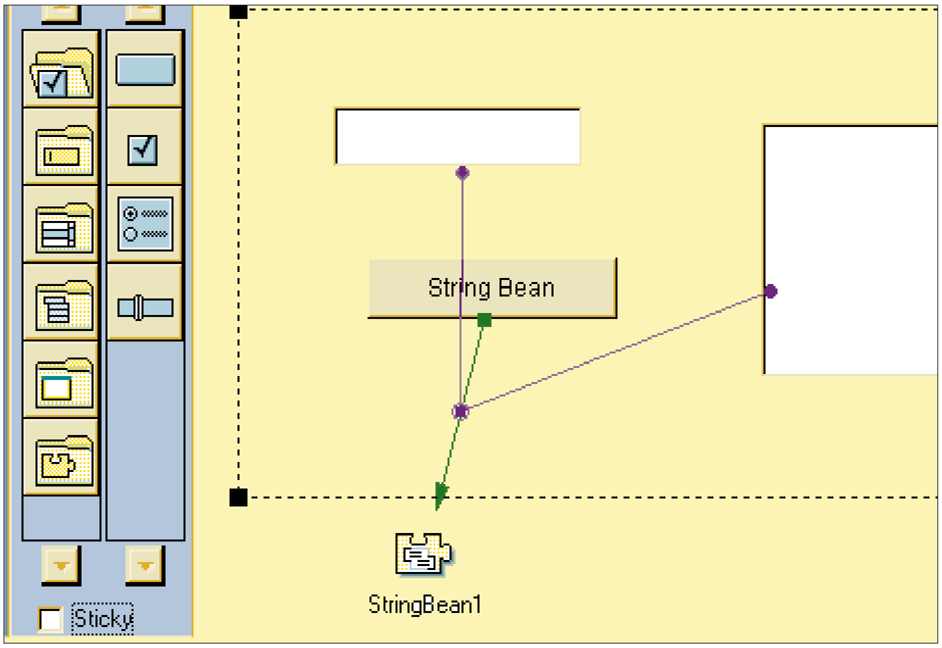


Figure 3

menu select Options, then Add Bean. In the dialog box displayed, you can either type StringBean or choose to Browse through the available beans and select it off the list. Click OK, then click again anywhere outside the frame boundary and the jigsaw part icon for StringBean1 to appear.

Now that all the beans are in place it's time to connect them together. For the first connection click the right mouse button on the String Bean Button, then select Connect and then actionPerformed(java.awt.event.ActionEvent) from the drop down menus. Move the mouse pointer to the StringBean1 icon and click mouse button 1, then select All Features. From the bean Features list, select the stringToken() method created earlier.

For the second connection to the TextField bean click mouse button 2 on the first connection line and select Connect. From the list displayed, select inString. Drag the mouse pointer to the TextField bean and click mouse button 1. From the list displayed, select text.

For the third connection to the List bean click mouse button 2 on the first connection line and select Connect. From the list displayed, select fillList. Drag the mouse pointer to the List bean and click mouse button 1. From the list displayed, select "this."

Now that all the connections between the beans are complete the design panel should look like Figure 3.

You can test the StringBean applet by selecting Tools->Test->Run from the main menu of Visual Age for Java. Type some text into the TextField(be sure to leave some blanks between words), click the String Bean button and a list of the words in the input string will be displayed in the list box.

I'm sure you can see how this humble foundation could be used to build more extensive applications. Other string handling and manipulation functions can be added to the StringBean bean at any time, making their addition to a new application a simple drag and drop operation. You could even add a StringBean feature that changes string text colors from black to green!

About the Author

Mike Fichtelman is a systems planning officer at ABN AMRO Information Technology Co., and is involved in various Web development projects. Mike can be reached at mike.fichtelman@abnaro.com.

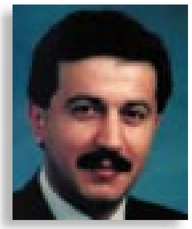
mike.fichtelman@abnaro.com

Listing 1.

```
/**
 * This method was created by a SmartGuide.
 * @return java.lang.String
 * @param fillList List
 */
public static void stringToken (String Instring, List fillList) {

  StringTokenizer tokenizer = new StringTokenizer(Instring," ");
   while (tokenizer.hasMoreTokens())
     {

      fillList.addItem(tokenizer.nextToken());
     }
 }
```

CODE LISTING

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



DashO-Pro
by preEmptive Solutions

Protect Your Work; Streamlining and Hiding Your Code

by **Ed Zebrowski**

Dash-O Pro
preEmptive Solutions.
26250 Euclid Ave, Suite 503
Cleveland, OH 44132
Phone: 216 732-5895
Fax: 216 732-5897
Web: http://www.preemptive.com
Email: dasho@preemptive.com
Requirements: Java 1.1 compatible runtime 64 Meg (min), 96 Meg (recommended)

I remember how I first got into Java. A friend called my attention to these neat little mini applications that could be easily embedded into HTML files. These applets provided a quick way to jazz up even the most mundane Web pages. She showed me some UseNet groups that contained huge libraries of these applets. I soon learned how to go into their code and change (dare I use the word "hack"?) them to fit my needs. Those applets were nobody's property; they were just there for the taking. Some people would come up with clever ways of making them run more efficiently, and would post their findings for all to use. When working with pubic archives, this is all well and good. The problem is many people learned how to hack Java applications that were not public domain. They would download applications, decompile the source and hack it to look like their own.

Today's complicated Java application involves many hours of hard work. Many of us do so under the employment of a company that stands to gain or lose hundreds of thousands of dollars based on the success of the application. We can't take the risk of unethical people using our code to suit their own needs. At the same time, code has to be quick and neat. What we need is a development tool that makes it literally impossible to hack source code without sacrificing speed and efficiency. What's needed is DashO-Pro from preEmptive Solutions.

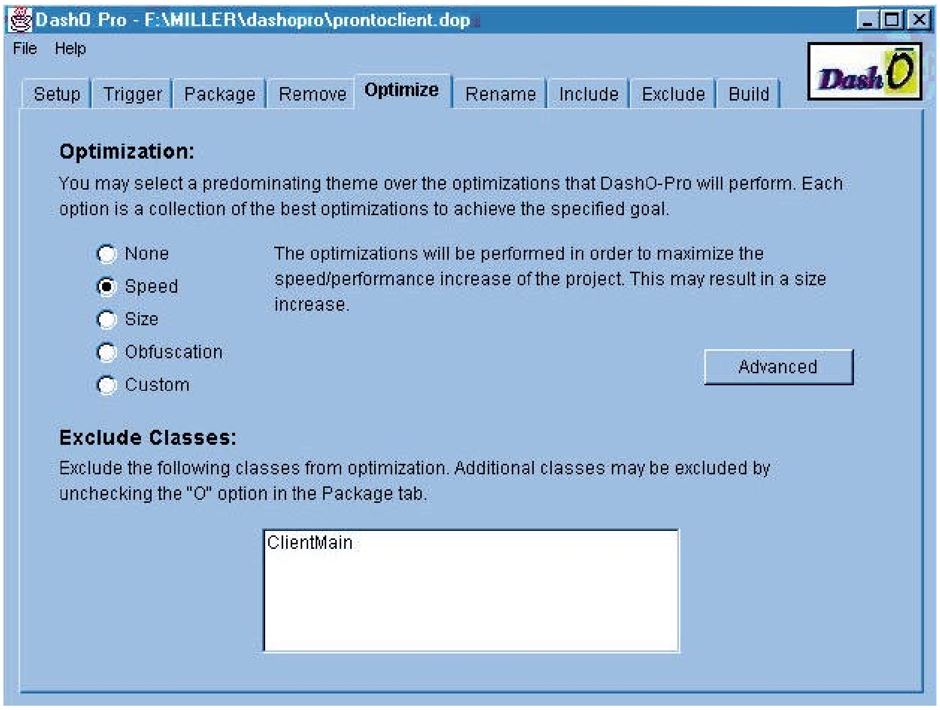
Obfuscation With Traditional and Advanced Methods

No program is totally safe from decompilers (to claim so would invite big trouble!), but DashO-Pro makes life as difficult as possible for them. Some of the methods used are:

- The removal of extraneous debugging information from class files.
- Removal of unused classes, fields and methods for maximum size reduction.
- Renaming all possible methods, classes and fields. All methods, such as public and private, can be renamed as long as they don't override a method from a non-included class. This process does not effect methods such as init and paint. Renaming reduces all to one or two-character names. Since decompilers have the ability to rename unprintable names back to printable ones, DashO-Pro provides sophisticated renaming properties that can't be bypassed by decompilers.
- The duplication of constant pool entries. This is a clever feature, as it won't rename multiply-used entries. Suppose the string "testing" is printed while there is a method called "testing;" the string would be printed as is but the method would be renamed.
- The use of irreducible control flow graphs for obfuscation. These can't be produced in the Java language due to its control structure. Code is changed in such a way that it no longer has an equivalent sequence in Java at the source level.

More Than Just an Obfuscator, It's an Advanced Optimizer, Size Reducer and Packaging System

Anyone who has ever done programming knows the value of correct, efficient code. Traditionally, an optimizer is used to increase code performance. Optimizing tools for Java have been up until now immature at best. DashO-Pro implements many



Optimization:

You may select a predominating theme over the optimizations that DashO-Pro will perform. Each option is a collection of the best optimizations to achieve the specified goal.

None
[✓ Speed]
Size
Obfuscation
Custom

The optimizations will be performed in order to maximize the speed/performance increase of the project. This may result in a size increase.

Exclude Classes:

Exclude the following classes from optimization. Additional classes may be excluded by unchecking the "O" option in the Package tab.

ClientMain

Figure 1: When using the GUI, Optimization can be as simple as clicking on radio buttons.

standard optimizing transforms unavailable in today's Java compilers, as well as some new transforms targeted specifically for Java bytecode.

One classic optimization is dead code elimination. DashO-Pro takes this to the nth degree by removing all unused information in your program. preEmptive Solutions has provided the following sample code to demonstrate DashO-Pro's removal techniques:

```
class MyClass {
   int Z;
   public static void main(String args[]) {
     System.out.println("Hello World");
   }

   public void doesNothing() {
     Z = 5;
     OtherClass X = new OtherClass();
     X.doOtherThings();
   }
 }
```

In this example, DashO-Pro's algorithms detect that the "doesNothing" method is never called: therefore, it is removed along with "OtherClass" and the "Z" variable. DashO-Pro's output only includes the absolute minimum set of classes, methods and fields required by your application. Your code size is minimized, often a desired feature for applets and other code that needs to move around the network.

1/2 Ad

Using DashO-Pro

DashO-Pro can be run as a command line or GUI application. When running as a command line program there are five runtime options that can be used with DashO-Pro:

1. –f :force execution. This option, as the name suggests, forces execution, even when your application uses dynamic class loading (e.g,. by using the Class forName method). To use this option, it is necessary to specify all dynamically loaded classes in the configuration file; alternatively, you may allow DashO-Pro to automatically detect possible dynamically loaded classes.
2. –v :verbose output. When this option is used information is given about the progress of the execution.
3. –i :investigate only. This option tells DashO-Pro not to create any disk files. A report will be generated which specifies the candidates for removal.
4. –q :run quietly. In this mode, DashO-Pro runs completely without printed output. Use this option for inclusion into application build sequences. The verbose option will be overridden here.
5. <configfile> : configuration file. This allows the naming of a specific configuration file, which is required for running DashO-Pro. This is a handy option when using multiple, tailor-made configurations in DashO-Pro. Trigger methods are not entered on the command line, as they must be included in configuration file.

When using the GUI, it is not necessary to write a configuration file, as the GUI is really a front end to the configuration file. The interface is initiated by running the DashO-ProGui class from the jar file. The Windows enthusiast will be delighted to know that double clicking the icon will run the interface. I found the GUI to be well laid out and a snap to move around in.

DashO-Pro's triple feature of optimization, obfuscation and compression makes it an extremely valuable tool in the professional Java developer's bag of tricks. If you have a need to streamline and hide your code, it's a must have!

About the Author

Edward Zebrowski is a technical writer based in Orlando, FL. 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

Other Java APIs and Products

The application programming interfaces that enhance the functionality

by **Ajit Sagar**

Let's take a minute to recap the discussion we've had so far in The Cosmic Cup. The Java platform is the software platform for the computing environment defined by Java. The Java platform APIs define the application programming interface for the Java platform, which consists of categories of APIs that address different segments of computing and related industries.

However, several APIs defined under the scope of Java APIs don't fall under any of the formal categories defined under the Java platform, namely Base Platform, Commerce, Security, Media, Enterprise and Server. In last month's column these APIs were mentioned as "Nonstandard Java APIs." This month we'll focus on these APIs.

Before we begin, I'd like to point out that this is not a category formally defined by JavaSoft; it is a category I have defined for the purposes of our discussion, to provide complete coverage of the scope of APIs defined under Java. Note also that for the moment we will forgo discussion of Java technologies that focus on hardware devices, embedded Java and Java operating system APIs (JavaOS). We will also postpone discussion on APIs that focus primarily on browser-based interfaces. These two kinds of APIs warrant independent discussions, and I will cover them in future articles for The Cosmic Cup.

Nonstandard APIs

The APIs defined under the "nonstandard APIs" category supplement the APIs formally defined under the Java platform APIs. They are based on some of the platform APIs and enhance the functionality of the APIs they extend. Figure 1 illustrates the APIs discussed in this article. These APIs are:

- JavaBeans Activation Framework (JAF)
- InfoBus
- JavaMail
- JavaSpaces
- JavaHelp

These APIs, and the roles of each API in Java and platform APIs related to each API, are provided in Table 1. The following section briefly examines the APIs listed in the table.

JavaBeans Activation Framework

This framework (formerly known as Glasgow) enhances the functionality of JavaBeans by adding data awareness to the data types used in JavaBeans-based component design. The standard Java platform does not provide a consistent strategy for determining the data types of a software component, binding typed data to a component or an architecture to support these features. JAF overcomes these shortcomings in the JavaBeans component model.

JAF uses MIME (Multipurpose Internet Mail Extensions) types to encapsulate and determine the data type. Using JAF, developers can write JavaBeans-based components that provide added functionality to Web browsers, office suites and other Beans-based environments. The JAF technology proposes data typing and a registry to hold definitions of these data types. It implements the following services:

- Determines the type of arbitrary data based on MIME types
- Encapsulates access to data based on the data's MIME type
- Discovers operations available on a particular data type
- Instantiates the software components that correspond to specific operations on a particular piece of data

The major pieces of the JAF are:

1. DataHandler class: This class provides a consistent interface between JAF-aware clients and other subsystems. It encapsulates a data object (which is an instance of the typed data) and provides methods that can act on this object.
2. DataSource interface: This interface

API	Description	Related Platform APIs	Role
JavaBeans Activation Framework (JAF)	Provides a set of standard services to introspect on a piece of data and to instantiate the appropriate JavaBeans component to perform operations on it	JavaBeans	Component Development, messaging
InfoBus	Provides a generic mechanism for exchanging data between JavaBeans components by defining a small number of interfaces between cooperating Beans and specifying the protocol for use of those interfaces	JavaBeans	Component development
JavaMail	Models a mail system that provides a framework for building Java-based mail and messaging applications	JAF	Networking, messaging
JavaSpaces	Provides a distributed persistence and data exchange for Java objects	Java RMI	Distributed component development, networking.
JavaHelp	Defines a universal Help system for the Java platform for designing platform-independent help content for graphical user interfaces	JFC	User interface design

Table 1: Nonstandard platform APIs

encapsulates an object that contains the data. The encapsulated object can return a string that defines the MIME type for the data, as well as a stream that provides data access. Classes extending the DataSource interface can be implemented for common data sources like the Web, file systems, IMAP, ftp, etc. The interface can also be extended to allow per-data-source user customizations. The DataSource is set in the DataHandler class. Once this is done, clients can determine the operations available on the data. JAF provides two convenience DataSource class implementations: FileDataSource, which allows access to data in a file, and URLDataSource, which allows access to data at a URL.

3. CommandMap interface: This interface allows consumers to determine the "commands" available on a particular MIME type. It also provides an interface to retrieve an object that can operate on an object of a particular MIME type. The CommandMap maintains a list of available operations on a particular data type. JAF defines a framework for the CommandMap that allows components to determine which commands are available on the system.
4. Command Object interface: This interface is extended by JavaBean components in order to interact with the JAF services. This enables the JavaBeans to directly access their DataSource and DataHandler objects to retrieve the data type and act on the data.

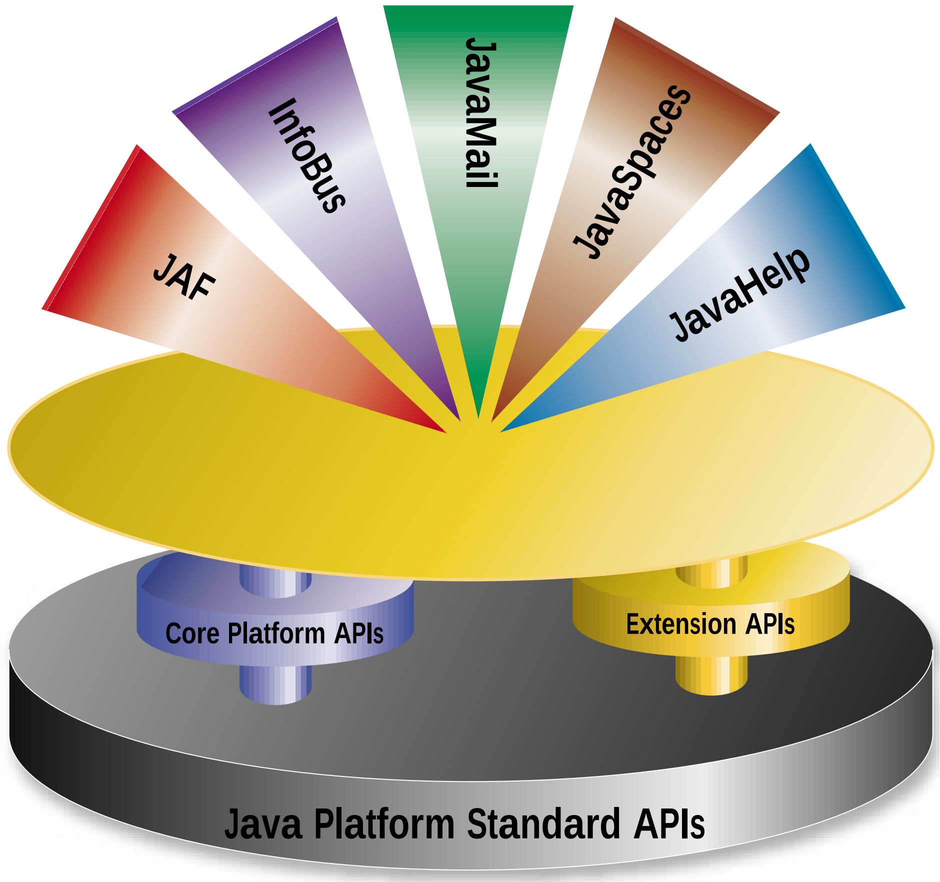


Figure 1

The JavaBeans Activation Framework is implemented as a standard extension. Sun provides a reference implementation of JAF for the draft 1.0 specification.

InfoBus

InfoBus enhances the functionality of JavaBeans by enabling dynamic exchange of data between JavaBeans components by defining a small number of interfaces between cooperating Beans and specifying the protocol for use of those interfaces. The JavaBeans model allows developers to construct applications by connecting components programmatically and/or visually. However, the specifications do not suggest methods by which these Beans should dynamically exchange data.

Both the InfoBus and JAF enhance the JavaBeans model, but in different ways. JAF supports data exchange using MIME types, and thus provides support for exchanging typed data. It also supports dynamic determination of the commands associated with the data. InfoBus defines sophisticated data models that have known interfaces to act on the data. It also defines the event mechanism and the protocol for data exchange within a single JVM.

The InfoBus protocols are based on a notion of an information bus. All components that implement these interfaces can plug into the bus. As a member of the bus, any component can exchange data with any other component in a structured way, including arrays, tables and database rowsets. The main components of the InfoBus are:

1. InfoBus class: This class is at the heart of this technology. An instance of the class is the meeting place where members can join. InfoBus instances can be "named" within the scope of a JVM. The class maintains a static list of existing InfoBus instances, bus members, producers and consumers, and enables communication among them for announcing and locating data items.
2. InfoBusDataProducer interface: This interface is used to indicate that an object provides data on the InfoBus. Producers announce the availability of new data items.
3. InfoBusDataConsumer interface: This interface is 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 to other consumers.
4. InfoBusMember interface: This interface must be implemented by classes that want to join an InfoBus. Typically, data producer and data consumer objects implement this interface.
5. DataItem interface: This is the base interface for data items. A very lightweight component, it is the basic data unit for data interchange.
6. InfoBusItemAvailableEvent interface: Event objects implementing this interface are broadcast on behalf of a producer to tell consumers about the availability of a new DataItem.
7. InfoBusItemRevokedEvent: Event objects implementing this interface are broadcast on behalf of a producer to let consumers know that a previously available DataItem is no longer available.
8. InfoBusItemRequestedEvent: Event objects implementing this interface are broadcast on behalf of a consumer to let producers know about the need for a particular DataItem.

The InfoBus architecture and API were developed by Lotus Development Corporation, Inc., and Sun Microsystems, Inc. The specification is a final specification as Version 1.1 and will be a standard extension in JDK1.2.

JavaMail

The JavaMail API provides a set of abstract classes that model a mail system. In other words, it provides a platform- and protocol-independent framework on which to build Java-based mail and messaging applications. It includes common convenience classes, which encapsulate common mail functions and protocols. The JavaMail API incorporates concepts from IMAP, MAPI, c-client and other messaging systems.

JavaMail clients use the JavaMail API and Service Providers to implement the JavaMail API. The JavaMail architecture consists of the following layers:

- The Abstract Layer, which declares interfaces and classes to support mail handling
- The Internet implementation layer, which implements parts of the Abstract Layer using Internet standards
- The JAF Layer, which uses JAF to encapsulate data using MIME types and to handle commands for that data

The main interfaces and classes in the JavaMail API are:

1. Part interface: This interface defines the attributes required to define and format data.
2. Message class: This is an abstract class that implements the Parts interface and defines a set of attributes and a content for a mail message. The attributes specify addressing information and define the content structure and type. The content is a DataHandler object (from the JAF API). Message subclasses can implement several standard messages. A Message may contain multiple parts, each with its own attributes and content.
3. Folder class: The Folder object stores Message objects. It can contain other Folders or Message objects.
4. Store class: This class defines a database that holds a folder hierarchy together with its messages. It also specifies the access protocol for accessing Folders and retrieving messages and methods to access a database, and fetch folders.
5. Transport class: This class models the transport agent that routes a message to its destination address. It provides methods for sending a Message object to a list of recipients.
6. Session class: This final concrete class defines global and per-user Mail-related properties that define the interface between a mail-enabled client and the network. It allows JavaMail components to set and get specific properties, a default authenticated session object that is shared between desktops. It also acts as a factory for Store and Transport objects.
7. MailEvent class: This class provides the base class for all JavaMail events. It is based on the JDK 1.1 event model.

The JavaMail API is implemented as a Java standard extension. Sun's reference implementation of the JavaMail software implementation is currently available as Version 1.0b.

JavaSpaces

JavaSpaces technology provides a tool for building distributes protocols. It uses RMI and object serialization features of Java to solve two related problems in distributed computing – distributed persistence and the design of distributed algorithms. JavaSpaces technology will be used in workflow systems, customer management systems, trading services, agent systems, supply chain management, publish and subscribe services, resource allocation management, etc.

"InfoBus enhances the functionality of JavaBeans by enabling dynamic exchange of data between JavaBean components."

JavaSpaces typically provides services for the middle tier of a typical three-tier client/server application. However, they can run in any tier of an n-tier distributed model. A JavaSpace offers a place for communication between distributed objects on clients and servers in a network. It may also be viewed as an event-driven distributed cache that also supports behavior transfer.

A JavaSpace holds Entries. An entry can be written into a JavaSpace, which copies the Entry into local storage. Entries can be looked up using Templates.

The main parts of a JavaSpace are:

1. Entry: This is a typed group of object references represented by a Java class.
2. JavaSpace interface: This interface allows operations on Entries held in the JavaSpace object. The operations supported are write, which writes an entry into the space; read, which returns an entry that matches the template or an indication that no match was found (this basically returns a copy of the entry); take, which returns the actual entry that matches the template or an indication that no match was found (it also removes the entry from the JavaSpace); and notify, which notifies a specified object when entries that match the given template are written into the JavaSpace.
3. Transaction: This groups multiple operations on an Entry. The operations are applied as a batch of operations.

The JavaSpace specification is in the final version 0.999.

JavaHelp

JavaHelp software is both an API and a platform-independent, extensible help system written entirely in Java. This system provides capabilities for navigating, searching and displaying help information, thus allowing end users to learn how to use an application, applet, component, operating system, device, or Web page. The JavaHelp software can also be used to distribute online documentation in a heterogeneous environment, such as a corporate intranet or Internet.

JavaHelp is currently available as an early access release, version 1.0. It is expected to be publicly available after the release of JDK 1.2.

Conclusion

In this article we examined the APIs that enhance the functionality provided by the standard Java platform APIs. We briefly examined the roles played by individual APIs that fall under this category and looked at the pieces that define these APIs. Links for detailed information on all these APIs may be obtained from Sun's Java Website at http://java.sun.com/products.

Cosmic Reflections

No API is an island. One interesting aspect of the APIs discussed here is that they extend the standard Java APIs to specific domains. In some respects they may be viewed as "helper" APIs. It remains to be seen what new helper APIs will be defined in future incarnations of Java and what aspects of existing APIs they will enhance.

About the Author

Ajit Sagar is a member of the technical staff at i2 Technologies in Dallas, TX. He holds a BS in electrical engineering from BITS Pilani, India, and an M.S. in computer science from Mississippi State University. Ajit focuses on networking, UI, and middleware architecture development. He is a Sun certified Java programmer with eight years of programming experience, including two in Java. You can e-mail him at Ajit_Sagar@i2.com.

Ajit_Sagar@i2.com

Enterprise Application Development

Using Java Blend

Writing Java applications that access relational databases has just become easier

by **Syed Q. Abbas**

Object-oriented software development using Sun Microsystems' Java language has become an industry standard for building scalable enterprise client/server applications. Additionally, most online transaction processing (OLTP) applications developed today by enterprises use relational databases as a standard DataStore, forcing developers to be proficient in the details and intricacies of both relational database design and object-oriented programming. They also must contend with mapping objects to relational schema, object management and caching, object navigation, transaction and concurrency controls. However, until recently developing Java applications with relational databases has been a time consuming process for developers.

To simplify development of application programs that use the Java programming language objects and relational databases, Sun Microsystems is introducing a new software product called Java Blend. The primary goal of Java Blend software is to provide programmers with a tool to easily develop business applications entirely in Java. With Java Blend, developers aren't required to learn the details of relational databases or Structured Query Language (SQL).

Java Blend consists of a development tool and a run-time environment. The development tool provides bidirectional mapping that can be used to map existing relational schema into a set of Java classes or map an application model written in Java to a new database schema to store instances of Java classes. The run-time environment provides transparent and automatic conversion of database records to Java programming language objects and Java programming language objects to database records, object navigation, transactions, concurrency and object caching for high performance.

Java Blend understands a database schema and can automatically generate the default mappings based on the relationship information between the tables, whether it is one-one, one-many, many-many, foreign key or inheritance. Additionally, it features the advanced capability to modify mapping of views on multiple tables, mapping a Java class to multiple tables or multiple Java classes to one table. Features such as using different names and types in Java and the database for the same fields are also supported, of course.

Applications written in Java Blend are highly portable, not only because the code generated is 100% pure Java, but also because it is based on the Object Database Management Group (ODMG) standard for object/relational mappings and object databases. Java Blend uses JDBC to communicate to relational databases so that any database with JDBC support automatically becomes usable by Java Blend; it supports several major databases in its first release, and will support any JDBC or ODBC based database in later releases.

Java Blend supports optimistic concurrency as well as the traditional pessimistic locking. Use of the optimistic concurrency control mechanism allows Java Blend to detect the values that have changed in the program's memory and database since the last transaction began. This results in higher performance and scalability at run time.

Application development using Java Blend can be broken down into four main steps.

1. Object/Relational Mapping

Using the Java Blend tool, map your database schema to Java classes or vice versa. For example, if you had two tables – Customer and Orders – in your database that were defined by the data definition statements in Listing 1, Java Blend would generate the class definitions for Customer and Orders in Listing 2.

Java Blend automatically generates member variables for each column and accessors/mutators for each member variable. Java Blend also understands relations between tables and maps them as one-one, one-many or many-many relationships between Java classes depending on the relationship between the tables. It then generates appropriate accessors/mutators and other helper methods. For example, Java Blend figured out the one-many relationship between Customer and Orders and generated an accessor method getOrdersForCustomer that returns a collection of Orders objects and other helper methods, like addOrdersForCustomer and removeOrdersForCustomer for adding and removing Orders objects for a Customer.

Before products like Java Blend, these mappings had to be done manually by someone who understood both the data model and the object model. As this example shows, such a mapping can now be created automatically, including more complex one-many and many-many relationships.

2. Add Business Logic

Almost every business application will have some business logic. With Java Blend, the business logic can be added into the generated Java classes in the form of new methods or embedded in the accessors and mutators.

For example, the code snippet in Listing 3 shows how the method addOrdersForCustomer in class Customer can be modified to check the customer's credit history before placing the order.

3. Write Queries

Queries are often required in applications to extract data. Java Blend supports industry standard Object Query language (OQL) that has been defined by ODMG. OQL has a rich syntax that allows the writing of simple to complex queries to extract data in the form of Java objects from a relational database. For example, the following query will extract all the Orders objects with a unit price greater than $100:

```
Select O from Orders O
Where O.unitprice > 100;
```

This query will be compiled using the Java Blend tool and then later instantiated in your program to get a collection of "Orders."

With OQL, queries are written using the object model, not the relational model. This makes it much more intuitive and easy for object-oriented programmers to use.

4. Write the Application

So far we have seen how the mapping

works, where to add business logic and how to write an OQL query. Still missing, however, are ways to open and close the database, start and commit transactions and, a method of concurrency control.

Java Blend provides a set of Application Programming Interfaces (API) for database and OQL queries and transactions. The code snippet in Listing 4 demonstrates how some of these APIs can be used in a typical database application.

Notice that the open method takes a database URL, a user name and a password as arguments.

Starting the Transaction is as simple as creating an instance of the Transaction object and invoking the begin method. The begin method takes an argument that specifies the concurrency mode. In this case the transaction mode is optimistic. The value Transaction.PESSIMISTIC will start the transaction in the traditional locking mode.

The new and set methods on objects between the begin and the commit statements will result in changes in the relational database. For example, in this case a Customer object with customerid 1000 will be created as a row in the Customer table in the relational database.

With Java Blend software, even transactions are object-oriented, not relational oriented. This makes application development much easier for developers who are familiar with the object model but not as familiar with relational databases.

Java Blend software allows programmers to easily develop business applications entirely in Java without having to learn the details of relational databases or SQL. With Java Blend, writing Java applications that access relational databases has just become much easier.

If you would like more information on the Java platform or Java Blend software, visit http://java.sun.com, or call 1-888-THE-JAVA.

CODE LISTING

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

About the Author

Syed Q. Abbas is a senior Java developer at Sun Microsystems in Menlo Park, CA. Syed received his master's degree in computer science from AMU Aligarh, India, and has worked in the software industry for eight years. Syed can be reached at syed.abbas@Eng.sun.com.

syed.abbas@Eng.sun.com

Listing 1.

```
CREATE TABLE Customer (
    CustomerId INTEGER NOT NULL PRIMARY KEY,
     Name VARCHAR(32),
    Address VARCHAR(100)
)

CREATE TABLE Orders (
     OrderId INTEGER NOT NULL PRIMARY KEY,
     CustomerId INTEGER NULL REFERENCES Customer,
     Item VARCHAR(20),
     UnitPrice NUMERIC,
     Quantity  INTEGER
     )
```

Listing 2.

```
// Customer class
public class Customer implements PersistenceCapable {
        private int customerid;
        private String name;
        private String address;
        private DCollection ordersForCustomer;

        // Accessor methods
        public int getCustomerid() {
                return customerid;
        }
        public void setCustomerid(int customerid) {
                this.customerid = customerid;
        }

        public String getName() {
                return name;
        }
        public void setName(String name) {
                this.name = name;
        }

        public String getAddress() {
                return address;
        }
        public void setAddress(String address) {
                this.address = address;
        }
        public Iterator getOrdersForCustomer() {
                return ordersForCustomer.iterator();
        }
        public void addOrdersForCustomer(OrdersordersForCustomer)
{
                this.ordersForCustomer.add(ordersForCustomer);
        }
        public void removeOrdersForCustomer(OrdersordersForCustomer) {
        this.ordersForCustomer.remove(ordersForCustomer);
        }

        // Constructors
        public Customer (Database db, int p_customerid) {
                this.customerid = p_customerid;
        }
}

// Orders class
public class Orders implements PersistenceCapable {
        private int orderid;
        private String item;
        private long unitprice;
        private int quantity;
        private Customer customerForOrders;

        // Accessor methods
        public int getOrderid() {
                return orderid;
        }
        public void setOrderid(int orderid) {
                this.orderid = orderid;
        }

        public String getItem() {
                return item;
        }
        public void setItem(String item) {
                this.item = item;
        }

        public long getUnitprice() {
                return unitprice;
        }
        public void setUnitprice(long unitprice) {
                this.unitprice = unitprice;
        }

        public int getQuantity() {
                return quantity;
        }
        public void setQuantity(int quantity) {
                this.quantity = quantity;
        }

        public Customer getCustomerForOrders() {
                return customerForOrders;
        }
        public void setCustomerForOrders(Customercustomer-
ForOrders) {
                this.customerForOrders = customerForOrders;
        }

        // Constructors
        public Orders (Database db, int p_orderid) {
                this.orderid = p_orderid;
        }
}
```

Listing 3.

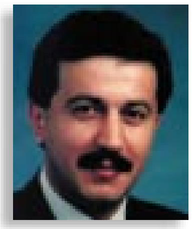
```
public void addOrdersForCustomer(Orders ordersForCustomer) {
// check the credit history before adding the order
if ( CreditHistory.getCreditHistory(this) == CreditHistory.BAD){
                System.out.println("Sorry, bad credit history");
                return;
        }
        // customer has a good credit history
        this.ordersForCustomer.add(ordersForCustomer);
}
```

Listing 4.

```
// open the database
Database db = Database.open("jdbc:microsoft://thestork:1433/order-
entry",
"scott", "tiger");

// start a transaction to create a new Customer

Transaction t = new Transaction();
t.begin(Transaction.OPTIMISTIC);
Customer c = new Customer(db, 1000);
c.setName("Foor Bar");
c.setAddress("901 San Antonio Road, Palo
Alto CA-94303");
// do some more stuff like create orders for this customer etc..
try {
        t.commit();
} catch (RetryException re) {
        System.out.println( "The transaction did not go through.
Try
again.");
}

// close the database
db.close();
```

1/2 Ad

Ad



SuperCede

by SuperCede, Inc.

Looking for a life raft in the vast sea of IDEs? Swim towards SuperCede!

by **Ed Zebrowski**

JAVA DEVELOPER'S JOURNAL JDJ ★★★★★★★★ WORLD CLASS AWARD

A C++ programmer buddy of mine got some extra work writing server-side applications in Java. He was a little pressed for time so he asked me about an IDE. "There are hundreds of those things floating around out there," he lamented. "How do I know which one is best?"

He had a point. Almost overnight there were more IDEs on the market than you can shake a 486 laptop at. Slogging through them all would be tiresome to say the least. I didn't hesitate when answering him, though, because I recently tried one that overwhelmingly impressed me. "Try SuperCede for Java Professional Edition," I said to him.

A Tutorial To Get You Started

Included with the Professional Edition of SuperCede is the MindQ tutorial. Whether you're just trying to get used to SuperCede as a new environment or you've never programmed before, MindQ is essential as a learning and reference tool. Among its many features it boasts a quick-access indexed format, hundreds of definitions of terms and links to Web sites that will further your knowledge of Java. I had never used SuperCede before, but with the help of MindQ I was on my way in minutes!

The Age-Old Problem of Simplicity Versus Effectiveness

In the past, IDEs came in two types: there were products that had many useful features, but were so confusing in their layout that they were burdensome to use; some were quite simple, but lacked many of the tools and features needed by the serious developer. SuperCede seems to have solved this dilemma. Although the interface is quite simple, I found this to be a powerful and complete building tool.

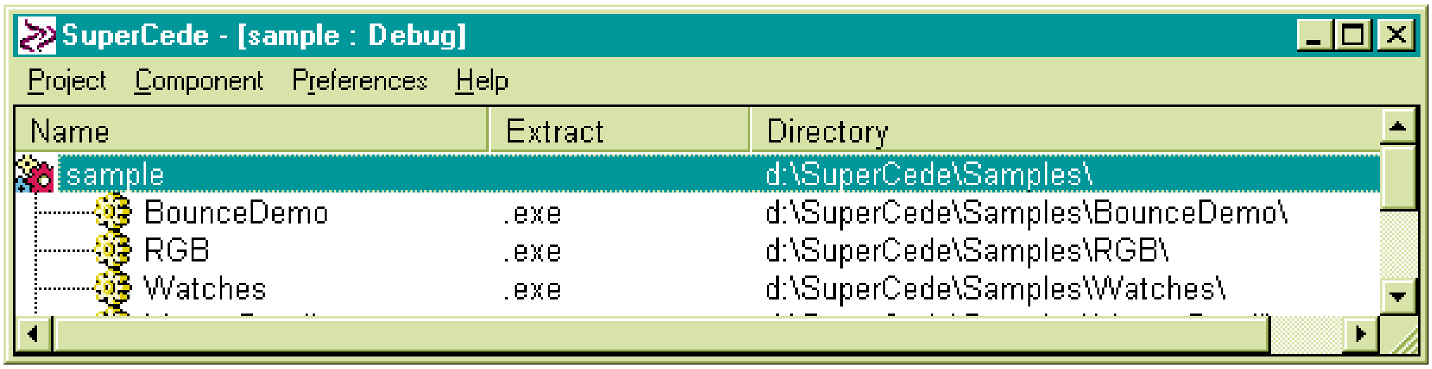
Simplicity...

SuperCede opens in a "Project" window which displays the contents of the current project. Another project can be selected by clicking on "Open Project..." in the project menu, or a new project can be initiated by selecting "New Project..." (see Figure 1).

One of the major problems I've had with other IDEs is the number of windows needed to keep track of all the building operations. For complicated applications the number of windows in some development environments is staggering. This is where SuperCede offers a big advantage. When a new project is started the "Component" window will open. Almost all aspects of building your application can be performed from this single window. This includes, but is not limited to, editing source files, creating forms, viewing files and viewing class hierarchies (see Figure 2).

This window is divided into two panes. On the left is the Browser pane, which allows selection of browsers for forms, beans, source files, data sources and even imported DLL files. When a file is opened from this browser, a viewer or editor opens in the Editor pane on the right. If a source file is selected, for instance, a source code editor opens on the right. Selection of a Bean will result in the opening of the Beans browser. This allows bean properties to be inspected. Beans may be dragged directly from the browser onto a form or from a preconfigured palette.

SuperCede
SuperCede, Inc.
110 110th Ave NE
Bellevue WA 98004
Phone: 800 365-8553
Fax: 425 637-5886
Web: http://www.supercede.com
Email: sales@supercede.com
Requirements: P90 or greater, 48 MB RAM (64 MB recommended), 200 MB disk space, Windows 95 or Windows NT 4.0
Price: $995 (Professional Edition Price)



Name	Extract	Directory
sample		d:\SuperCede\Samples\
BounceDemo	.exe	d:\SuperCede\Samples\BounceDemo\
RGB	.exe	d:\SuperCede\Samples\RGB\
Watches	.exe	d:\SuperCede\Samples\Watches\

Figure 1: SuperCede opens with a "Projects" window

At the top of the Component window is the Component toolbar. The toolbar exemplifies SuperCede's ability to perform tasks simply without sacrificing power. A simple click is all that's needed. Entire projects can be compiled by choosing "Build All." Selecting "Execute" runs a component. Any recompiling that is necessary is done automatically at this point. When "Go" is selected the component is automatically debugged and run. If a breakpoint is used the run will be stopped at that point. If a cursor is set at a certain point selection of "Run to Cursor" will cause the component to run to that point and stop, making a breakpoint unnecessary. "Extract" does just that -- extracts class, jar or executables.

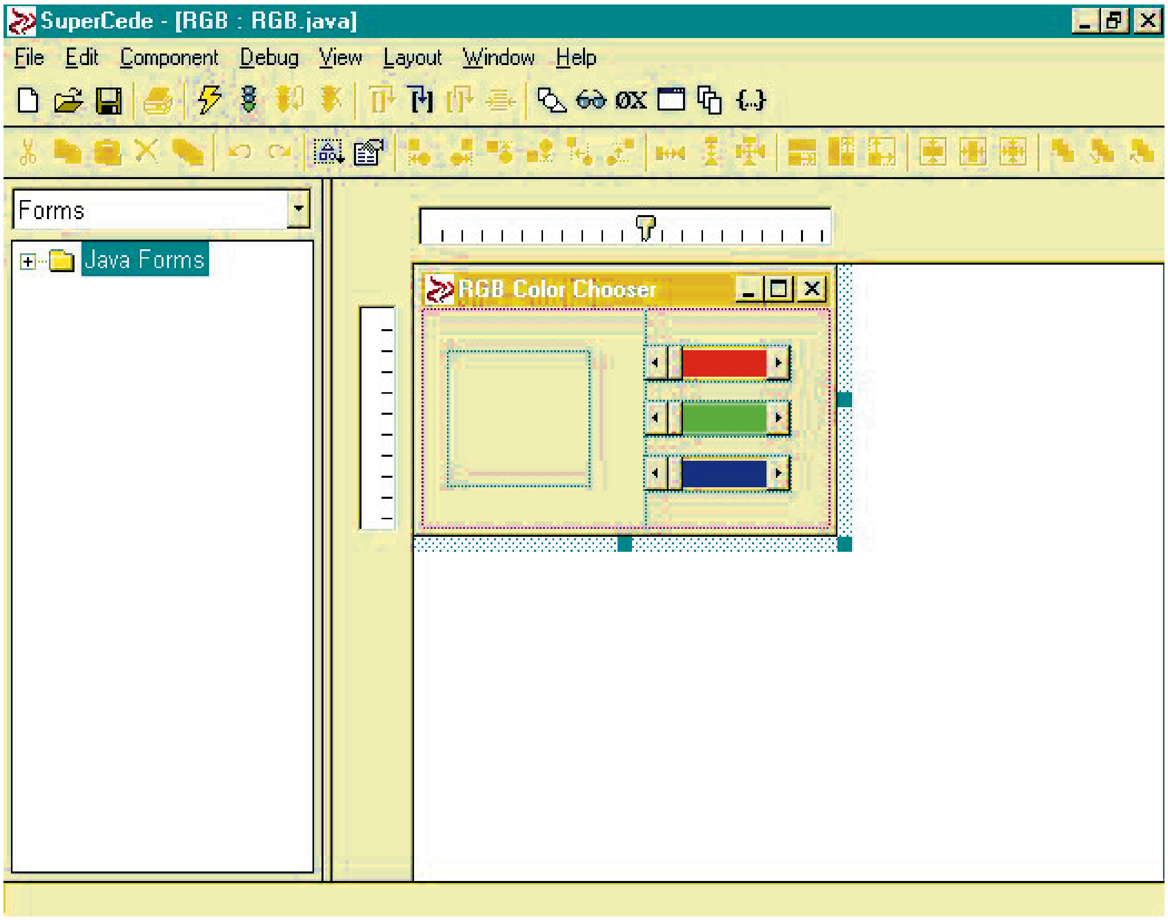

SuperCede - [RGB : RGB.java]
File Edit Component Debug View Layout Window Help
Forms
Java Forms
RGB Color Chooser

Figure 2: The Component window is one of the best organized I've seen

...and Power

Building the graphical interface for applications has never been this quick. When using the forms editor, buttons, bars, boxes and many other useful components, they are clicked and dragged from a toolbar on the right. By right-clicking on them it is possible to set their properties and functions. If more than one of the same component is needed, a click on the "Duplicate" button on the toolbar will create another of the selected component. The spacing and alignment of the components can be adjusted quickly by use of the "Layout" menu.

SuperCede also has the ability to create data-aware components. It is possible to communicate with a database without writing SQL statements, although they may be created if desired. The data source is imported by selecting the "Data Sources" from the Browser box, right-clicking the Browser pane and choosing "Import" from the popup menu. This opens the "Data Sources Properties" dialog box. The appropriate URL, driver, user name and password are then filled in.

I once helped on an application that needed some last-minute, fine adjustments. It seemed that every time we went back and ran the application, it was still just a couple of pixels off here and a fraction of a hair off there. We couldn't tell what the exact results of our changes would be until we recompiled and ran the application. This ended up costing us a night's sleep. SuperCede has come up with an ingenious way of solving a problem like this. They call it their "Flash Compiler." It provides an interactive environment in which programs can be changed as they run. First, click in the Component window, select the appropriate source files and change the code. When "Update" is chosen from the Component window, the program is recompiled as it runs. The changes made will appear while the application is running!

SuperCede offers some other neat debugging features as well. A breakpoint can be set by toggling through the code to the desired breakpoint, right-clicking and selecting properties. If "Tracepoint" is selected, rather than "Breakpoint," the application will not stop but will display a message window when the program reaches this line.

Using the "Debug Scratch Area" enables you to examine anything accessible from the current program scope. Expressions or statements can be entered and their results can be viewed instantly. Clicking the "Debug Scratch Area" inspector button at the desired breakpoint does this. This can be used to evaluate or execute either expressions or full blocks of code.

SuperCede also offers full interoperability in these other languages:

- C++: Existing C++ objects can be used as if they were developed in Java. The library can either be imported and built into the Java application, or calls can be made directly from Java code.
- ActiveX: Supercede includes ActiveX controls but allows installation of your own.
- Visual Basic: Import existing Visual Basic form files into the SuperCede IDE. The form will automatically be converted to an equivalent Java source file.

SuperCede is perhaps the most powerful and complete Java IDE I've yet seen. Its main highlight is that it maintains this power without giving up simplicity of use. My C++ buddy was glad I shared this information with him. Now if I could just get him to return the favor and bring back the lawn tools he's borrowed.

About the Author

Edward Zebrowski is a technical writer based in Orlando, FL. 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

JDK Naming Services: *COSnaming and JNDI*

What they are and how to use them

by **David Cittadini**

If you have created distributed enterprise applications, you would have had to deal with naming services and/or directory services in one form or another. Your applications would have needed to use these services to ensure that they gained access to distributed enterprise resources in an integrated and consistent manner. If you were creating smaller or more localized systems, however, you probably would have implemented solutions that did not use or need to know about naming or directory services.

Recently, a number of factors have conspired to ensure that the days of not considering or using a naming service and/or a directory service are probably numbered. The computing world has become more distributed and the resources more vast; finding these resources using naming and directory services has become pure necessity. Systems that were reasonably isolated in the past are now becoming part of large, integrated intranets, and in turn are merging or using the Internet. The upshot? A need for integrated naming and directory services.

The cost and availability of products implementing these services have made them available to a wide audience, not just the domain of large and expensive systems. With this in mind, it is no surprise to learn that future versions of Java will include facilities to make it easier for everyone to create applications that use naming and directory services. Specifically, JDK 1.2 will include: (1) a transient implementation of the CORBA Naming Service; (2) classes and interfaces to communicate with a CORBA Naming Service; and (3) the Java Naming & Directory Interface (JNDI) as a standard extension to enable access within Java to any naming or directory service.

This will bring the world of naming and directory services to everyone in the Java world, and probably means that a naming service should be standard in every Java application. This article focuses on the naming services and explains:

- What a naming service and directory service are
- The CORBA Naming Service
- How to use the CORBA naming service facilities included in JDK 1.2
- JNDI 1.1

What are naming and directory services? A *naming service* lets you find resources in a distributed system using human-readable names. This contrasts with a *directory service* that provides a database of information about the resources in the distributed system and how to interact with them. By its very nature, a directory service incorporates a naming service. Sometimes these services are so closely intertwined that it is difficult to distinguish between the two. From a Java application development perspective, our interest in this article is the naming service, the interface we deal with, not the underlying directory service.

The CORBA Naming Service

This service (also known as the Common Object Services Naming Service – CosNaming for short) became an OMG standard in September 1993. It provides a principal mechanism through which an ORB-based system can locate objects it intends to use. The purpose of the service was not to reinvent new naming and directory services, but to provide an object-oriented interface to existing nonobject-based name and directory services, such as the DCE CDS, ISO X.500, Sun NIS+ and the Internet's LDAP. Key design points for the specification included the following:

- The design imparts no semantics or interpretation of the names themselves.
- Clients need not be aware of the physical location of the name servers.
- Existing names and directory services employed in different network computing environments can be encapsulated transparently using name contexts.

The explosion of support for the Java platform over the past three years has not been without some problems; this month David Cittadini explores the two bundled JDK 1.2 services for mapping string names ("human readable" names) into references to which (in CORBA parlance) requests can be sent. JNDI and COSnaming provide similar services; where do they come from and how do they compare?

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

Because the CORBA Naming Service is a specification, it can have many different implementations and clients may differ in how they use it. Although it is possible to find an object using the standard "mini-naming service" provided by the standard ORB object invocation services, the CORBA service provides significant advantages:

- An object's interface name is defined at compile time. To change an interface name requires that you recompile your applications. A naming service, on the other hand, allows object implementations to bind logical names to its' objects at runtime.
- An object may implement only one interface name, but the naming service allows you to bind more than one logical name to a single object. (When I mention a naming service in this article, I will be referring to the CORBA Naming Service.)

Overview

The naming service maps human-readable names to object references and stores those names in a namespace. This name-to-object reference is called a *name binding*. To resolve a name means to find the object associated with the name. To bind a name is to create a name-to-object association. You have the option to associate one or more names with an object reference. Every object has a unique reference. The naming service organizes the namespace in a tree-like structure of naming contexts. A naming context contains a list of one or more names that have been bound to an object or to another naming context object. Name context hierarchies enable clients to navigate through the different naming context trees to find the objects they want. Naming contexts from different domains can be used together to create federated name services for objects.

You can reference an object using a sequence of names that forms a hierarchical naming tree. This sequence of names is known as a compound name. You always define a name relative to its naming context. The last component in the compound name is known as the object's simple name. You can start from any context and use a sequence of names to resolve a name. An example structure of a CORBA Naming Service is shown in Figure 1; a compound name would be /Offices/Australia/Melbourne/David, where Offices, Australia and Melbourne are the context names and David is the simple name

The Java Interfaces

All CORBA interfaces are defined using the language-neutral OMG Interface Definition Language (IDL). The OMG then provides language-specific mappings from the OMG IDL to Java (see the OMG IDL-to-Java Language Mapping document orbos/98-01-06 Final), enabling implementation of the CORBA interfaces in Java. The CosNaming module (mapped to the org.omg.CosNaming package in Java) provides all the interfaces required by the CORBA Naming Service with the resulting key Java interfaces as follows:

- **NamingContext:** This, the central interface in CosNaming, provides operations to bind names to object references and creates subcontexts to resolve names to object references.
- **NameComponent:** This represents the names, defined as sequences of NameComponent objects. All that is really required is to set the id for the NameComponent.
- **BindingIterator:** The NamingContext list method returns a BindingIterator object that enables you to iterate on a set of names and thus traverse the naming hierarchy.

Specific Java ORB vendors implement these interfaces to provide the specific services and functionality of their products.

How To Use the Naming Service

NamingContext and BindingIterator are the two key interfaces that implement the naming service. NamingContext objects contain a set of name-to-object bindings. The new_context method returns a naming context. The bind_new_context method creates a new context and binds it to the name you supply. You invoke the bind method on the NamingContext object to associate an object's name with a naming context. You can use rebind to bind a name that already exists; this will create a naming hierarchy. You unbind to remove a name from a binding context. Finally, the destroy method deletes a naming context.

You can find a named object by using the resolve method. This will retrieve an object bound to a name in a given context. You can then use the list method, which returns a BindingIterator object to iterate through a returned set of names.

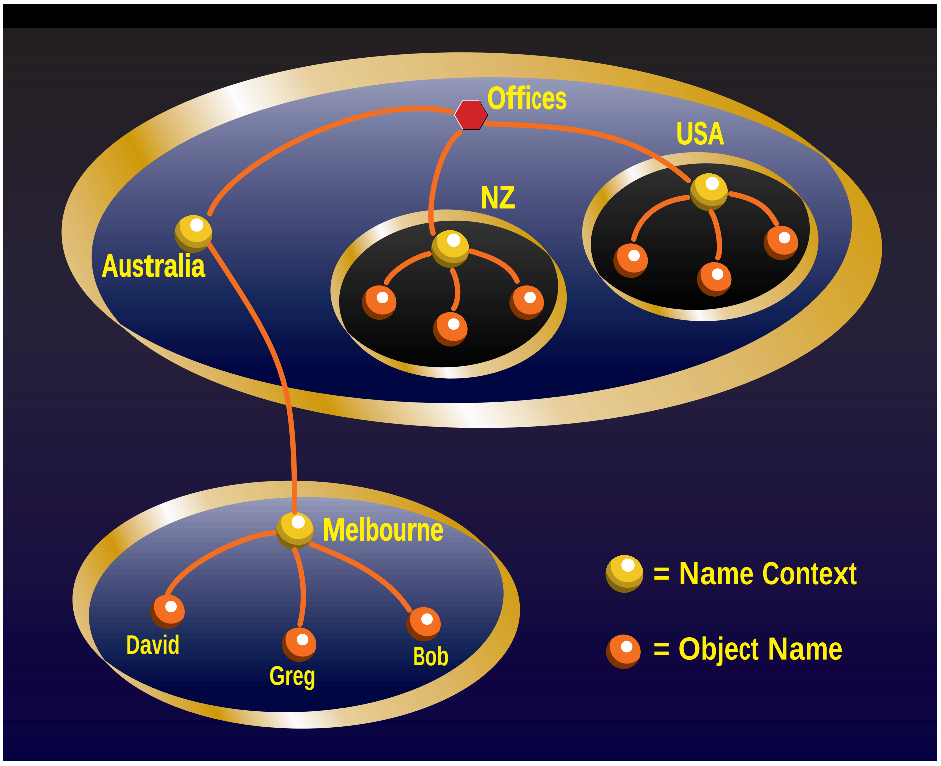


Figure 1

Using the CORBA Naming Service in JDK 1.2

JDK 1.2 comes with the following:

- **Java IDL Object Request Broker**: This is compliant with the CORBA/IIOP 2.0 Specification (OMG document orbos/97-02-25) and supports transient CORBA objects – objects with lifetimes limited by their server process' lifetime.
- **Java Mapping of the CORBA Specifications contained in the following packages:**
  – org.omg.CORBA contains the core CORBA interfaces and classes.
  – org.omg.CORBA.ContainedPackage contains a class describing a CORBA object in a CORBA container.
  – org.omg.CORBA.ContainerPackage contains a class describing a CORBA container object.
  – org.omg.CORBA.InterfaceDefPackage contains a class that is the description of a CORBA interface definition.
  – org.omg.CORBA.ORBPackage contains a class for the CORBA InvalidName exception.
  – org.omg.CORBA.portable contains classes and interfaces for developing ORB implementations.
  – org.omg.CORBA.TypeCodePackage contains classes for CORBA exception that can be thrown by TypeCode operations.
  – org.omg.CosNaming contains classes and interfaces for communicating with the CORBA Naming Service.
  – org.omg.CosNaming.NamingContextPackage contains helper and holder classes for CORBA exceptions that can be thrown by the CORBA Naming Service.
- **Transient CORBA Naming Service, tnameserv, stored in the bin directory:** This service does not store any namespace information and all namespace data is lost once the naming service is closed. This provides a base implementation so that developers can test their applications, but a full persistent implementation will require the purchase of a

commercial naming service.

In addition to these components included in the JDK, you can also download the idltojava compiler free from the Java Developer Connection. idltojava is hard-coded to use a default preprocessor. On Windows machines it uses the MS Visual C++ preprocessor (although this can be changed). Therefore, to use idletojava you have to have a compatible preprocessor.

Using the JDK 1.2 Naming Service

The following example shows how to develop a simple application using the JDK 1.2 naming service. A more complex example could have been created, but this simple one is excellent for the purpose. We will create a distributed version of the "Hello World" application. For all intents and purposes, it will operate as a standard ORB-based application except that we will use a naming service to find the object instead of letting the ORB try and find it. The "Hello World" program has a single operation that returns a string to be printed.

The resulting application will operate as follows: the JDK 1.2 tnameserv will be started. A HelloServer will be started and this will:

- Create and initialize the ORB (the default ORB in JDK 1.2).
- Create a HelloServer and register it with the ORB.
- Get the root naming context from the transient naming service.
- Bind an object reference of the HelloServer to the name "Hello" in the naming context.
- Wait for invocations from the client.

A HelloClient will be started and this will:

- Create and initialize the ORB.
- Get the root naming context from the transient naming service.
- Use the naming context to resolve the object named "Hello" to an object reference. This will return an object reference to the remote HelloServer.
- Use the object reference and call the HelloServer's sayHello() method. The ORB will handle the communications and the operation will return a string from the remote HelloServer.
- Print the string returned from the HelloServer.

Creating the Application

Step 1: Create a simple Hello interface in hello.idl as follows:

```
module HelloApp
{
 interface Hello
 {
  string sayHello();
 };
};
```

Step 2: Use the idltojava compiler as follows to compile the hello.idl file into the required Java mapping:

```
idltojava Hello.idl
```

This will automatically produce the following files (do not modify them):

```
Hello.java
HelloHelper.java
HelloHelper.java
_HelloImplBase.java
_HelloStub.java
```

Step 3: Use the JDK 1.2 Java compiler and compile the foregoing classes:

```
javac *.java
```

"You can reference an object using a sequence of names that forms a hierarchical naming tree."

Step 4: Create the HelloServer.java source file as in Listing 1.

Key points of the code include the following: HelloServant extends the _HelloImplBase server-side skeleton automatically created by the idltojava compiler. Typically, a naming context is passed to a naming server when it is started. You will always invoke resolve_initial_references to obtain the initial naming context regardless of how the ORB finds it. The ORB method resolve_initial_references returns a reference of type Object so we must narrow it to the derived NamingContext type. The NamingContextHelper is a helper object automatically generated by the idltojava compiler. We create a naming context called "Hello" and rebind it relative to the initial naming context.

Step 5: Create the HelloClient.java source file as in Listing 2.

Key points of the code include the following: we use almost exactly the same process as HelloServant except that instead of binding a name to a naming context, we resolve a name. The resolve returns an object reference with which we can invoke methods: in this case, the sayHello() method.

Step 6: Compile HelloServer.java and HelloClient.java, ensuring that the HelloApp package is correctly referenced in your classpath.

```
javac HelloServer.java HelloClient.java
```

Running the Application

Step 1: Start the transient naming service.

```
tnameserv -ORBInitialPort 900
```

Step 2: Start the HelloServer.

```
java HelloServer -ORBInitialPort 900
```

Step 3: Start the HelloClient.

```
java HelloClient -ORBInitialPort 900
```

The outcome is that the HelloClient will print the string "Hello World" to the console.

Using Other CORBA Naming Services

You may decide that the transient naming service is not enough for your production system, that you want to use something like Iona's ORBIX or Borland's VisiBroker in production. This will require no changes to the previous example. All you need to do is start the required naming service and ensure that the clients and servers use the correct port. That's it! The following shows the same example being run using Borland's VisiBroker.

Step 1: Start the VisiBroker Smart Agent.

```
start osagent -c
```

Step 2: Start the naming service.

```
java -DORBservices=CosNaming
-DSVnameroot=HelloTest -DJDKrenameBug
com.visigenic.vbroker.services.CosNaming.Ext
Factory HelloTest namingLog
```

-DORBservices=CosNaming is a required parameter for initializing the ORB to use the naming service. -DSVnameroot=HelloTest sets the root context for the naming service. -DJDKrenameBug is a work around for a bug in the Windows JDK.

Step 3: Start the HelloServer.

```
java -DORBservices=CosNaming
-DSVnameroot=HelloTest HelloServer
```

Step 4: Start the HelloClient.

```
java -DORBservices=CosNaming
-DSVnameroot=HelloTest HelloClient
```

JNDI

JNDI, a new addition to the Java APIs, provides Java with a unified interface to multiple naming and directory services. It doesn't matter what underlying implementations you use because your interface at the application level will always be the same.

JNDI provides both an API for almost all software developers and a Service Provider Interface (SPI) for the underlying service providers. These interfaces are contained in three packages:

- javax.naming for the naming operations
- javax.naming.directory for the directory operations
- javax.naming.spi, which contains the SPI

JNDI has the following key benefits:

- It integrates multiple naming and directory interfaces into a single interface.
- It takes advantage of the Java environment by using native Java objects for all processing.

As a result, messy object type conversions are eliminated. The API is simple and clean. As with all Java APIs, a lot can be done with a little.

The Naming Interface

The core interface in javax.naming is Context. Like NamingContext in the CORBA specifications, it defines basic operations such as adding a name-to-object binding, looking up the object bound to a specified name, listing the bindings, removing a name-to-object binding and creating and destroying subcontexts, etc. Of all the operations, lookup() will probably be used most often. This will return an object of whatever class is required by the Java application. One deals with complete Java objects and not generic Java wrappers. This approach differs from the CORBA approach in which data is described and wrapped by generic objects. The JNDI approach makes it much easier and cleaner to deal with naming services in the Java world while still having the flexibility of the world promised by the CORBA Naming Service.

Other key classes and interfaces in javax.naming include:

- Binding, which represents a name-to-object binding found in a context
- CompositeName, which represents a composite name that is a sequence of component names spanning multiple namespaces
- CompoundName, which represents a name from a hierarchical namespace
- InitialContext, which is the starting context for performing naming operations
- Name, which represents a generic name
- Reference, which represents a reference to an object found outside the naming/directory system

The Directory Interface

Unlike the CORBA Naming Service, JNDI extends its world to include the directory service. The DirContext interface enables the directory capability by defining methods for examining and updating attributes associated with a directory object. Each directory object contains a set of zero or more objects of the class Attribute. An Attribute represents an attribute associated with a named object and is denoted by a string identifier; it can have zero or more values of any type.

"The JNDI approach is easier and cleaner in dealing with naming services"

Once again, DirContext.search() will probably be the most commonly used operation. This supports content-based searching and returns the matching directory objects along with the requested attributes. Other key classes and interfaces in javax.naming.directory:

- Attributes represent a collection of attributes.
- InitialDirContext is the starting context for performing directory operations.
- SearchControls encapsulates factors that determine the scope of search and what gets returned as a result of the search.
- SearchResult represents an item returned as a result of the DirContext.search() methods.

The Service Provider Interface

The SPI provides a way for service providers to develop and hook up their naming and directory implementations to the JNDI. JNDI allows specification of names that span multiple namespaces. Thus the SPI provider methods allow different provider implementations to cooperate so as to complete client JNDI operations.

Service Provider Reference Implementations

In addition to providing the SPI, JNDI 1.1 also provides reference implementations for the following naming and directory services:

- CORBA Naming Service (as used in the following example)
- LDAPv3
- NIS
- File system

Most major naming and directory service vendors have made a commitment to support JNDI and develop specialized SPI implementations. To use these reference implementations, JNDI 1.1 requires that certain environment variables or system properties be set, thus providing JNDI with information on the service provider configuration. This may not be how commercial providers implement their solutions, but it is a good way to ensure that the required settings are not hard-coded into the solution. All reference implementations require you to set the java.naming.factory.initial system property to the name of the factory class that produces the initial context implementation (you set system properties via the -D switch when you run java). Some service provider reference implementations also require the setting of specialist properties as in Listing 3.

In addition, with the CosNaming server provider reference implementation, you will also need to set java.naming.corba.orb (see the next example and Listing 4).

The Outcome

This example will do exactly the same as the JDK 1.2 example previously outlined, except that instead of using the CORBA APIs we will use the JNDI APIs. This will also use the reference implementation of the CosNaming SPI provided in the JNDI 1.1.

Creating the Application

Steps 1 to 3 are the same as for the previous JDK 1.2 example.

Step 4: Create the HelloServer2.java source file as in Listing 4.

Key differences with this version of HelloServer2 (compared to HelloServer) are that our process to get a naming context is considerably easier. Instead of calling resolve_initial_references and then narrowing the object using NamingContextHelper in the CORBA naming service example, with the JNDI example all we do is create a new InitialContext. In addition, in the CORBA naming service example we create a NameComponent and rebind that to the naming context, whereas in the JNDI example we can rebind the object directly, making the process much simpler.

Step 5: Create the HelloClient2.java source file as in Listing 5.

Key differences with this version of HelloClient2 (compared to HelloClient) are that our process to get a naming context is considerably easier. Instead of calling resolve_initial_references and then narrowing the object using NamingContextHelper as in the CORBA naming service example, with the JNDI example all we do is create a new InitialContext. Note that the InitialContext has been set using the environment variable set at java.naming.corba.orb (this is not required, but is a nice implementation approach). In the CORBA naming service example we create a NameComponent and resolve that to the naming context; in the JNDI example we can look up the object directly, making the process much simpler.

Step 6: Compile HelloServer2.java and HelloClient2.java ensuring that the HelloApp package is correctly referenced in your classpath.

```
javac HelloServer2.java HelloClient2.java
```

Running the Application

Step 1: Start the transient naming service.
tnameserv -ORBInitialPort 900

Step 2: Start the HelloServer2.

```
java -Djava.naming.factory.initial=com.sun.
  jndi.CosNaming.CNCtxFactory
HelloServer2 -ORBInitialPort 900
```

When running an application using JNDI we need to set the system property, telling JNDI which naming service to actually use. In this example we will use the CORBA naming service. Note that the default port for the naming service is 900. You can reset this, however, by resetting the org.omg.CORBA.ORBInitialPort system property (as undertaken with the -ORBInitialPort argument above).

Step 3: Start the HelloClient2.

```
java -Djava.naming.factory.initial=com.sun.
  jndi.CosNaming.CNCtxFactory
HelloClient2 -ORBInitialPort 900
```

Summary

Naming services provide a way to find resources using names. In a world of increasing complexity and interconnectivity, it has now become mandatory to use these services. Therefore, no matter how simple your Java application is, you should seriously consider designing and implementing it so that it uses a naming service to find the resources required by your application. The new version of the JDK makes the choice of using a CORBA Naming Service easy as one is provided free in the JDK itself. Although this will not provide all the functionality you need – after all, it's only a transient service – it will give you everything you need to develop an application that uses a naming service. That's half the battle. As can be seen from the examples, using a naming service is not really a complex process, but they add much power and flexibility to the implementation of your applications.

With the advent of JDK 1.2, Java now brings the use of CORBA Naming Services to the masses. However, the additional introduction of JNDI 1.1 extends this functionality so you can use any naming service and any directory service as well. For the first time, the power of both services is integrated into a simple implementation independent Java API. The JNDI allows you to go beyond just CORBA Naming Services to provide an integration point for all naming and directory services in an enterprise. By using JNDI, you can get access to any naming and directory service from anyone, anywhere, anytime, and thus increase the power of Java as a key tool in the enterprises of the future.

About the Author

David Cittadini is based in Melbourne, Australia, and is a director of software development and consulting companies in Australia and New Zealand. He specializes in developing and consulting in leading-edge technologies, enterprise architectures and distributed enterprise systems. He is a member of the OMG and is also a Sun-certified Java developer. He can be reached by e-mail at David.Cittadini@tb91.com.

David.Cittadini@tb91.com

Listing 1.

```
// The package containing our stubs.
import HelloApp.*;
// HelloServer will use the naming service.
import org.omg.CosNaming.*;
// Special exceptions
import org.omg.CosNaming.NamingContextPackage.*;
// All CORBA applications need these classes.
import org.omg.CORBA.*;

public class HelloServer
{
 public static void main(String args[])        {
  try{
   // Create and initialize the ORB
   ORB orb = ORB.init(args, null);

   // Create the servant and register it with the ORB
   HelloServant helloRef = new HelloServant();
   orb.connect(helloRef);

   // Get the root naming context
   org.omg.CORBA.Object objRef =
     orb.resolve_initial_references("NameService");
   NamingContext ncRef = NamingContextHelper.narrow(objRef);

   // Bind the object reference in naming
   NameComponent nc = new NameComponent("Hello", " ");
   NameComponent path[] = {nc};
   ncRef.rebind(path, helloRef);

   // Wait for invocations from clients
   java.lang.Object sync = new java.lang.Object();
   synchronized(sync){
    sync.wait();
   }

  }
  catch(Exception e) {
   System.err.println("ERROR: " + e);
   e.printStackTrace(System.out);
  }
 }
}

class HelloServant extends _HelloImplBase {
 public String sayHello() {
  return "\nHello world!!\n";
 }
```

CODE LISTING

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

JDJ NEWS

Inprise Announces Delphi 4™

(New York, NY) - Inprise Corp. has announced Borland Delphi 4, a major new version of its award-winning rapid application development tool for Windows. Designed to help corporations deliver large scale enterprise-class business applications faster, Delphi 4 simplifies the integration of client, middleware and database development. Delphi 4 now includes support for both CORBA and COM, the leading distributed computing standards, as well as the Microsoft Transaction Server (MTS) and the new Oracle8 database server. Delphi will be available in the following versions: Delphi 4 Client/Server Suite, Delphi 4 Professional and Delphi 4 Standard.

Delphi 4 includes enhancements that allow organizations to integrate data throughout their enterprise and make it available on any desktop at any time.

For more information, including pricing, visit Borland's Web site at www.borland.com.

ObjectShare Announces PARTS™ for Java

(Irvine, Ca) - ObjectShare has released the Standard and Lite editions of PARTS for Java. Combined with the previously released Professional edition, developers can now choose a PARTS package that provides the feature set best suited to their needs.

PARTS for Java Lite is the entry-level edition. It includes the visual programming tool's support for JavaBeans and JFC/Swing provided in the rest of the PARTS line, and provides the same open support for switching between versions of the JDK.

PARTS for Java Standard builds on the Lite product, with the addition of JavaBeans for creating database applications quickly using JDBC. It also includes Oracle Lite along with jKit/Grid and Netscape Navigator. The powerful database components include ObjectShare's Form bean which can automatically generate user interfaces for any domain object. These components work with any JDBC datasource.

PARTS for Java Standard is available on CD only and costs $795, with an introductory price of $500 until the end of July. PARTS for Java Lite is available electronically and costs $149. PARTS for Java Professional lists at $1495.

For more information, call ObjectShare tollfree at 800 759-7272 or visit their Web site at www.objectshare.com.

TowerJ™ 2.0 Released

(Austin, TX) - Tower Technology, a provider of optimizing native Java deployment compilers and run-time software, has released TowerJ 2.0, a high performance server-side Java execution environment.

TowerJ converts Java bytecode into optimized .exe's, which allow Java applications to execute with reliable C++-like performance across a range of server-class computers.

For more information, visit Tower's Web site at www.twr.com or www.towerj.com.

InterBase 5.0™ Released

(Scotts Valley, CA) - InterBase has announced the release of InterBase 5.0. This version combines ease of installation, use and maintenance, a versioning architecture and features such as event alerters and arrays.

New features include: SuperServer architecture on all platforms, InterClient (the all-Java JDBC driver), improved performance in the query optimizer, UDF (User Defined Function) library, group privileges, index garbage collection, SQL roles, international character sets, cascade declarative referential integrity, new security check for reference privileges, attachment governor, gbak improvements and multifile backup, new temporary file management and guardian process to monitor server up time.

For more information, visit InterBase's Web site at www.interbase.com or contact Amelia Arnett at 408 430-1506 or e-mail her at aarnett@interbase.com.

KL Group in Top 100

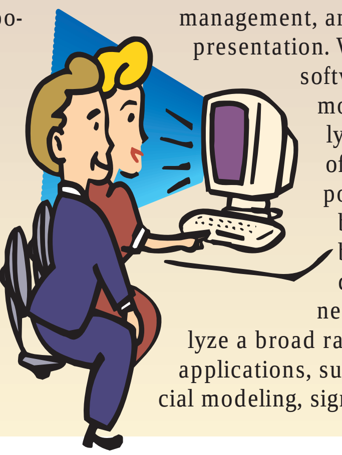
(Toronto, ONT) - KL Group has been recognized for the third year in a row as one of the 100 fastest growing companies in Canada, with revenues growing an impressive 692% from 1992 to 1997. The survey was based on data compiled from over a half million ballots sent out across the country.

KL Group recently launched its new JProbe™ Java profiling tool. JProbe and JClass, KL Group's family of certified 100% Pure Java JavaBean components are used by professional developers worldwide. JClass has also catalyzed a number of key technology partnerships with leading vendors, including Inprise and Silverstream.

For more information, visit www.klg.com.

Rogue Wave Software, Inc. Ships Analytics.h++™

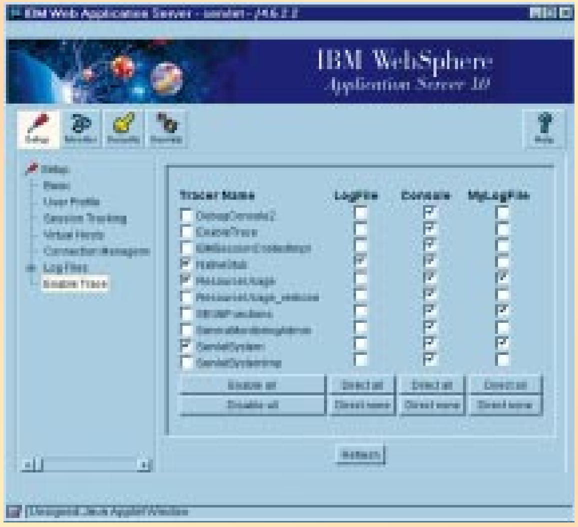
(Boulder, CO) - Rogue Wave Software, Inc. has announced that it's shipping Analytics.h++. The product is a new suite of components and classes that enables developers to focus on higher-level, conceptual issues in designing and building business data models.



Analytics.h++is an ideal solution for development teams building custom, in-house applications that perform sophisticated data analysis. Virtually all applications have four data-driven tasks: access, management, analysis and presentation. Within a software parts model, Analytics.h++ offers the powerful building blocks developers need to analyze a broad range of applications, such as financial modeling, signal processing, curve fitting, computer animation, simulation modeling and decision support. These powerful sets of classes and components give developers the flexibility to concentrate on building the appropriate business models without having to focus time and effort on analytics to manipulate data.

Analytics.h++ costs $2995 for the product and $895 for support per platform selected. For more information, call Rogue Wave Software tollfree at 888 442-9641 or visit their Web site at www.roguewave.com.

IBM Enhances and Expands WebSphere™



(San Francisco, CA) - IBM has announced the availability of its WebSphere Application Server and enhancements to the IBM WebSphere product line, including packaging the popular Apache HTTP Server with the WebSphere Application Server. IBM also announced availability of the NetObjects ScriptBuilder as the first development tool for the Websphere product line. WebSphere's products allow customers to implement e-business solutions.

The line consists of the WebSphere Application Server, IBM's Java servlet-based Web application server, which helps customers deploy and manage Web-based applications, and WebSphere Performance Pack, a Web facilities management software that supports rapid growth at high-volume Web sites. WebSphere Application Server, for Sun Solaris, Windows NT and IBM AIX, is available in the US for $795. For more information, visit IBM's Web site at www.software.ibm.com or www.ibm.com.

ILOG JViews™ Receives World Class Award from *Java Developer's Journal*

(Mountain View, CA) - ILOG S.A. has announced that ILOG JViews™received the World Class Award from ***Java Developer's Journal***. ILOG JViews is a 100% Pure Java class library for developing high performance, intuitive 2D graphic displays. It is used in conjunction with conventional graphical user interface components, including AWT, JFC and JavaBeans, to create interfaces such as network topologies, rich map displays and customized editors. ILOG JViews has been selected in recent months by a growing number of network management vendors as a key component for implementing sophisticated Web-based client applications.

For more information, visit ILOG's Web site at www.ilog.com or call 415 688-0200.

Object Matter Introduces Beta Visual BSF™

(Miami, FL) - Object Matter will soon announce the release of Visual BSF, a new product based on the BSF core library, that allows you to design persistent classes that do not have to inherit from BSF's base class and that can use Java's Integral objects, primitive data types, object references and data structures holding other persistent objects as persistent attributes.

Visual BSF is made up of two components: a GUI based schema mapping tool and a supporting core library. The product will be geared to building application servers for distributed systems in the enterprise. The heart of the product will be an object cache that will be shared by all clients within an application. The cache will boost performance by minimizing actual database accesses.

For more information, visit Object Matter's Web site at www.objectmatter.com or call 305 718-9101.

DeNova Releases Latest J'Express™

(Vancouver, BC) - DeNova has released the latest version of J'Express, a cross-platform Java installer, auto-updater and distributor. Now developers, or their assistants, can use a GUI interface to create Java installers integrated with Swing and an applet-based installer which supports the Java Plug-in.

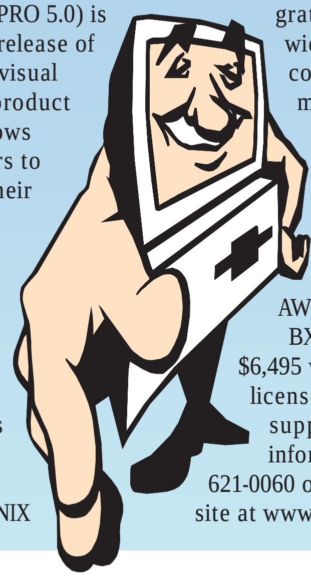
This new version also ends the requirement that target machines need the Java virtual machine (JVM) before a Java app can be installed. J'Express automatically installs the JVM for the most popular platforms.

Two editions are available. The Standard edition includes a wizard to create the installer and auto-updater, and lists at $99. The Professional version includes all of the features of the Standard edition plus customization, finding classes and distribution, and costs $499.

For more information, visit www.denova.com or call Nancy Atwell at 408 490-2852.

ICS Announces Deploy Anywhere™

(Cambridge, MA) - Integrated Computer Solutions (ICS) has announced its "Develop on UNIX, Deploy Anywhere" strategy, and a new product to support it. Builder Xcessory PRO™ 5.0 (BX PRO 5.0) is the major new release of ICS' integrated visual devfelopment product suite which allows UNIX developers to easily deploy their applications on multiple platforms. The BX PRO product family serves developers of business critical applications that need the reliability and scalability of UNIX servers along wih the flexibility to deliver applications via any client device.



New features include enhanced support for managing large projects, easier integration of third party widgets and user C++ components (the most extensive support for C++ GUI development), simplified migration from other GUI builders and support for Java AWT 1.1.

BX PRO 5.0 costs $6,495 with a floating license with one year of support. For more information, call 617 621-0060 or visit the ICS Web site at www.ics.com.

GUI Client/Server vs. Java-Internet/Web Paradigms:

THE GRIND

by Java George

"The Java-Internet/Web connection is a no-brainer."

As a Java evangelist, I field many questions about the so-called *new paradigm* of developing and deploying Java/Internet-based business applications. Will we have problems in our development phase? Will the application perform? Will the Microsoft lawsuits ruin the platform? Will Bill Gates buy the Brooklyn Bridge?

New paradigms, old paradigms, paradigms, schmaradigms. Paradigm is consultant talk. Remember all the talk about GUI-based client/server back when it was touted as the latest and greatest paradigm? People asked the same paradigm questions, "How do we partition and deploy it? How do we get performance? And where are the standards?"

People never asked the most important question back then, which is "What are the gains from building and deploying a GUI client/server business application?" If somebody ever did ask such return-on-investment (ROI) questions and actually got an honest answer, nobody would have bothered with GUI client/server computing. The applications cost a fortune to build and another fortune to maintain, and what was the payback? A more pleasing user interface.

Spurred by the adoption of the Windows platforms, the GUI gave users a nicer look and feel. This is what organizations invested so many millions of dollars for? The bang from the investment in GUI client/server was questionable at best. Maybe the organization got some slight reduction in the user learning curve, but I doubt it. An entire, highly profitable industry emerged around GUI-Windows user training.

In the early years of this decade countless resources of time and labor, not to mention hard cash, were spent converting business applications to GUI client/server. Countless tool vendors jumped on the GUI client/server bandwagon. Many product generations and thousands of acronyms later, here we are and what do we have to show for it? At best, we have a generation of users who know how to use a mouse and click on an icon, and a new generation of developers who are finally beginning to understand event-driven programming. So much for the ROI of the GUI client/server development paradigm.

Now the same people are asking the same questions about the Java-Internet/Web development paradigm. But again, they are not asking the only question that really matters: the ROI question. As far as I'm concerned, it is the only question to ask.

When it comes to the Java-Internet/Web paradigm, ROI is the only raison d'être for Java business applications. And its ROI can be spelled out in one word: accessibility. The Java-Internet/Web paradigm allows you to reach new users (new customers, suppliers, partners, anybody you want) as well as old users easily.

Accessibility translates into more business for you. Here is Java George's formula for Java-Internet/Web ROI: Customers + Accessibility + Product = Sale = Service = More/Better Business. You can plug in your own actuals. If they don't add up to more and better business for you, then skip Java and stick with what you are doing. (But check your math first because you don't want to make an error.)

The Internet is the primary catalyst behind the new paradigm of Java applications, just as Windows platform was the primary catalyst behind GUI client/server. With the Internet population running into the hundreds of millions worldwide and heading to the billions faster than the speed of Moore's Law, Java-Internet/Web is the Titanic of paradigms.

This means that a business providing products and services to people and organizations via Java-Internet/Web applications receives a material increase in prospective customers, and a potentially exponential increase in customers over time. Also, the organization that can interface with new and existing partners through business-to-business Java applications will reduce operating costs, shorten business cycles and increase its ability to respond to market conditions fast, further adding to the ROI.

Occasionally, people blame Java George for being overly optimistic towards the Java platform. But as far as the long term ROI for this new paradigm, I consider myself a conservative. Compared to the GUI client/server paradigm, Java-Internet/Web is a no-brainer.

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

George@sys-con.com

Ad

KL Group
Full Page Ad