ADVERTISER INDEX

ADVERTISER	URL	PH	PG
Adrenaline	www.adrenaline.com	877-890-9280	66
Aegis Consulting	www.aegisconsulting.com	703-448-5420	22
BEA Systems	developer.bea.com	408-570-8000	13
Cape Clear	www.capeclear.com	866-219-5944	31
Cerebellum Software	www.cerebellumsoft.com	888-862-9898	89
Cimmetry Systems	www.cimmetry.com	800-361-1904	56
Compoze Software	www.compoze.com	866-COMPOZE	37
Compuware	www.compuware.com/numega	800-4-NUMEGA	25
Corda Technologies	www.popchart.com	801-802-0800	43
Dice.com	www.dice.com	515-280-114	4
Elixir Technology	www.elixirtech.com/download	65 532-4300	69
Esmertec	www.esmertec.com	877-751-3420	35
Fiorano	www.fiorano.com	800-663-3621	71
Flashline.com, Inc.	www.flashline.com	800-259-1961	55
Generic Logic	www.genlogic.com	413-253-7491	34
Hit Software	www.hitsw.com	408-345-4001	63
IAM Consulting	www.iamx.com	212-580-2700	75
ICJD 2001	www.javacon2001.com		105
Inetsoft Technology Corp.	www.inetsoftcorp.com	908-755-0200	59
Infragistics, Inc.	www.infragistics.com	800-231-8588	3
Infragistics, Inc.	www.infragistics.com	800-231-8588	81
Instantiations, Inc.	www.instantiations.com	800-808-3737	47
INT	www.int.com	713-975-7434	38
Internet World Spring 2001	www.internetworld.com	800-500-1959	85
Intuitive Systems, Inc.	www.optimizeit.com	888-655-0055	57
Java Developer's Journal	www.sys-con.com	201-802-3020	107
JDJ Store	www.jdjstore.com	888-303-JAVA	112-113
JustComputerJobs	www.justcomputerjobs.com	877-905-NERD	14, 82
Macromedia	www.macromedia.com/ultradev	415-252-2000	15
Metrowerks	www.metrowerks.com	800-377-5416	21
Netaphor	http://cyberons.com	877 NETA4SW	56
NetDive	www.netdive.com	415-981-4546	83
No Magic	www.magicdraw.com	303-914-8074	7
Northwoods Software Corp.	www.nwoods.com	800-226-4662	66
OOP.com	www.oop.com	877-667-6070	53
Parasoft	www.parasoft.com/jdj1	888-305-0041	33
Pramati	www.pramati.com	408-965-5513	67
PreEmptive Solutions	www.preemptive.com	800-996-4556	91
Programix	www.jthreadkit.com		103
Programmer's Paradise	programmersparadise.com	800-445-7899	17
Progress Software	www.sonicmq.com/jdj	800-989-3773	2
Prosyst	www.prosyst.com	678-366-4607	45
Quadbase	www.quadbase.com	408-982-0835	93
QuickStream Software	www.quickstream.com	888-769-9898	44
Rational Software	www.rational.com/jdj2	800-728-1212	27
Recruitment Opportunities Section	http://careers.sys-con.com	201-802-3028	114-118
SecureByDesign	www.securebydesign.com	906-487-6520	49
Segue Software	www.segue.com	800-287-1329	11
Sitraka Software	www.sitraka.com/deploy	888-361-3264	19
Sitraka Software	www.sitraka.com/components	888-361-3264	51
Sitraka Software	www.sitraka.com/performance	888-361-3264	73
Sitraka Software	www.sitraka.com	800-361-3264	124
Slangsoft	www.slangsoft.com	972-2-648-2424	123
Softwired	www.softwired-inc.com	41-14452370	77
Sybase	www.sybase.com/partner/internetapplicationpartner4	800-8-SYBASE	23
SYS-CON Custom Media	www.sys-con.com	925-244-9109	101
SYS-CON Media Career Center	http://careers.sys-con.com		107
SYS-CON Media, Inc.	www.sys-con.com	800-513-7111	109
Tidestone Technologies	www.tidestone.com	800-884-8665	41
Togethersoft Corp.	www.togethersoft.com	919-833-5550	6
Unify Corporation	shop.unify.com	800-GO UNIFY	79
Verge	www.ejip.net		87
Virtualscape	www.virtualscape.com	877-VSCAPE4	39
Visualize Inc.	www.visualizeinc.com/jdj	602-861-0999	103
Wall Street on Java Technology	www.javaonwallstreet.com	212-286-0333	99
WebGain	www.webgain.com	408-517-3815	29, 65
Wireless Business & Technology	www.wireless-magazine.com	201-802-3020	95
XML DevCon Europe	www.xmldevconeurope2001.com		121
Zero G	www.ZeroG.com/updater	415-723-7244	61

Please visit the Web sites of our advertising partners who make it possible for us to bring you this Digital Edition (PDF) of *JDJ*

FREE **FORTE FOR JAVA COMMUNITY EDITION** P. 101

SUN'S NEW INTEGRATED DEVELOPMENT ENVIRONMENT FOR JAVA TECHNOLOGY

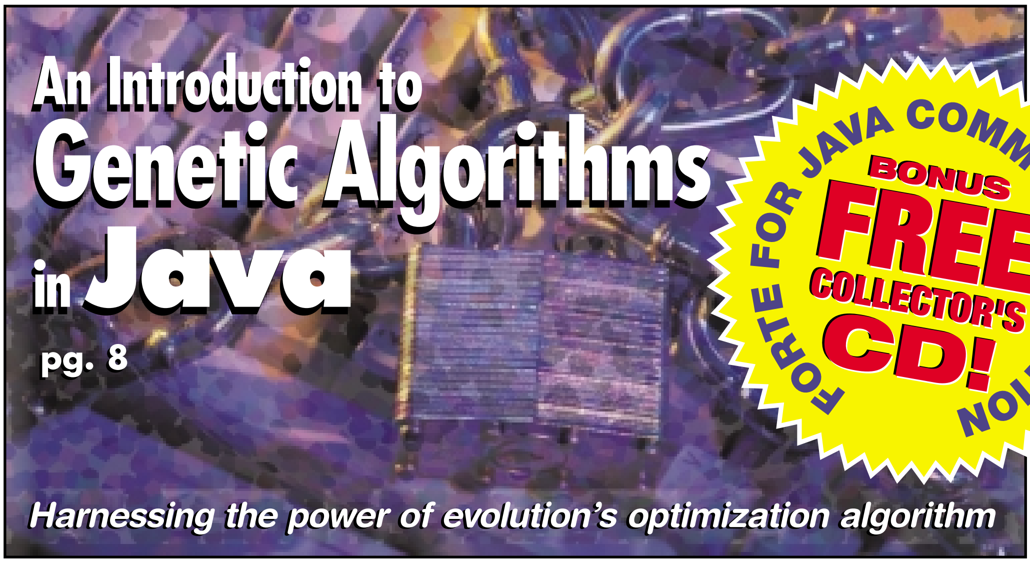
JAVA™ DEVELOPER'S JOURNAL

The World's Leading Java Resource

January 2001 Volume: 6 Issue: 1

JAVADEVELOPERSJOURNAL.COM

New York City
Feb. 27 – Mar. 2, 2001
IJD INTERNATIONAL CONFERENCE FOR JAVA DEVELOPMENT
Maximizing Java for the Enterprise
pg. 105



An Introduction to Genetic Algorithms in Java

pg. 8

Harnessing the power of evolution's optimization algorithm

FORTE FOR JAVA COMMUNITY EDITION BONUS FREE COLLECTOR'S CD!

SYS-CON MEDIA

EDITOR-IN-CHIEF:	SEAN RHODY
EXECUTIVE EDITOR:	M'LOU PINKHAM
ART DIRECTOR:	ALEX BOTERO
MANAGING EDITOR:	CHERYL VAN SISE
EDITOR/COPY CHIEF:	NANCY VALENTINE
ASSOCIATE EDITOR:	BETTY LETIZIA
ASSOCIATE EDITOR:	JAMIE MATUSOW
EDITORIAL INTERN:	SUZANNE AUGELLO
EDITORIAL CONSULTANT:	SCOTT DAVISON
TECHNICAL EDITOR:	BAHADIR KARUV
PRODUCT REVIEW EDITOR:	ED ZEBROWSKI
INDUSTRY NEWS EDITOR:	ALAN WILLIAMSON
E-COMMERCE EDITOR:	AJIT SAGAR

SUBSCRIPTIONS
FOR SUBSCRIPTIONS AND REQUESTS FOR BULK ORDERS, PLEASE SEND YOUR LETTERS TO SUBSCRIPTION DEPARTMENT
SUBSCRIPTION HOTLINE: 800 513-7111
COVER PRICE: $4.99/ISSUE
DOMESTIC: $49/YR. (12 ISSUES) CANADA/MEXICO: $69/YR.
OVERSEAS: BASIC SUBSCRIPTION PRICE PLUS AIRMAIL POSTAGE (U.S. BANKS OR MONEY ORDERS). BACK ISSUES: $12 EACH

PUBLISHER, PRESIDENT, AND CEO:	FUAT A. KIRCAALI
VP, PRODUCTION:	JIM MORGAN
SENIOR VP, SALES & MARKETING:	CARMEN GONZALEZ
VP, SALES & MARKETING:	MILES SILVERMAN
ASSISTANT CONTROLLER:	JUDITH CALNAN
CREDIT & COLLECTIONS:	CYNTHIA OBIDZINSKI
ACCOUNTS PAYABLE:	JOAN LAROSE
ADVERTISING ACCOUNT MANAGERS:	ROBYN FORMA
	MEGAN RING
ASSOCIATE SALES MANAGERS:	CARRIE GEBERT
	CHRISTINE RUSSELL
SALES ASSISTANT:	ALISA CATILANO
CIRCULATION MANAGER:	CHERIE JOHNSON
ASSOCIATE ART DIRECTOR:	DINA ROMANO
ASSISTANT ART DIRECTORS:	CATHRYN BURAK
	LOUIS F. CUFFARI
GRAPHIC DESIGNER:	ABRAHAM ADDO
GRAPHIC DESIGN INTERN:	AARATHI VENKATARAMAN
WEBMASTER:	ROBERT DIAMOND
WEB DESIGNERS:	GINA ALAYYAN
	STEPHEN KILMURRAY
WEB DESIGNER INTERN:	PURVA DAVE
SYS-CON EVENTS MANAGER:	ANTHONY D. SPITZER

EDITORIAL OFFICES
SYS-CON MEDIA, INC., 135 CHESTNUT RIDGE RD., MONTVALE, NJ 07645
TELEPHONE: 201 802-3000 FAX: 201 782-9600
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., 135 Chestnut Ridge Road, Montvale, NJ 07645. Periodicals Postage rates are paid at Montvale, NJ 07645 and additional mailing offices. POSTMASTER: Send address changes to: JAVA DEVELOPER'S JOURNAL, SYS-CON Publications, Inc., 135 Chestnut Ridge Road, Montvale, NJ 07645.

SEAN RHODY, EDITOR-IN-CHIEF

Process Improvements

One of the nice things about working for a large consulting company is that I have access to our strategic services department. These are the people who help develop strategies for our clients and research industry trends and conditions. I recently spoke with a few of our folks who are concentrating on the business-to-business (B2B) market. This discussion was part of what fueled this month's column.

Last year's hottest trend in B2B, the Net market, has cooled down considerably, for a couple of reasons. First is the general trend in the market for business plans with a concrete path toward profitability. In addition, the experience of the past 18 months has shown that one of the early principles for the development of these exchanges was fundamentally flawed.

Neutrality was seen as a prime consideration in the Net market, as it was felt that companies wouldn't wish to put their inventory (and thus a great deal of business intelligence) in the hands of a rival by running an exchange. But time has shown that the liquidity of the market is more important to its success than the neutrality. Liquidity is a basic measure of the community that a Net market attracts to its site. While neutrality is useful, coalitions of large companies are forming exchanges that become either the prime conduit for their business or one of the leading channels. This in turn draws others in the industry to participate because of the increased ability to do so. As my strategist friends like to say, "Liquidity trumps neutrality." A great example of such an exchange is Enron Online, which is generating incredible volume and was one of the first B2B markets to cross over from its primary industry (energy) into other types of business. These two factors have combined to reduce the number of Net markets and have led to a consolidation in this sector.

The market has also seen several large software providers go from a model in which they sold products to an ASP model in which they supply services and host software. ePit is a good example of such a plan. One effect these companies have had is to basically commoditize the trading engine market, even to the point of entering the end business themselves.

All this is driving us toward the next step in the trend – business process engines. We already have software that's approaching this idea, things like EAI and middleware, designed to allow loosely coupled systems to interact. But so far these systems have been largely about communication between applications. The next generation of systems will be about connecting business processes and allowing companies to collaborate within their supply chain to model, adopt, and change their overall processes without the need for intense IT redevelopment.

The vocabularies – languages needed to allow businesses to communicate at this level – are already available or under development. RosettaNet is one; UDDI is another. Most are about defining business processes in a specific context, so processes can be modeled and acted upon.

What's not available but will be in the next stage in the evolution of the business world are systems that will interpret these languages in a way that will allow companies to model their processes in a powerful, easy-to-use fashion. Companies like BEA are working on products like eCollaborate that are approaching this goal.

There's still a rub here. Years ago I helped develop an optimization system for the paper industry. Problem was it was extremeley complex. So is any business of appreciable size.

When you look at a business process, you can usually break it down into many smaller derived processes. It's one thing to define messaging that will allow applications to talk and interact. No matter how you look at it, application-to-application integration still leaves the development of business processes square in the hands of programmers – because they write the applications. It's quite another task to move the responsibility for that definition into the hands of business owners. But that's the ultimate value-add in this space – allowing a business to define how it wants to go to market based on business conditions, not software APIs.

It's still a ways away. The software has to be developed, and more importantly, the business community will have to get used to it. Once that's happened, though, we'll be adding incredible value to the industry. Now that's process improvement.

sean@sys-con.com

AUTHOR BIO
Sean Rhody is editor-in-chief of **Java Developer's Journal.** *He is also a respected industry expert and a consultant with a leading Internet service company.*

WRITTEN BY SILVANO MAFFEIS

Wireless Info Services Need Java Messaging

This year will be the genesis of mobile devices and wireless applications. In fact, several European and American carriers have begun rolling out high-speed, packet-oriented wireless networks based on General Packet Radio Service (GPRS) as well as other standards. I've also noticed interesting Java devices based on the Symbian operating system and the J2ME MIDP environment.

The combination of Java-enabled devices and packet-oriented wireless bearers is a compelling catalyst leading to the rapid adoption of "collaborative" and highly interactive applications, delivering the right information (according to our user profile) in real time.

What does this mean to us? Maybe a new type of Internet, based on zillions of gadgets exchanging information via wireless networks!

Such communicators and smart phones will always be on and capable of receiving information in real time. This means a new way of delivering information to the user. Browsing might not be the right model in this mobile world, since your wireless device is able to receive alerts and e-mails at all times. As a user, I don't want to chase or browse for information. I want to receive an alert when something interesting happens .

Reality check. Developing Java applications that connect mobile devices to server applications in a scalable and reliable manner is a challenge. In the wireless world a mobile application must be able to "speak" various protocols and cope with varying bandwidth and loss of network coverage (imagine issuing a stock purchase transaction from your mobile device while driving through a tunnel). Wouldn't it be nice if you could build mobile Java applications on a platform or middleware to take care of those issues?

Building a highly scalable platform to host mobile Java applications isn't easy either. I've been involved in this type of development and the biggest challenge was to devise a communications middleware that copes well with the specific aspects of wireless networks, namely, varying bandwidth, intermittent connectivity, and packet loss. We opted to base our middleware on the messaging paradigm and notably on the Java Message Service (JMS) standard because (1) messaging can deal with intermittent communication links using store-and-forward message delivery, and (2) messaging can hide long communication latencies by transmitting messages in an asynchronous mode (this means the sender of a message doesn't have to wait until the message has reached its destination). Also, messaging middleware can be implemented in a lightweight manner, meaning it can be deployed directly on a mobile device. The wireless JMS is born!

What does this mean? By taking advantage of a wireless-enabled JMS middleware, you can develop mobile applications that deliver information over various wireless bearers, in spite of sudden changes in bandwidth or network coverage. For application developers it truly offers a write-once-go-anywhere possibility irrespective of the bearer. The delivery of information is guaranteed by the JMS store-and-forward message-queuing mechanism, and its timely delivery to large groups of receivers is made possible by the JMS publish-and-subscribe model and by taking advantage of the multicast capabilities of wireless bearers.

This type of JMS middleware acts as a highly versatile bridge between server-side applications running on a J2EE platform and J2ME applications running on a mobile device. If a server application transmits information to a mobile device that's turned off, nothing is lost as the JMS middleware will store and deliver the information automatically and transparently. The same occurs when the device transmits information to a server. The information, which takes the form of messages, is stored on the device and delivered to the server automatically as soon as a wireless connection is physically possible.

What does this solve? The commonality between mobile commerce platforms, wireless financial data feeds, location services, instant messengers, and multiuser games is that they all embody a distributed systems architecture in which various pieces of (Java) code need to communicate by wireless and wireline protocols. JMS has proven viable when complex distributed systems need to be developed and deployed quickly.

As we witness innovative wireless services and applications, a wireless messaging middleware could stand at the vanguard of this revolution!

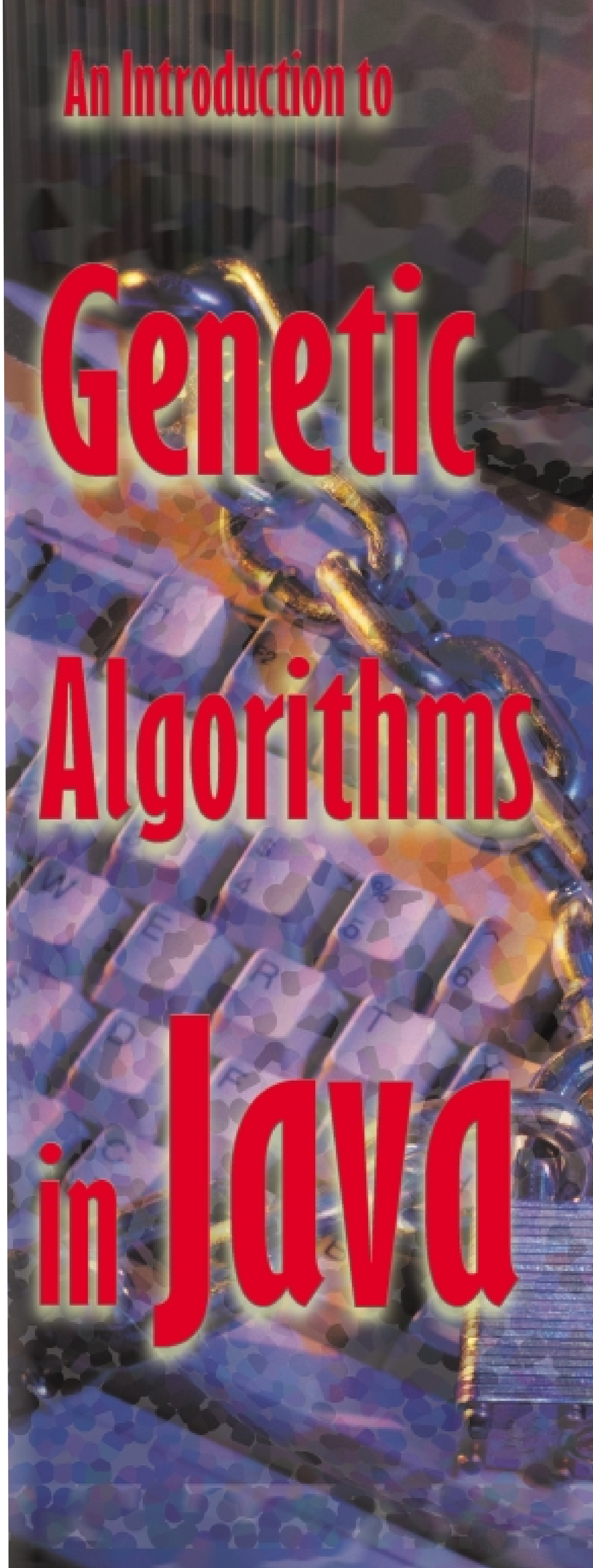
silvano.maffeis@softwired-inc.com

AUTHOR BIO

Dr. Silvano Maffeis is CTO at Softwired (www.JavaMessaging.com), a leading vendor of application-to-application messaging solutions. He is the codeveloper of iBus, a JMS middleware for wireless and wireline systems.

FEATURE

An Introduction to Genetic Algorithms in Java



Written by Michael Lacy

Starting about 3.5 billion years ago with bacteria, nature embarked on the grandest of all algorithms: the evolution of highly complex and dynamic machines capable of interacting with and adapting to their environments in order to solve problems. We know these machines as plants and animals.

One look at the genetic code of even the simplest living organism reveals a structure that's enormously complex and efficiently tuned, ensuring the survival of the organism in its environment. We might even use the terms *fault-tolerant, highly parallel, high performance,* and *ubiquitous.* Don't forget that nature accomplished this extraordinary programming feat without a single developer coding an exhaustive list of if–then rules and switch statements to account for all possible scenarios. It was simply based on a random set of interactions with the fittest organisms surviving to replicate their genetic code into the next generation.

With the advent of the Internet over the past decade, an entirely digital world has arisen in which Web sites and applications are the organisms fighting for survival in a highly complex, internetworked environment replete with computer viruses, server crashes, and the like – an environment in which only the fittest will survive. As such, it's my belief that more sophisticated means of software development are needed to build Web applications capable of interacting with and adapting to the complexities of the new digital world thriving within our computers.

One simple, yet extremely powerful, technique that will likely play a role in the evolution of the Internet (and the Web applications that live within it) borrows several concepts from the biological world and transforms them into bits and bytes with the goal of building adaptive software systems.

This article is the first of a two-part series that examines a technique from the AI community called *genetic algorithms*, which borrows concepts from biology to solve complex and often nonlinear problems encountered in the world of computer science. This article will introduce you to the concepts of genetic algorithms and discuss why Java is well suited to their implementation. The next installment will investigate the details of implementing these algorithms in Java. It's my hope that after reading these articles, you'll think a little differently about software development and its future. Genetic algorithms provide a problem-solving technique that's too powerful to ignore.

Genetic Algorithms

First a little history. Genetic algorithms were born out of the idea of evolutionary programming introduced by I. Rechenberg in the 1960s. John Holland, a professor at the University of Michigan at the time, is credited with the invention of genetic algorithms following the publication of his 1975 book *Adaptation in Natural and Artificial Systems.* In his book Holland formulated the basics of genetic algorithms as models of machine learning that derive their behavior from concepts of biology's theory of evolution. It was one of Holland's students, David Goldberg, who popularized the use of genetic algorithms when he was able to solve a difficult problem involving gas-pipeline transmission for his dissertation in 1989.

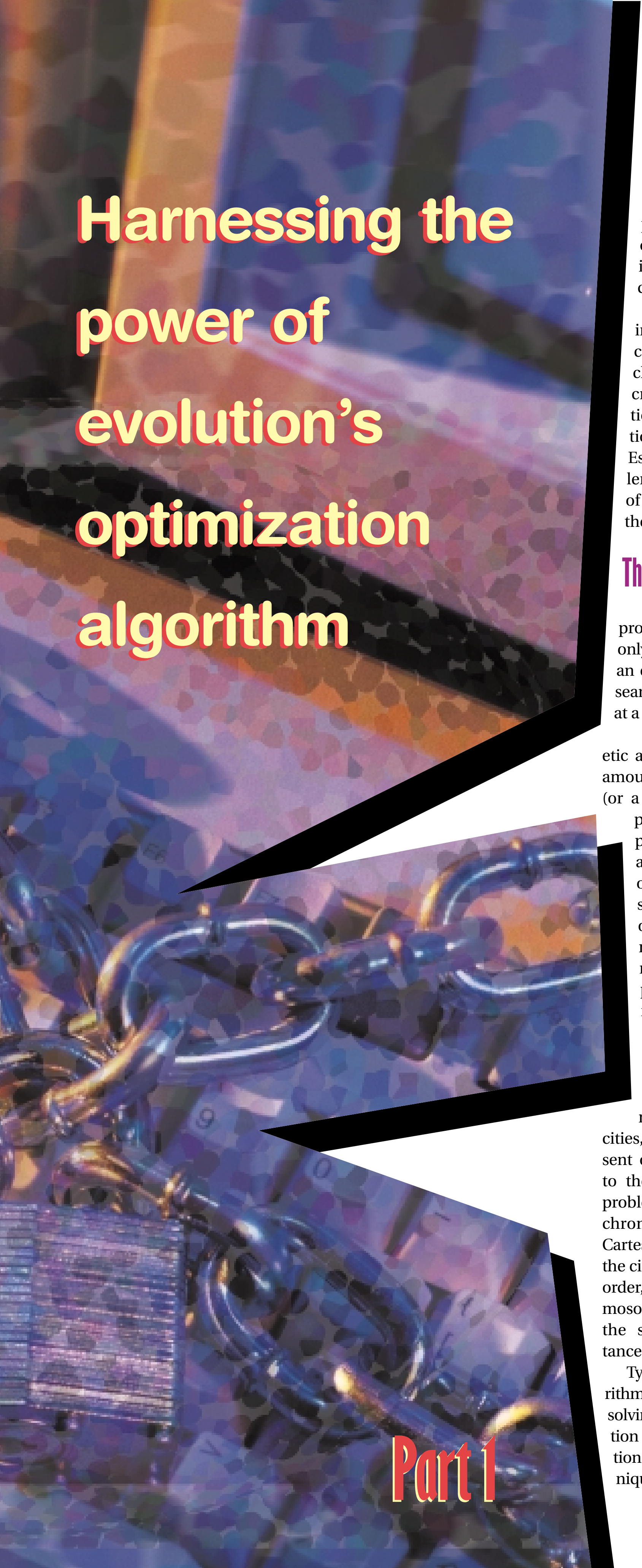
That said, what exactly is a genetic algorithm? What are they used for? What are the benefits over traditional programming techniques? How does Java fit into this? I'll attempt to answer these questions so you'll have the foundation needed to start implementing genetic algorithms (see Figure 1).

Darwin in Your Computer

A genetic algorithm can be thought of as a model for machine learning in which a population of randomly created individuals goes through a simulated process of evolution – a digital survival of the fittest where

Harnessing the power of evolution's optimization algorithm

Part 1



each individual represents a point in the problem's solution search space. Using correct terminology, an individual is represented by a chromosome, which consists of several genes. Genes are essentially the parameters of the problem to be solved. A collection of chromosomes is considered a population and is the fundamental unit on which a genetic algorithm operates. Once the algorithm is set into motion, individuals are selected from a population and combined in a process called *crossover* to create a set of children. The children are randomly mutated to create a new set of chromosomes to be reinserted into the population. Once enough children chromosomes have been created to replace a population, a generation is said to have passed.

With each generation, all the chromosomes are evaluated according to some fitness criterion that's a measure of the strength of the chromosome compared to the rest of the population. Only the fittest chromosomes survive into the next generation where the selection, crossover, and mutate process begins anew. After a number of generations have elapsed, the best chromosome is selected from the population and represents the optimal solution to the problem being solved. Essentially what's happening is that a random set of solutions to a problem within a given search space is created and evolved over an amount of time to find an optimal solution. A concrete example will help clarify the concepts described above.

The Traveling Salesman

The traveling salesman problem (TSP) is a classic computer science problem in which a salesman must traverse a number of cities, visiting each only once, while minimizing the distance traveled. For the case of 20 cities, an exhaustive search method that examines all possible routes dictates a search through over 2.4 billion billion (20!) permutations which, if evaluated at a rate of 500 million per second, would take over 150 years to complete.

Employing a genetic algorithm reduces the amount of time to seconds (or a fraction thereof, depending on the computing power available) and produces the optimum solution in some cases and a near optimal solution in most others. The representation of this problem in the genetic algorithm domain consists of cities with their x and y coordinates serving as individual genes. A chromosome is a list of cities, in order, that represent one possible solution to the traveling salesman problem. The fitness of the chromosome is then the Cartesian distance between the cities when traversed in order, with the fittest chromosomes being those with the shortest overall distance (see Figure 2).

Typically, genetic algorithms have been utilized in solving complex optimization problems when traditional programming techniques (such as exhaus-

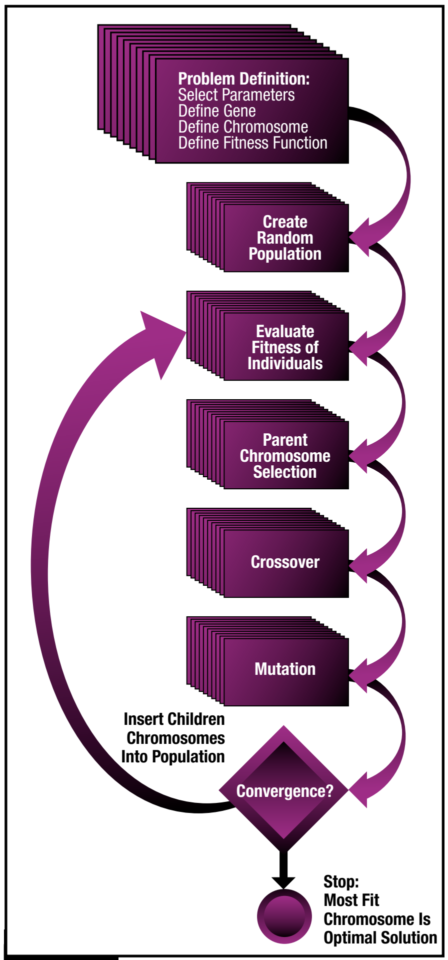

Problem Definition:
Select Parameters
Define Gene
Define Chromosome
Define Fitness Function
Create Random Population
Evaluate Fitness of Individuals
Parent Chromosome Selection
Crossover
Mutation
Insert Children Chromosomes Into Population
Convergence?
Stop: Most Fit Chromosome Is Optimal Solution

FIGURE 1 Flowchart for a genetic algorithm

tive search, analytic optimization, and line minimization) fail to arrive at a solution in a reasonable amount of time. Genetic algorithms confer the following advantages:

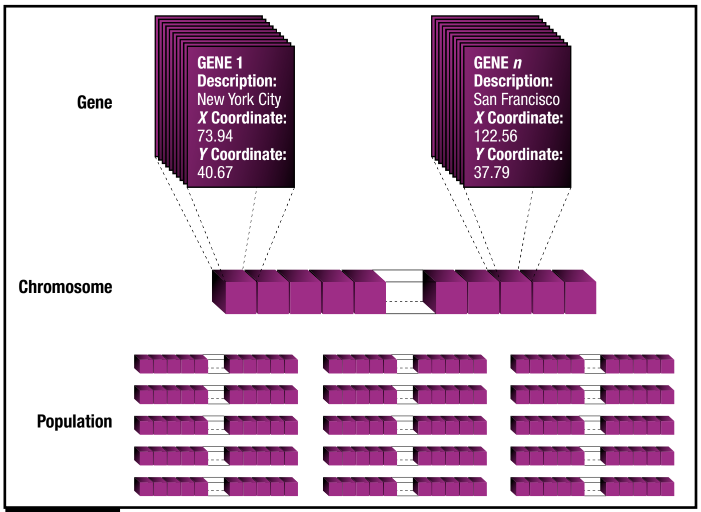


FIGURE 2 Graphical representation of the traveling salesman problem

- They evaluate several solutions simultaneously, covering a large search space.
- They work well in parallel implementation.
- They optimize parameters with very complex cost functions.
- They create a list of optimal solutions, not just a single solution.
- They work with various data types.

This leads to the next question: Why use Java?

Why Java?

As you can see, genetic algorithms can become computationally expensive depending on a number of parameters (including the size of the population, the complexity of the fitness function, the size of the chromosome, and the time to converge on an optimal solution. Thus, in choosing a language for implementation, weighing the benefits of using Java versus using a compiled language such as C or C++ is essential. For Java to be a viable language for genetic algorithm implementation, it must present significant advantages to make up for its degraded performance as compared to other compiled languages. And it does! The advantages of Java are particularly evident in the area of distributed computing.

Simple and Object-Oriented

Given the dynamic memory requirements for a genetic algorithm, Java's garbage collector relieves us from having to allocate and deallocate memory for chromosomes in each generation. This allows us to focus specifically on coding the problem at hand and not worrying about memory management details. Also, the use of objects allows us to create an endless number of problem encodings and still use the genetic algorithm framework. This means that once the basic algorithm structure is developed, implementing a genetic algorithm to solve new problems becomes a matter of defining the problem and its encoding. Next month we'll take an in-depth look at what this means during implementation .

Robust and Secure

Java was designed for creating software that's highly reliable and capable of operating in distributed environments. As developers start to move genetic algorithms from a single CPU to a network of parallel and distributed CPUs, robustness and security are essential. Think of partitioning a genetic algorithm into a number of populations and letting them evolve separately in parallel, frequently distributing the most fit from each population into all the populations. JavaSpaces presents itself as an excellent candidate for moving genetic algorithms into a distributed environment.

Architecture-Neutral and Portable

As referenced above, the real power of genetic algorithms can be obtained in parallel and distributed environments. With Java's platform-neutrality, populations and the algorithm to evolve them can be distributed among a network of computers for processing, provided that a JVM is available. Don't worry about the implementations for different operating systems and CPUs. Think of the SETI@home project that utilized over two million PCs connected to the Internet to churn through radar data in the search for extraterrestrial intelligence. Genetic algorithms are ideal candidates for use in such a distributed environment, with Java being the obvious language of choice given its portability. Computing power is no longer an issue; there will be more than enough to go around.

Building Blocks

Now that we've briefly examined the nature of genetic algorithms and why Java makes sense as the development language of choice, let's take a more detailed look at the fundamental components that make up a genetic algorithm. For the sake of simplicity, we'll cover the most basic implementations of genetic algorithms and introduce the essential core concepts. I highly recommend further research and study if genetic algorithms spark a deeper curiosity. A number of resources are available on the Web for such study.

Genes

A gene can be defined as the encoding of a single parameter in a genetic algorithm. A gene can take many forms depending on the problem definition. For the traveling salesman problem, a gene represents a city and its longitude and latitude coordinates. However, when solving a high-order, nonlinear, partial differential equation, a gene can represent one of the variables to solve for and its range of acceptable values.

This highlights the two main flavors of genetic algorithms: permutation-encoded versus real-parameter. In the former version, the goal is to find the optimal ordering of a set of genes such as in the TSP. As for the latter, an example of a real-parameter genetic algorithm is finding x and y such that the following function is minimized: $f(x, y) = 2x * \sin(3 * y) + 4y * \cos(5 * x)$.

Historically, genes were represented as sequences of 1's and 0's. However, this approach has not been shown to yield better performance and introduces a layer of complexity as a translation is needed between the actual values of parameters and their binary representation. In addition, handling genes as objects in Java makes the implementation more intuitive and can be extended to make them reusable across different genetic algorithm implementations. (More on this in next month's article.)

Gene Pool

Much like its biological equivalent, the gene pool for a genetic algorithm is a collection of all the available genes. From the gene pool, chromosomes are created at the beginning of a genetic algorithm by randomly drawing genes from the gene pool and assembling them to build a chromosome that represents one solution for the search space defined for the genetic algorithm.

Returning to the examples mentioned above, the gene pool for solving the traveling salesman problem consists of one gene per city to be traversed. For the case of 20 cities, there will be 20 genes in the gene pool from which random chromosomes will be created. For real parameter genetic algorithms, such as minimizing the function f(x, y), the gene pool will consist of two genes, one representing the variable x and the other representing the variable y.

Chromosomes

Continuing with definitions, a chromosome is a collection of genes representing a single point in the solution search space. The fitness of a chromosome is determined by a cost function determined prior to the execution of the genetic algorithm. Again, returning to the traveling salesman problem, the fitness of a given chromosome is the sum of the distances between the cities when traversed in the order specified by the chromosome. For the real parameter chromosome (f(x, y)), the fitness is the result of substituting the *x* and *y* values back into the original function and performing the calculation. Note that the fitness of a chromosome tells you nothing about its strength relative to other chromosomes; rather, it's a raw evaluation of the chromosome's fitness. It's at a higher level that fitnesses are compared and selection proceeds according to the rules of a genetic algorithm. This higher level is the population.

Population

A population is a collection of all the chromosomes being evolved in a genetic algorithm. As new chromosomes are created and reinserted into the population, less fit chromosomes are replaced and only the most fit survive into the next generation. As mentioned previously, it's here that the process of digital evolution occurs, as the fitness of the competing chromosomes is compared in order to select parent chromosomes to reproduce.

Depending on the search space for a given problem, population size can range from a few dozen chromosomes to several hundred, several thousand, or more. Given the fact that a chromosome represents a single point in the solution search space, for problems with extremely large search spaces (such as the 20-city TSP), it makes sense that a large population size is needed to cover as much of the space as possible. Otherwise, the genetic algorithm may approach a local minimum and converge toward it, rather than the global minimum. Convergence is a core issue in genetic algorithm implementation, and I highly recommend further examination outside of this article to gain additional insight.

Genetic Algorithm Operations

Now that we've discussed the requisite components of a genetic algorithm, it's essential to understand how a genetic algorithm operates on each of the components to create a simulated evolutionary environment that combs a search space for an optimal solution. There are five elementary genetic algorithm operations:

- ***Fitness evaluation:*** With the examination of a chromosome and its role within a population, we talked briefly about fitness evaluation and its importance. The proper definition and evaluation of a fitness function is critical to the success of the genetic algorithm. It's the means by which chromosomes are compared to one another to determine the most fit individuals. The primary goal here is differentiation between the more fit chromosomes and the less fit chromosomes. Remember, it's survival of the fittest.

- ***Selection:*** This is the method by which chromosomes are chosen to reproduce in order to create children for the next generation. The goal of selection is to choose individuals that, on average, are more fit than others to pass on their genes to the next generation while, at the same time, maintaining genetic diversity. If a population consists of identical individuals, genetic diversity is lost and it's difficult for the genetic algorithm to explore different regions of a search space.

  Several different methods are available for genetic algorithm selection, but for the sake of simplicity and brevity I'll focus on a technique labeled *tournament selection.* With this technique, a group of individuals is selected at random and the two most fit are selected for reproduction (i.e., they win the tournament). Keeping the tournament size small (4–8 chromosomes) ensures genetic diversity as the group is small, and what appears to be the most fit within the group may actually be a weak chromosome when compared with the entire population.

- ***Crossover:*** Once two parent chromosomes are selected, they reproduce two child chromosomes via the crossover operation. One of the parameters of a genetic algorithm is the crossover probability (typically 75–90%) that represents the statistical chance that two given chromosomes will cross over. For each potential crossover, a random number between 0.0 and 1.0 is generated. If the number is greater than the crossover rate, then crossover doesn't occur and the children chromosomes are exact replicas of their parents. If crossover does occur, then the parents randomly exchange genes to create new chromosomes.

  There are three types of crossover covering a wide range of problem encodings:
  - ***Permutation encoding with unique genes:*** In this case, a gene can appear only once within a chromosome. One example is the TSP. Each city may appear only a single time within the chromosome.
  - ***Crossover operating on the permutation encoding, with the exception that genes don't have to be unique***: Let's imagine that we have a genetic algorithm that's evolving a musical piece within the key of C. All the notes in the key of C are viable and can be repeated indefinitely up to the size of the chromosome.
  - ***Real parameter chromosome crossover:*** In a real parameter chromosome, each gene will represent a parameter to be applied to a given cost function. Building on the function, f(x, y) described earlier, two parent chromosomes will have genes for the *x* variable, both representing different values. A method for crossing over the two genes might involve creating a new gene for the *x* variable with the value being the average of the two parent genes.

  Crossover is another essential genetic algorithm operator that ensures genetic diversity within a population. The conceptual goal of crossover is, over time, to combine the good portions of chromosomes into newer and better chromosomes. For a better understanding, see Figure 3. I highly recommend further exploration of the crossover operator before attempting to implement your own genetic algorithm.

- ***Mutation:*** Similar to crossover in that it randomly modifies chromosomes, it operates on only a single chromosome at a time (see Figure 4). As with crossover, there's a probability associated with the occurrence of mutations, albeit a small one (typically 5–25%). Yet again, returning to the TSP, a typical mutation can include randomly selecting two endpoints within a chromosome and reversing the order of the genes. Several mutation techniques that can be utilized depending on the problem encoding won't be discussed here. It's important to remember that mutation is a fundamental operator for ensuring genetic diversity within a population, which translates into a better coverage of the search space.

- ***Insertion:*** This is the final algorithmic step to conclude a generation in a genetic algorithm. Insertion is the process of introducing children chromosomes into a population and removing the less fit chromosomes. One common technique for insertion utilizes a technique called *elitism* in which the *n* best chromosomes of a population are kept for the next generation and the rest are replaced with new children. This ensures that the most fit chromosomes survive into the following generation and have the opportunity to reproduce again.

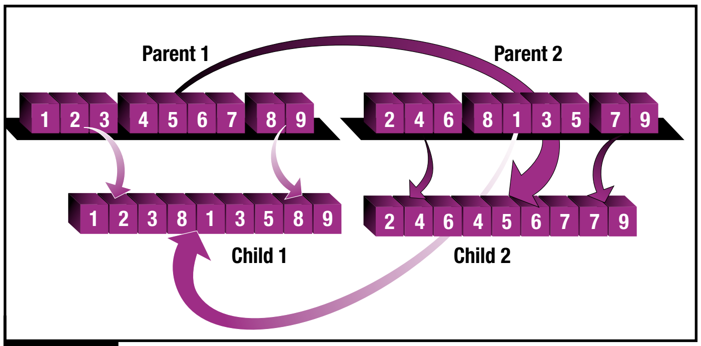


FIGURE 3 Graphical representation of the crossover operator

Genetic Algorithm Considerations

By now you should have a basic understanding of what a genetic algorithm is and how it works. Let's now quickly look at some considerations when implementing a genetic algorithm.

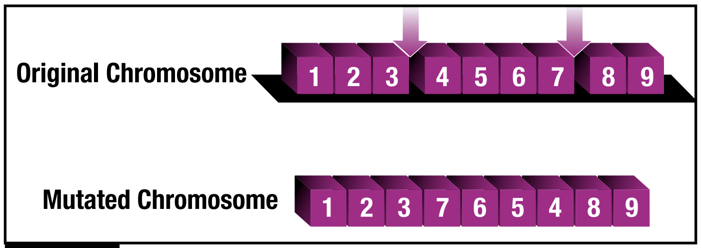


FIGURE 4 Graphical representation of the mutation operator

Convergence

The goal of implementing any genetic algorithm is convergence on an optimal solution for a given search space. Convergence will be affected by numerous factors associated with the implementation of the genetic algorithm, such as parameter encoding, population size, crossover and mutation rates, and selection technique. Depending on the problem being solved, these factors are usually determined only by experience working with genetic algorithms of all flavors. My recommendation is to start coding!

Performance

Performance is an issue that has constantly plagued genetic algorithms due to their heavy-duty processing power requirements. With the combination of Moore's Law and the increased availability of highly parallel, distributed computing power, I don't think performance will be an issue in the near future.

Real-World Applications

Here's the number one barrier to acceptance of genetic algorithms as a practical programming technique: real-world applications. Genetic algorithms have resided primarily in academia solving classic computer science problems. Their use in business and commercial environments is highly unproven. As computing power becomes more readily available, I think we'll see an increase in adaptive software systems with genetic algorithms at their core.

One particular area of work that may break down the wall is security. Researchers have begun to develop operating systems modeled after the immune system of animals. As new viruses invade the system, strategies are evolved to combat the virus, remove it from the operating system, and identify similar attacks in the future. And with the proliferation of highly sophisticated attacks on Internet sites, such an "immune" system offers a much better (and quicker) response than waiting for a human to recognize the attack and code a patch to fix it or close ports on a firewall to deny it.

Another interesting outbranching of genetic algorithms is the field of genetic programming pioneered by John Koza. Without getting into the details, genetic programming is essentially using genetic algorithms with the genes representing programmatic constructs (e.g., AND, OR, IF, THEN, +, and -). What's evolved are chromosomes representing computer programs. It's an exciting field that's worth a deeper look.

Conclusions

The goal of this article wasn't to encourage you to implement genetic algorithms in your code tomorrow, but rather to inform and educate you about one technique for building software capable of adaptation. As the Internet continues to grow at a furious pace, a new digital world is being created that operates in the language of 0's and 1's. The organisms fighting for survival are the Web sites that you and I create on a daily basis. Whether fighting for survival in the sense of attracting new customers or warding off the latest computer hacker, adaptability will be crucial to survival in the complex digital world. Hopefully this article has sparked a newfound interest in software development and its future. If so, stay tuned for the next issue of ***JDJ***, in which I'll demonstrate a simple implementation of a genetic algorithm.

AUTHOR BIO

Michael Lacy is an engineer for the platform development group at Shutterfly, an online photo service, where he develops Web-based solutions for digital image printing, enhancing, and sharing. He's also a certified Java programmer and developer.

mlacy@shutterfly.com

How to Develop Message-Driven Beans

These beans fill a void in the EJB architecture

WRITTEN BY
JASON WESTRA

This month in EJB Home I'll show you how to build a message-driven bean. Knowledge of this EJB will enhance your toolkit for developing asynchronous Enterprise Java applications – whether they're mission-critical or not.

The Enterprise JavaBeans specification 2.0 introduced another bean into the mix. One of the primary goals for the EJB 2.0 release was to define how EJB interacts with the Java Message Service (JMS), thereby defining how these Enterprise Java APIs interact within the Java 2 Enterprise Edition (J2EE) platform. Until version 2.0 of the specification was released, the defined enterprise beans, entity and session, supported only synchronous operations. However, the need to perform asynchronous processing is critical in many high-volume applications.

Years ago, for example, I worked on a messaging system for NORAD at Cheyenne Mountain in Colorado. The system needed to process thousands of messages a second to multiple subscribers. The messages needed to be in the correct order, and not a single message could be lost – not a single one! This is an example of a mission-critical messaging application.

While messaging systems are crucial to mission-critical applications, often there's less critical yet useful functionality that can be off-loaded to an asynchronous process. For instance, JMS is a practical solution for sending e-mails and logging events. Using a messaging service to perform the processing allows for quicker response time to an end user, as he or she isn't held up while the system sends the e-mail.

JMS in a Small Nutshell

Before building a message-driven bean, I'll cover some basics of the Java Message Service. JMS is an API for message-based systems. It's generic enough to allow existing messaging products like IBM's MQSeries to incorporate it into their offering, yet powerful and flexible enough to provide a rich feature set for enterprise-level messaging. JMS provides publish-and-subscribe through the "topics" and point-to-point messaging through "queues." It also provides the ability to acknowledge that a message was received and is being processed as well as guarantee message delivery even when the receiver (e.g., durable subscriber) isn't available. Last, JMS messages can be transactional to provide reliable delivery and handling of mission-critical messages such as those in a military command-and-control system.

With that quick overview, let's dig into building our message-driven bean to see how JMS and EJB fit together.

Overview of Message-Driven Beans

A message-driven bean is a JMS MessageListener that indirectly consumes messages from its container. The container delegates a received message either to an existing method-ready instance or to a new instance allocated to handle the message. Message-driven beans are stateless; therefore, any instance may service a message equally. Likewise, similar to stateless session beans, message-driven beans are anonymous, having no identity to a client.

Why are message-driven beans so cool? Why not just use straight JMS in your EJB application? Why are message-driven beans an exceptional addition to the EJB specification? The reasons are numerous.

First, message-driven beans provide a component model around JMS messaging. They support both topics and queues, allowing publish-and-subscribe and point-to-point messaging in a component execution environment, which is particularly important for scalability and portability.

This environment promotes scalability through its efficient resource management while processing messages. When developing with message-driven beans, you take advantage of the container's ability to pool bean instances and resources such as JDBC connections. You no longer have to code your own queue and topic pooling classes.

Concerning portability, the component execution environment (e.g., container) for message-driven beans offers portable access to resource factories and resource environment entries. The container uses structural information from the bean's deployment descriptor to link references to existing resources in the environment in a portable manner. Also, with message-driven beans you don't have to develop special "start-up" classes that initialize JMS Destinations at application server start-up.

Second, message-driven beans ease the development of asynchronous systems for both clients and Bean Providers. Message-driven beans are a snap to use from clients. The client looks up the JMS Destination by using JNDI and sends the message. Since message-driven beans have no home or remote interface, they're simply represented by a topic or queue JMS Destination (see Figure 1).

Message-driven beans live by the threading restrictions imposed on all EJBs; namely, you may not start your own threads in the bean (see EJB specification 2.0 for details beyond this article). This restriction eases development of message-driven beans because the container performs concurrency control for you. You don't have to worry about developing thread-safe business logic.

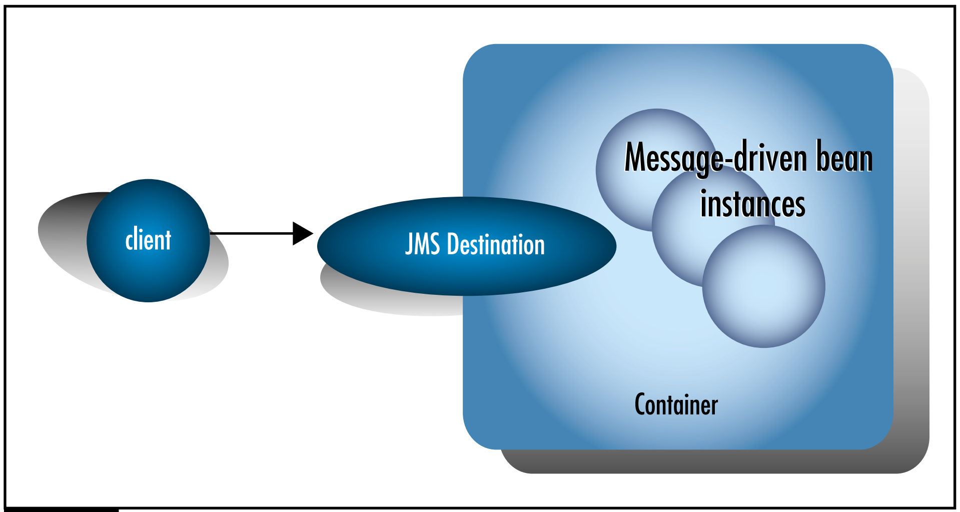


FIGURE 1 Client contract for message-driven bean lookup

"How-to" Guide

Let's roll our sleeves up and build a message-driven bean. In general, developing a message-driven bean entails two steps: code the bean class and create the bean's deployment descriptor. I've chosen to use the WebLogic Server 6.0 beta (the only product offering message-driven beans at the time of this writing) to create my message-driven bean, but the component should be deployable to other EJB 2.0–compliant servers. Listing 1 contains the code of an example bean that performs remote logging. Use it as a guide as I cover the basics of coding your first bean.

Coding a Message-Driven Bean

Realizing that a message-driven bean is a JMS MessageListener, your first step should be to create a Java class that implements the javax.jms.MessageListener interface. Your bean class must be declared public and must not be final or abstract. It may implement the MessageListener interface directly or inherit from another class that implements the interface, thereby implementing it indirectly. When you implement the MessageListener interface you must code an onMessage method, which performs the business logic required to process the message. The onMessage method shouldn't throw exceptions, as they won't be handled by the client application and may cause the container to choke on ejbRemove (see below). The code in my message-driven bean example (Listing 1) is as simple as it gets. It logs to system out, which is presumably redirected to a log file of interest. Notice how onMessage has a try block that contains all code and catches java.lang.Exception to make sure nothing is thrown from the method.

Your message-driven bean must have an empty constructor, which is called from the container when it creates a new instance of the bean. Likewise, you must code an ejbCreate method that also takes no arguments. The ejbCreate method should be public and must not be static or final. These rules allow the container provider to extend your bean class, if that's how it implements its container, and allow the container to call the ejbCreate method when it creates a new instance of the bean. Finally, the ejbCreate method must declare no application exceptions in its throws clause. Clients cannot catch exceptions thrown by message-driven beans.

Last, your message-driven bean must define an ejbRemove method. The ejbRemove method follows the same rules as ejbCreate as far as the signature. If your message-driven bean has opened a resource in its ejbCreate or onMessage method, you must release the resource(s) in your bean's ejbRemove method. It's important to note that execution of ejbRemove by the container isn't guaranteed. The ejbRemove method may not be called in cases where there's an EJB server crash or a system exception is thrown back to the container. It's the responsibility of the application utilizing message-driven beans to provide a way to regain resources when necessary. It's not the responsibility of the Bean Provider to do code recovery logic. The complexity of managing a resource recovery process beckons the Bean Provider to open and close resources in onMessage rather than waiting for ejbRemove to be called. A try block with a finally clause in onMessage can ensure that the resources are closed when it finishes, no matter if an exception occurs or not. However, there's a performance penalty incurred with this approach. Pick your poison.

Creating the Deployment Descriptor

As with any EJB, you must create a deployment descriptor, which defines the structural and application assembly information of your component. This information assists the container in managing your component. Three important steps define your message-driven bean's deployment descriptor. *Note:* Listing 2 is an excerpt from my ejb-jar.xml deployment descriptor. Use it as a guide for creating your bean's descriptor.

First, associate your message-driven bean to a JMS Destination. To send messages to the bean, a client uses this destination rather than a home and remote interface, which don't exist in the message-driven bean's component model. When associating your bean with a queue destination, don't associate multiple message-driven beans with the same queue. JMS doesn't define how multiple message consumers will handle messages, and your application will behave strangely to say the least.

I've chosen to map my example, MessageLogBean, to a Topic. This allows multiple message-driven beans to subscribe to the topic and log to various sources, send e-mail alerts on critical messages, and so on. Topics allow for easy third-party integration. If you're interested in hearing about what's being logged as well, simply create a message-driven bean that subscribes to the same topic!

Second, map resources are needed in the message-driven bean's environment to perform its business logic (remember, resources and resource factories allow your bean code to be portable across servers). My bean has no need for resources, but if I logged to a URL instead of sysout, I'd probably map to a URLConnection factory defined in the server's available resources.

Since I used WebLogic Server, I defined a JMS Server and the Topic for my message-driven bean's destination in the server's config.xml file (see Listing 3). WebLogic Server maps the Topic to the bean in its proprietary deployment descriptor, but since this association is declarative, you may map it as you see fit for your own server.

Last, set your message-driven bean's transactional properties. While entity beans can only be container managed, message-driven beans, like session beans, may manage their own transactions. Whether your bean is container managed or bean managed has an impact on the transactional attribute for your onMessage method as well as the guaranteed delivery of your message handling.

If your bean is bean managed, remember that only the onMessage method should define transaction boundaries. It must commit or roll back the transaction before it returns. With bean-managed, message-driven beans, the act of dequeueing the message is not within transactional boundaries (EJB 2.0 specification, section 16.2.6). Therefore, message delivery isn't guaranteed with bean-managed transactions.

It's advised to use container-managed transactions instead. This lets the container assume responsibility for the transactional scope. It will manage the transaction and guaranteed message delivery for you. The act of dequeueing, calling other EJBs, and database interactions are executed within the transaction.

A message-driven bean that has a transaction type of container-managed can declare only a transaction attribute of NotSupported or Required. Distributed transaction contexts don't propagate to message-driven beans, so there's never a transaction context associated at the time of message arrival. Thus a bean either processes its message with no transaction (NotSupported) or the container creates a new transaction for the bean to process within (Required).

Wow! You're done developing your message-driven bean. That's right, there's no need to develop a remote interface or a home interface for message-driven beans.

Restrictions on Message-Driven Beans

To apply the EJB component model to JMS, certain restrictions evolved that you must be aware of. Among these restrictions is the inability to order messages. The multithreaded container model may process messages in any order, so your message-driven beans must be prepared to process messages out of order. The inability to automatically order messages is a liability for mission-critical applications such as the command-and-control system for NORAD. I see message ordering as an opening for J2EE server vendors to provide a value-added feature that distinguishes their message-driven bean implementation from others.

You shouldn't use the JMS acknowledgment API to acknowledge a message that was successfully processed. Acknowledgment is the container's responsibility. If the bean uses bean-managed transaction demarcations, the acknowledgment can't be part of the transaction.

Last, there are several restrictions on a message-driven bean's context. The container sets a MessageDrivenContext into a message-driven bean to provide the bean access to its contextual information. MessageDrivenContext is derived from EJBContext, yet many of its methods are not accessible to you because they don't make sense in the context of an asynchronous process. The container will throw an IllegalStateException if you try to call these methods, and they're deemed inaccessible by the EJB 2.0 specification.

While the API is present, you can never call getCallerPrincipal and isCallerInRole because the client security context isn't passed along with the call to the JMS Destination represented by the message-driven bean.

Similar to entity and session beans, a context's getRollbackOnly and setRollbackOnly are methods accessible only in container-managed, message-driven beans and only within onMessage. Other methods in the bean have no transaction context and will result in a thrown exception. On the other hand, the method getUserTransaction can be called only from within a bean-managed bean, a restriction also imposed on session and entity beans.

Finally, you may not call getEJBHome on the MessageDrivenContext object. Message-driven beans have no EJBHome objects, thus this method has no meaning.

Conclusion

Message-driven beans fill a void in the EJB architecture, which previously defined a model only for synchronous processing. This new bean provides a component wrapper around JMS's asynchronous messaging. The component model provides ease of development, ease of deployment, and the scalability and portability you've come to expect from the EJB architecture. I foresee many useful applications of the message-driven beans in the future. They'll be used to off-load simple functionality to separate processes and enable the development of asynchronous, mission-critical applications. In this article you saw how easy message-driven beans are to develop. I recommend you take advantage of them when developing asynchronous processing within your EJB application.

Author Bio

Jason Westra is CTO at Verge Technologies Group, Inc., a Java consulting firm specializing in e-business solutions with Enterprise JavaBeans.

jwestra@vergecorp.com

Listing 1: MessageLogBean.java source

```
package jdj.january;

import javax.ejb.CreateException;
import javax.ejb.MessageDrivenBean;
import javax.ejb.MessageDrivenContext;

import javax.jms.Message;
import javax.jms.MessageListener;
import javax.jms.TextMessage;

/**
 * @author jwestra  Verge Technologies Group, Inc. 2000
 */
public class MessageLogBean implements MessageDrivenBean, Mes-
sageListener {

private MessageDrivenContext m_context;

/**
* Public, no argument constructor
*/
public MessageLogBean() {}

/**
 * ejbActivate is required by the EJB Specification
 */
public void ejbActivate() { }

/**
 * ejbRemove is required by the EJB Specification
 */
public void ejbRemove() {
  m_context = null;
  }

  /**
   * ejbPassivate is required by the EJB Specification
   */
  public void ejbPassivate() { }

  /**
   * Sets the MessageDrivenContext.
   *
   * @param ctx               MessageDrivenContext Context
for session
   */
  public void setMessageDrivenContext(MessageDrivenContext
ctx) {
    m_context = ctx;
  }

  /**
   * ejbCreate() with no arguments is required by the EJB 2.0
specification
   */
  public void ejbCreate () throws CreateException {}

  /**
   * Implementation of MessageListener.
   *
   * onMessage logs to sysout.  Using an asynchronous, mes-
sage-driven bean
   * to log eliminates the bottleneck of a synchronized log
manager in
   * your application and enables distributed logging from
remote clients.
   */
  public void onMessage(Message msg) {
    // everything is within a try block to ensure no excep-
tions are thrown
    // from within this method
    try {
      TextMessage tm = (TextMessage) msg;
      String text = tm.getText();
      System.out.println("MessageLogBean : " + text);
    }
    catch(Exception ex) {
      // catch all exceptions
      ex.printStackTrace();
    }
  }
}
```

Listing 2: Excerpt from ejb-jar.xml deployment descriptor

```
<message-driven>
      <ejb-name>MessageLogBean</ejb-name>
      <ejb-class>jdj.january.MessageLogBean</ejb-class>
      <transaction-type>Container</transaction-type>

      <message-driven-destination>
        <jms-destination-type>javax.jms.Topic</jms-destina-
tion-type>
      </message-driven-destination>
</message-driven>
```

Listing 3: WebLogic Server config.xml entries

```
<JMSServer Name="myJMSServer" Targets="myServer">
    <JMSTopic JNDIName="LogMgrTopic" Name="LogMgrTopic"/>
</JMSServer>
```

Download the Code!

The code listing for this article can also be located at

www.JavaDevelopersJournal.com

Serving Web Pages

Putting our Java database knowledge to work

WRITTEN BY
ROBERT J. BRUNNER

After reading the previous articles in this series, we're now ready to apply our Java database knowledge to real-world applications. Perhaps the simplest example is utilizing JSP to dynamically present data stored in our database over the Internet.

While many commercial systems exist that can facilitate the entire development and deployment process (e.g., JDeveloper and Oracle 8*i*), I'll focus instead on one of the products from the Apache Software Foundation (ASF). The Tomcat server provides the reference implementation for both JSPs and Servlets. Because it's an Open-Source organization, all Apache products are free and include all source code, so you can look under the hood.

In this article, we'll first go over the basics of JSP, including the overall architecture and most commonly used JSP tags. Then we'll look at how to connect JSPs and databases to provide dynamic data presentation.

Just the Facts

Before we proceed any further, it's always a good idea to clarify the fundamentals or, in this case, outline the basics of the JSP, including what it is and how it works. As part of the Enterprise edition of the Java language, JSP is not included with the Standard edition of the Java language (J2SE). As a result, it's in a standard extension package (i.e., its root package is javax, not java), and due to the manner in which it works, it's actually a subpackage of the servlet standard extension package (javax.servlet). Unlike Servlets, which are 100% Java, JSP pages combine HTML and Java (or other languages that conform to the JSP model) into a single server processed file. The Java code is distinguished by being enclosed in specific JSP tags, which look similar to HTML, and can be easily generated by an IDE. As a result, JSP is extremely easy to use and provides a rapid design, development, and deployment cycle.

In fact the JSP file is effectively processed in two stages. When the JSP file is first loaded by the JSP engine, the file is initially processed and the Java source code is compiled into a Java class file, which is generally a Servlet class. This step, formally known as the HTTP translation stage, is where all HTML tags and some of the simpler JSP tags are processed.

The second stage occurs when a user actually makes a request from the JSP page (e.g., a Web browser requests the JSP file), and the JSP engine processes the request by executing the compiled JSP file. This step is formally known as the request processing stage and allows the user to send specific data to the JSP file. The Java code that's executed is known as a *scriptlet* and is dynamically executed with each user request, allowing the content to be different depending on each specific request. This ability can be used to build dynamic Web sites, such as e-commerce, by generating the results of a request from a database.

In the interests of brevity and not insulting your intelligence, I won't cover the HTML aspects of a JSP file, in other words, HTML tags like <HTML> or <H2>. If you want more information about HTML, visit the home page of the World Wide Web, www.w3c.org, where you can find information on all sorts of things related to the Web. I'll go over some of the basics of the JSP tags, which, unlike the HTML tags, are case sensitive.

Since a JSP file has HTML and Java code intermixed, it's important to know how to quickly identify the JSP tags. This is actually straightforward, since JSP tags start with only two different character sequences, either <% or <jsp. In addition, the characters that follow these identifiers indicate the exact nature of the JSP tag, which completely specifies its behavior.

The first tag in a JSP file is generally the page directive, which defines the global attributes of the file, such as the scripting language, and allows the JSP developer to import Java packages. For example, the following page directive indicates that the JSP page uses Java (which is the default, and of course, what we're interested in using!) and imports the java.util package:

```
<%@ page
  language="java"
  import="java.util.*"
%>
```

After this, we can start to include Java code in our file, which can be explicitly done in three different tags: the declaration, the expression, and the scriptlet. The first two tags are processed during the HTTP translation stage, while the third is processed for each user request, although certain optimizations can be applied by your JSP server. This can have important design and performance implications, so take care when using them.

The declaration tags are used for variables and methods used later within the current JSP file. For example, the following tag declares a connection object that we can use throughout the JSP file:

```
<%! Connection connection %>
```

The expression tags are important because they're evaluated, and the result replaces the expression tag in the resulting JSP page. For example, given the following JSP line:

```
Our output =  <%= "Hello World" %>,
```

after the JSP HTTP translation stage, the resulting line will be:

```
Our output = Hello World
```

To really perform some serious dynamic processing, however, you need to utilize a scriptlet, which can do a lot more than either the declaration or expression tags allow. For example, given the previous declaration tag that declared our connection object, we can obtain an actual connection object in the following scriptlet:

```
<% connection =
  DriverManager.getConnection(
  "jdbc:acme://localhost:1234/jdbc",
  "java", "sun") ;
%>
```

An important issue with scriptlets is that they can span multiple lines and even be intermixed with HTML tags, as in the following example:

```
<UL>
<% while(rs.next()){ %>
  <LI> <%= rs.getString(1)  %>  </LI>
<% } %>
</UL>
```

where we iterate through a result set, printing out the first column in each row as an HTML list item from an unordered list. Notice how the scriptlet is split over three separate lines, with HTML tags and an expression JSP tag intermingled. In particular, we have the closing curly brace for our result set iteration on a separate line in its own scriptlet tag.

To improve the overall maintainability of the Java code in a JSP file, you should, of course, liberally comment the programming structure and algorithms. You could do this with an HTML comment known as an output comment in JSP parlance, like this:

```
<!--This will establish a connection
to the Corporate payroll database -->
```

But since HTML comments are sent to the browser where they can be viewed along with the rest of the source code for the HTML page, this may not be what you want. Fortunately, there's another choice for commenting your JSP file, namely hidden comments, which are removed by the JSP processor and therefore not sent to the browser. To convert the preceding output comment to a hidden comment, change the enclosing tags as follows:

```
<%--This will establish a connection
to the Corporate payroll database --
%>
```

Output comments can still be quite useful, however, since placing JSP expression tags inside them can dynamically alter the resulting comment. In this example (albeit a bit contrived), we can dynamically create an HTML comment that indicates which day of the current month the page was generated:

```
<!-- This page was generated on
  <%=Calendar.get(
          Calendar.Day_Of_Month)
  %>
```

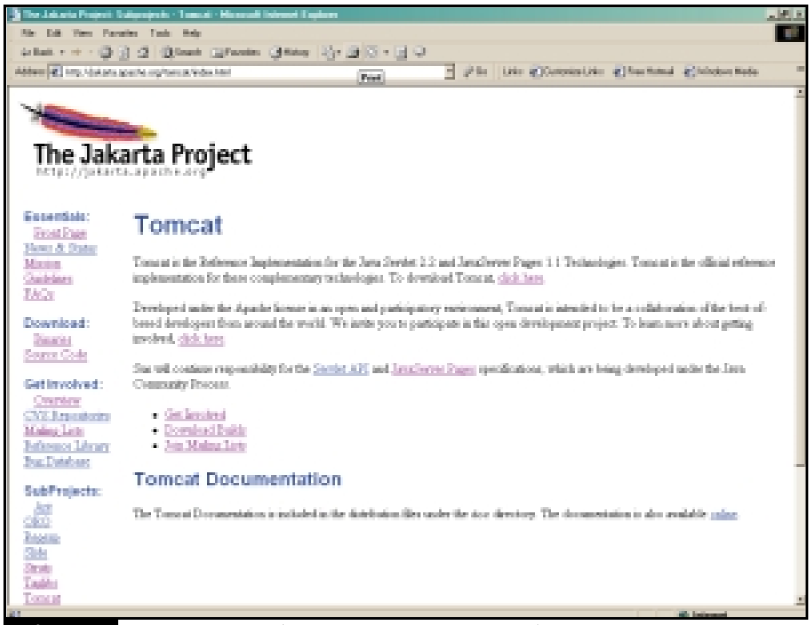
If that's all there was to JSP files, they'd still be quite useful, if not lengthy and somewhat unwieldy to implement. JSP was, however, designed to work within the JavaBeans framework. As a result, you can encapsulate specific concepts inside a JavaBean and access the object using the JavaBeans framework. Furthermore, JSP allows development of custom tags to encapsulate routine operations. As these topics could span several books in their own right, we won't cover them here, but look online at the JSP home page (http://java.sun.com/products/jsp) for more information.

"...Scriptlets... can span multiple lines and even be intermixed with HTML tags..."

Letting the Cat Out

To actually use JSP, you need to run a server process that can convert JSP files into HTML pages that can be rendered on an ordinary browser. This server can be a Web server that's able to serve Java pages, an application server, or even be incorporated into an IDE (e.g., JDeveloper). I strongly recommend, however, that you use the Tomcat server from the Apache Software Foundation. Before discussing how to connect to a database using JSP, we need to get the server and set it up.

The first thing we need to do is download the entire server package. Figure 1 shows the home page for the Jakarta-Tomcat server project (http://jakarta.apache.org/tomcat/index.html), where you can obtain the latest version of the software (currently 3.1, but maybe something different by the time you read this article). The Tomcat server is the official reference implementation for both JavaServer Pages and Servlets. From the Tomcat home page you'll want to click on the "Download Binaries:" link, which takes you to the binary downloads Web page. Click the first Tomcat link under the "Release Builds" heading. This should take you to a directory listing for the current release builds for several Jakarta projects. The only one we're interested in right now is the Jakar-



Tomcat

Tomcat is the Reference Implementation for the Java Servlet 2.2 and JavaServer Pages 1.1 Technologies. Tomcat is the official reference implementation for these complementary technologies. To download Tomcat, click here.

Developed under the Apache license in an open and participatory environment, Tomcat is intended to be a collaboration of the best-of-breed developers from around the world. We invite you to participate in this open development project. To learn more about getting involved, click here.

Sun will continue responsibility for the Servlet API and JavaServer Pages specifications, which are being developed under the Java Community Process.

- Get Involved
- Download Builds
- Join Mailing Lists

Tomcat Documentation

The Tomcat Documentation is included in the distribution files under the doc directory. The documentation is also available online.

FIGURE 1 The home page for the Jakarta–Tomcat project

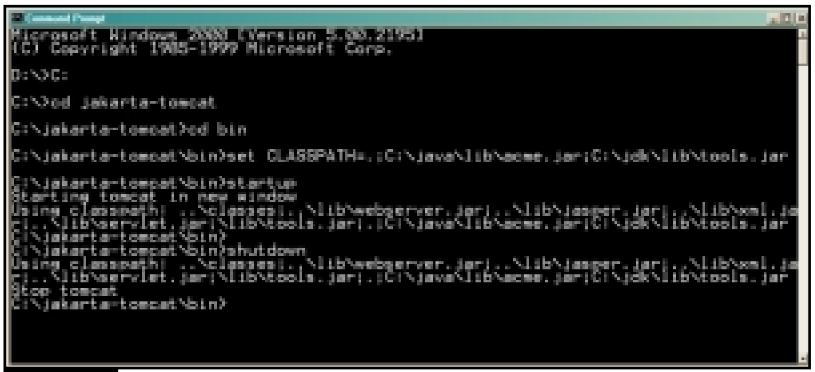
ta-Tomcat project; hence click on the jakarta-tomcat.zip file to download the necessary files.

Once you've downloaded the correct file you'll need to extract the zip file's contents. By default, all files will be created in a new directory named *jakarta-tomcat*. Thus if you extract everything to the C:\ drive, all will be in the directory "C:\jakarta-tomcat\". While you can actually configure the Tomcat server to properly integrate with a variety of different production-quality Web servers, including Apache or IIS, you can also have it run as its own Web server.

The Jakarta-Tomcat server is a Java application and, as a result, runs in a Java Virtual Machine. To start the server, however, we need to run only the startup batch file (or startup.sh on UNIX systems), which sets up all the necessary environmental variables before actually creating a new JVM and running the Tomcat server. Conversely, to shut down the server we need to run only the shutdown batch file (or shutdown.sh on UNIX).

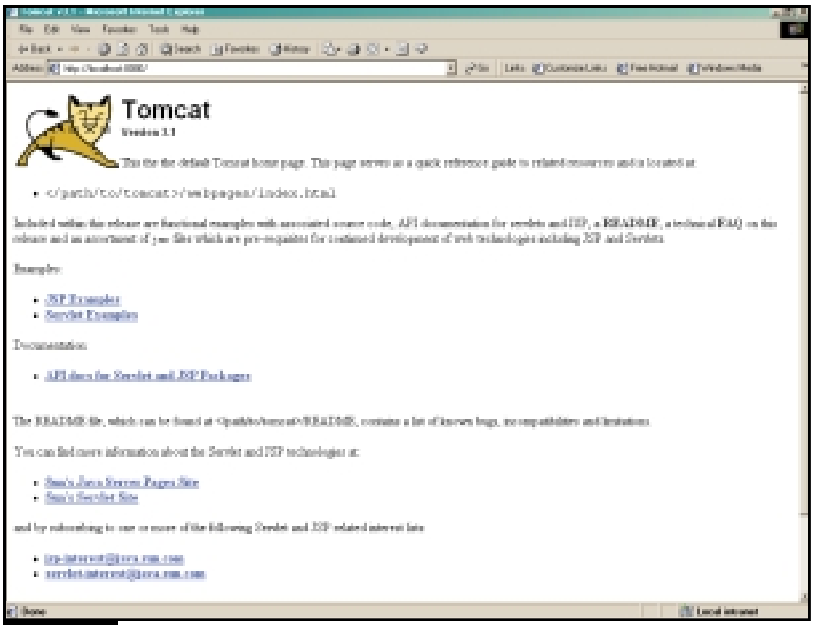
Since JSP files need to be compiled by the server, there's one last caveat before running the startup script; the Java compiler (i.e., the class file that contains javac) needs to be in the classpath environmental variable. This can be done by directly modifying the appropriate script files, by placing the tools.jar file in the jakarta-tomcat\lib directory, or by placing the tools.jar file in your classpath and running the startup script from a command prompt. I prefer the command prompt mode for startup as shown in Figure 2, since I generally have other JAR files that need to be in the classpath of the JVM that's running the Jakarta-Tomcat server (e.g., JDBC driver JAR files). For example, as demonstrated in Figure 2, we first set up the classpath to include both tools.jar (which in this example is in the C:\jdk\lib directory) that contains the javac class file, and also acme.jar (which in this case is in the C:\java\lib directory). Remember that acme is the fictional JDBC driver we've used in previous articles. For completeness sake, the shutdown sequence is also shown.

If the server starts up properly, you'll be able to view the reference information as well as demonstrations of both JavaServer Pages and Servlets. This can be accomplished by browsing to the URL: http://localhost:8080/. The result for version 3.1 of the Tomcat server is shown in Figure 3. By following the links to JSP examples, you'll be able to see JSP demonstrations. If the JSP examples don't work, the tools.jar file is probably not in your classpath variable.



```
Microsoft Windows 2000 [Version 5.00.2195]
(C) Copyright 1985-1999 Microsoft Corp.

D:\>C:

C:\>cd jakarta-tomcat

C:\jakarta-tomcat>cd bin

C:\jakarta-tomcat\bin>set CLASSPATH=.;C:\java\lib\acme.jar;C:\jdk\lib\tools.jar

C:\jakarta-tomcat\bin>startup
Starting tomcat in new window
Using classpath: ..\classes;..\lib\webserver.jar;..\lib\jasper.jar;..\lib\xml.ja
r;..\lib\servlet.jar;\lib\tools.jar;.;C:\java\lib\acme.jar;C:\jdk\lib\tools.jar
C:\jakarta-tomcat\bin>
C:\jakarta-tomcat\bin>shutdown
Using classpath: ..\classes;..\lib\webserver.jar;..\lib\jasper.jar;..\lib\xml.ja
r;..\lib\servlet.jar;\lib\tools.jar;.;C:\java\lib\acme.jar;C:\jdk\lib\tools.jar
Stop tomcat
C:\jakarta-tomcat\bin>
```

FIGURE 2 A demonstration of starting and stopping the Tomcat server in Windows 2000 from the command prompt window



Tomcat

Version 3.1

This is the default Tomcat home page. This page serves as a quick reference guide to related resources and is located at:

- </path/to/tomcat>/webpages/index.html

Included within this release are functional examples with associated source code, API documentation for servlets and JSP, a README, a technical FAQ on this release and an assortment of jar files which are pre-requisites for continued development of web technologies including JSP and Servlets.

Examples:

- JSP Examples
- Servlet Examples

Documentation:

- API docs for Servlet and JSP Packages

The README file, which can be found at <path/to/tomcat>/README, contains a list of known bugs, incompatibilities and limitations.

You can find more information about the Servlet and JSP technologies at:

- Sun's Java Server Pages Site
- Sun's Servlet Site

and by subscribing to one or more of the following Servlet and JSP related interest lists:

- jsp-interest@java.sun.com
- servlet-interest@java.sun.com

FIGURE 3 The starting Web page for the Tomcat server version 3.1

Hooking into a Database

Now we can finally get down to the business at hand, namely dynamically extracting data from a database into our JSP file. Assuming we have already created and populated a database with the following schema:

```
CREATE TABLE Contacts (
First VARCHAR(20) NOT NULL,
Middle VARCHAR(20) NOT NULL,
Last VARCHAR(20) NOT NULL,
Phone INT,
Age INT
) ;
```

we now want to build a Web page that displays all the contacts in our database as an HTML table.

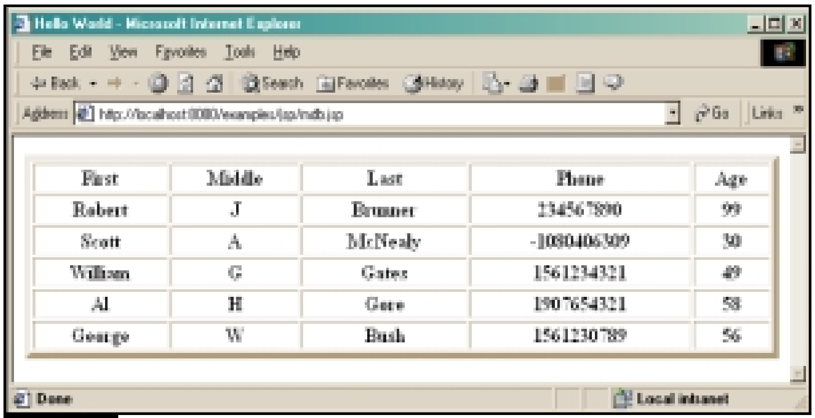
The first step is to write out our page directive:

```
<%@ page
  errorPage="error.jsp"
  language="java"
  import="java.sql.*"
%>
```

This directive first specifies that all errors will be handled by a special JSP file called error.jsp (more on this later). The rest of the directive indicates that we'll be using Java (big surprise) and that we want to import the java.sql package. After this we have some variable declarations that go in their own declaration directive:

```
<%!
  Connection connection ;
  Statement statement ;
  ResultSet rs ;
  ResultSetMetaData rsmd ;
%>
```

We can now include our normal HTML tags that start our HTML files (you may have something more fancy that relies on cascading stylesheets):


Hello World - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites History
Address http://localhost:8080/examples/jsp/mdb.jsp
Go Links
Done
Local intranet

First	Middle	Last	Phone	Age
Robert	J	Brunner	1234567890	99
Scott	A	McNealy	-1080406309	30
William	G	Gates	1561234321	49
Al	H	Gore	1907654321	58
George	W	Bush	1561230789	56

FIGURE 4 The resulting Web page for the mdb.jsp file. The data is entirely fictional.

```
<HTML>
<HEAD>
<TITLE>
ACME Database JSP Demonstration
</TITLE>
</HEAD>
<BODY>
```

Now we need to do some of the heavy lifting, so we'll place our JDBC code in a scriptlet:

```
<%
  Class.forName(
    "com.acme.sql.JDBCDriver") ;

  connection =
    DriverManager.getConnection(

"jdbc:acme://localhost:1234/jdbc",
      "java", "sun") ;

  statement =
    connection.createStatement() ;

  rs =
    statement.executeQuery(
      "SELECT * FROM Contacts") ;

  rsmd = rs.getMetaData() ;

%>
```

In this scriptlet we load our acme JDBC driver into the JVM, establish a connection to our database, create a JDBC statement object, execute the SQL statement that populates our ResultSet object, and obtain the ResultSet Metadata information. We can now start our HTML table:

```
<TABLE
  WIDTH="100%"
  BORDER="+5"
>
```

and dynamically output the header row for our table, which we get from the Result Set Metadata ColumnLabel method.

```
  <TR>
    <%
      for(int i = 1 ;
        i <= rsmd.getColumnCount() ;
        i++) {
    %>

    <TH>
      <%= rsmd.getColumnLabel(i) %>
    </TH>

    <% } %>

  </TR>
```

We'll now want to iterate over all rows in our ResultSet, converting each row of our ResultSet into a row in our HTML table:

```
  <% while(rs.next()) { %>

    <TR>

      <% for(int i = 1 ;
        i <= rsmd.getColumnCount() ;
        i++) {
      %>

        <TH>
          <%= rs.getString(i) %>
        </TH>

      <% } %>

    </TR>
  <% } %>
```

We can now finish off our HTML tags:

```
</TABLE>
</BODY>
</HTML>
```

The last step is to save all these JSP and HTML tags into a file called mdb.jsp. To have the Tomcat serve this file to our Web browser, we need to put it into an HTML hierarchy. While we could configure Tomcat to automatically look in our development directories, I'll take the easier approach and just copy mdb.jsp to the directory where Tomcat looks for its example JSP files. In our installation scenario, this is the C:\jakarta-tomcat\webapps\examples\jsp directory. Once you've copied our JSP file to the correct directory, restart the Web server and browse to http://localhost:8080/examples/jsp/mdb.jsp. When I modify mdb.jsp to connect to my contacts database, I get the result shown in Figure 4. To make this example work, you'll need to have the contacts data in a database table, modify mdb.jsp to use the appropriate JDBC driver and database, and finally, add your JDBC driver to the classpath before starting up the Tomcat server.

You may recall that at the start of our JSP page, we had the errorPage="error.jsp" line in our page directive. You may also have noticed that none of the usual try...catch blocks were wrapping our database code. If so, kudos, you're very observant. In any event, these two are intimately linked, and one of the things that makes JSP so quick to develop Web applications is that you don't need any exception-handling code in your JSP file. Instead, you designate an error page that handles all error conditions. You can have one error page for all JSP files or separate error pages for every JSP file. The level of detail is up to you. Since the error page is a JSP file, you can do quite a bit inside them, including simple things like changing the color or fonts of the HTML page to automatically contacting administrators with error conditions.

Conclusion

While this example is somewhat contrived, it demonstrates all the relevant concepts needed to connect a JSP page to a database. In a production system a better approach would be to encapsulate the database operations in JavaBeans, which would simplify their reuse and maintainability. Since JSP files can also be connected to Servlets (which can maintain state and are higher performance than JSP), we could also push our query interactions to a Servlet and leave the JSP to handle the user interactions. Clearly there's a lot more to explore here than I can fit into one article. Hopefully you'll now feel confident to play with this example and the other JSP articles in this issue.

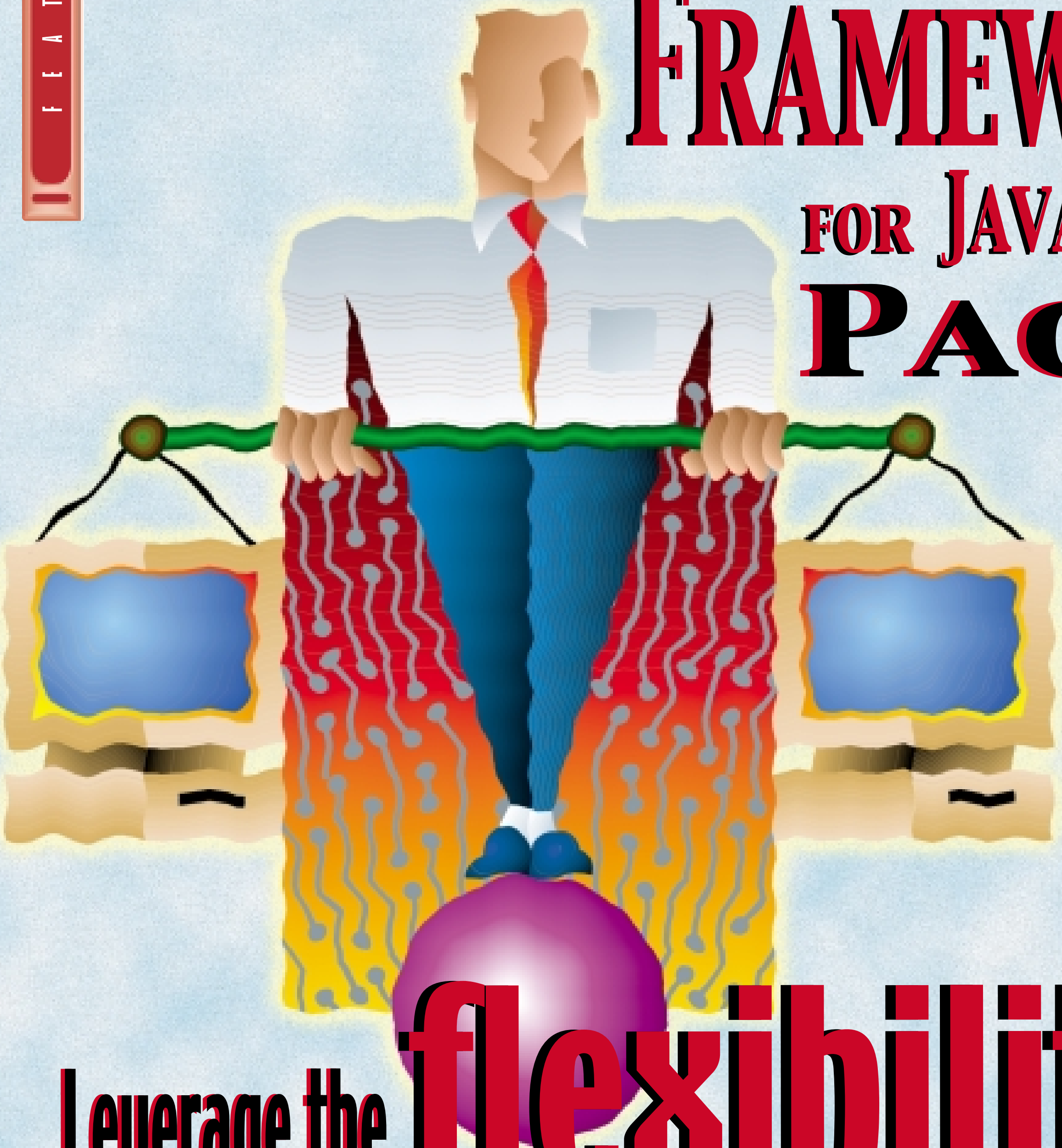
rjbrunner@pacbell.net

Author Bio

Robert Brunner is a member of the research staff at the California Institute of Technology, where he focuses on very large (multiterabyte) databases, particularly on KDD (Knowledge Discovery in Databases) and advanced indexing techniques. He has used Java and databases for more than three years and has been the Java database instructor for the Java Programming Certificate at California State University, Pomona, for the past two years.

FEATURE

A Model View Controller Framework for JavaServer Pages



WRITTEN BY SCOTT GRANT AND
JOSEPH CAMPOLONGO

Leverage the flexibility of JSP

JavaServer Pages is a hot technology right now, as all Java developers are aware. In its simplest explanation, JSP provides the ability to combine Java code with HTML content to achieve dynamic content output from a single source file. Behind the scenes the JSP is compiled into a Java servlet that can be run in any compliant Java servlet engine/container. In essence, a JSP is a way to dynamically create a servlet with a large amount of HTML output and some Java code/logic. So instead of putting a large number of out.println("<HTML CODE HERE>"); statements in a servlet with very little logic, a developer can simply create a JSP page containing standard HTML and a little logic. The JSP compiler will compile the page into a servlet, handling all the ugliness of those out.println() statements.

Given the nature of Web-based applications, this technology is obviously useful, but it comes with some potential costs. Embedding Java code within HTML can lead to maintenance nightmares, with huge files that require both an HTML "Web smith" and a Java developer to maintain them. A JSP framework that could provide a separation of the rather static content of the HTML code from the dynamic content typically provided by the Java code would be a boon.

Another problem with JSP for Web-based front ends and especially for application service providers such as CascadeWorks is the challenge of using a technology like JSP to create a client front end that emulates a lot of the functionality of a standard client application's GUI interface through the Web browser. In addition to needing a separation of HTML content from Java code, it would be nice to emulate the rather event-driven mechanisms and visual components of a GUI framework, such as the one provided by the Swing framework of the Java Foundation Classes Library (JFC), to try to achieve similar functionality for a Web-based interface. However, one of the problems we faced at CascadeWorks is that JSPs are essentially a "page-driven" technology. That created a few problems for us (which we'll address) and led to an interesting solution in creating a JSP framework that would allow pages to be constructed quickly through a template-like process using sets of standard components and events.

Building a Model-View-Controller Framework for JSPs

Most GUI frameworks use a Model-View-Controller (MVC) design pattern. This means that the data (Model) is separated from the visual presentation (View) of that data, and the two are tied together, typically, by a controller of some kind, which can be another class, an underlying set of events, both, and so on. In creating an MVC for a Web-based application using JSP, we must look at what's needed. We investigated many different attempts at such a framework, including the Struts project, which is part of the overall Jakarta/Tomcat Open Source project. Before we discuss our solution, though, a short digression on Struts is called for.

Struts

Struts is a Tomcat subproject whose goal is to "…provide an Open Source framework useful in building Web applications with Java Servlet and JavaServer Pages (JSP) technology. Struts encourages application architectures based on the Model-View-Controller (MVC) design paradigm."

Struts provides a model for seemingly every possible HTML element as a JSP Taglib, which allows them to easily access and be accessed by the server code. Struts then ties every request to an action object through an ActionMapping. So when a request comes in to the server the perform() method on a specific action instance will be called. Struts also provides JSP Taglibs that perform logic that might be needed by the display, such as iterators and comparators. These Taglibs and the supporting framework solve the problem of having logic and therefore JSP scriptlets in the HTML page.

We faced another problem at CascadeWorks, though, that created a need beyond what a page-centric component model like this would support. We wanted to do cleanup for an included page after it was rendered but before the next included page was rendered. For this we needed a framework that would render an included page, then call an event for that page. This meant we needed to build up more full-fledged application functionality in a JSP framework, and for that we really need a full-fledged event model that breaks the page-centric model of JSP, something that's currently lacking in Struts.

Our JSP Framework Wish List

When looking at an MVC design for our JSP pages, there are a few things we'd like to have:

1. The ability to support a container model so that one JSP page can be included within another with no modifications of either being necessary. This would help us create templates for including pages within pages, so that a page could contain headers, footers, menu bar columns, and so on, yet let us easily include or swap out individual pages in a "main" display area dynamically.
2. An event model, in case we need to do any calculations before or after a page is displayed. In other words, we'd like to be able to fire an event just before a JSP page is displayed and just after. We'd also like to fire events on all the pages that make up one display (our frame in the source code) before we begin displaying the first page. So along with a preJspDisplay event per page, we'd also like to have a preWindowDisplay event, which is invoked for each frame and wraps a set of pages.
3. The ability to support a component model so that commonly used HTML data input elements can be reused easily.

Looking at the first of these wish-list items we see that the JSP specification allows for including one JSP page within another. Let's look at the two ways it allows us to do this.

One way is to use a JSP directive, <%@ include file=… %>. When the JSP compiler comes across this directive it adds the interpreted code of the included file directly in the service() method of the servlet the compiler is creating.

In Tomcat's example for <%@include …%>, the compiler generates the following code to insert foo.jsp inside include.jsp (removing the generated comments):

```
out.write("<!--\r\n  Copyright (c) 1999
The Apache Software Foundation. All rights
\r\n  reserved.
\r\n-->\r\n\r\n<body
bgcolor=\"white\">\r\n<font
color=\"red\">\r\n\r\n");
out.print( System.currentTimeMillis() );
out.write("\r\n");
```

The other way is to use a JSP action, <jsp:include page=… />. When the JSP compiler comes across this directive it adds a pageContext.include() to the servlet the compiler is creating.

From the same example the following code is generated for the <jsp:include …/>:

```
String _jspx_qStr = "";
out.flush();
pageContext.include("foo.jsp" + _jspx_qStr);
```

The problem with both of these solutions is that neither allows for item number 2 in our wish list. We'd like to have our framework support an event model. In our case this means that before and after a page is displayed (written out to the output stream), we'd like to have the framework fire an event. Neither the include directive nor the include action allows us to support the kinds of events we want and have included pages. So to support a container model as well as an event model (of the type specified), we can't use the provided JSP include functionality.

We'll return to a specification for the event mechanism in more detail later. Now let's look at the third item on our wish list – the ability to have components. This can be relatively straightforward if we approach components the same way the Struts project did. Every common HTML element or sets of elements or even logic can be created as Taglibs with a reasonably simple interface into the server code. But what we'd really like is to have our framework fire events on the components as well, initializing them with data when necessary and storing out any of the data they've acquired when necessary (this would involve tying a binding mechanism to back-end storage).

Unfortunately, we were unable to solve this problem without, as was done at CascadeWorks, preregistering all the components and precompiling all the JSP pages. This is a separate and involved discussion and beyond the scope of this article. So we'll simply use the Struts' conception of components – it's a Taglib that's a display-oriented item with its own well-defined set of events that are specified in the JSP 1.1 spec and leave it at that.

Our MVC and EVENT Framework

We're going to put together a framework that uses our own mechanism for including pages and supports a full event mechanism. Components are more of an adjunct to the framework than an integral part of it, so we won't discuss them here beyond repeating that the Struts model is a good one.

Our framework starts by providing support for our container concept and event model to the JSP pages we'll be using. This support comes in the form of "backing" document objects. Every JSP page will have a corresponding document object associated with it and vice versa. The collection of JSP pages (remember, we're going to be including JSP pages within JSP pages) that make up the client's view will be modeled by a frame object, which is simply a container for the JSP page's backing document objects. See Listing 1 for an example of a JSP page in our model. (Code listings can be found on the ***JDJ*** Web site.)

This acts as the outer page for every single one of our displayed pages. It will always be displayed. In its turn it includes a JSP page that acts as a header and one that acts as the main page. Each of these documents may include other documents. Its backing document class is in Listing 2.

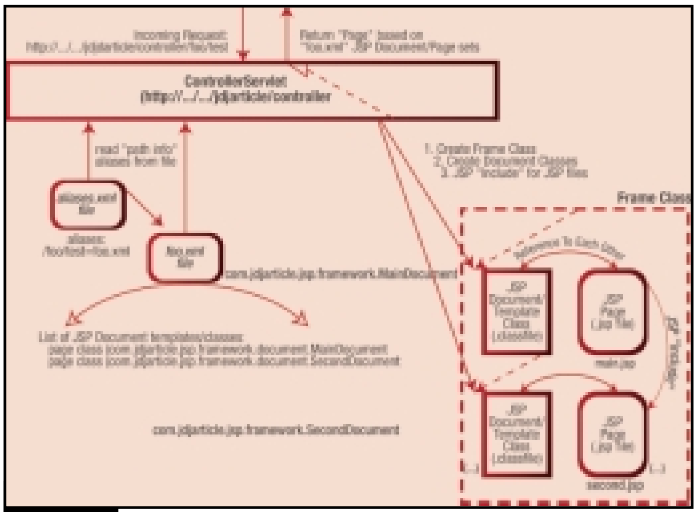
This class defines itself and the page it references, allowing the controller to access the backed JSP document through this class. It also extends the abstract class document, which defines all the events that may be called on the document (see Listing 3).

The controller will fire each of these events on every document with a JSP page to be displayed. Every backing document class also contains a ModelContext object, which allows it to access the ServletContext, HttpRequest, and HttpResponse objects of the current request.

The controller is the linchpin of this setup, driving our event model. It's the single access point of our system, loading every page, and works in conjunction with the frame to display the pages requested (see Figure 1).

The controller is represented by a single controller servlet that performs the following actions:

1. Creates a frame object that represents the HTML page to be displayed based on the incoming URL.
2. Loads up the frame to be displayed.
3. Initiates the framePreProcess() events on the document objects that make up the frame to be displayed.



Incoming Request: http://…/…/jdjarticle/controller/foo/test
Return "Page" based on "foo.xml" JSP Document/Page sets
ControllerServlet (http://…/…/jdjarticle/controller
read "path info" aliases from file
aliases.xml file
aliases: /foo/test=foo.xml
foo.xml file
1. Create Frame Class
2. Create Document Classes
3. JSP "include" for JSP files
Frame Class
Reference To Each Other
com.jdjarticle.jsp.framework.MainDocument
JSP Document/ Template Class (.classfile)
JSP Page (.jsp file)
main.jsp
JSP "include"
List of JSP Document templates/classes:
page class (com.jdjarticle.jsp.framework.document.MainDocument
page class (com.jdjarticle.jsp.framework.document.SecondDocument
com.jdjarticle.jsp.framework.SecondDocument
second.jsp

FIGURE 1: Simple MVC JSP framework overview

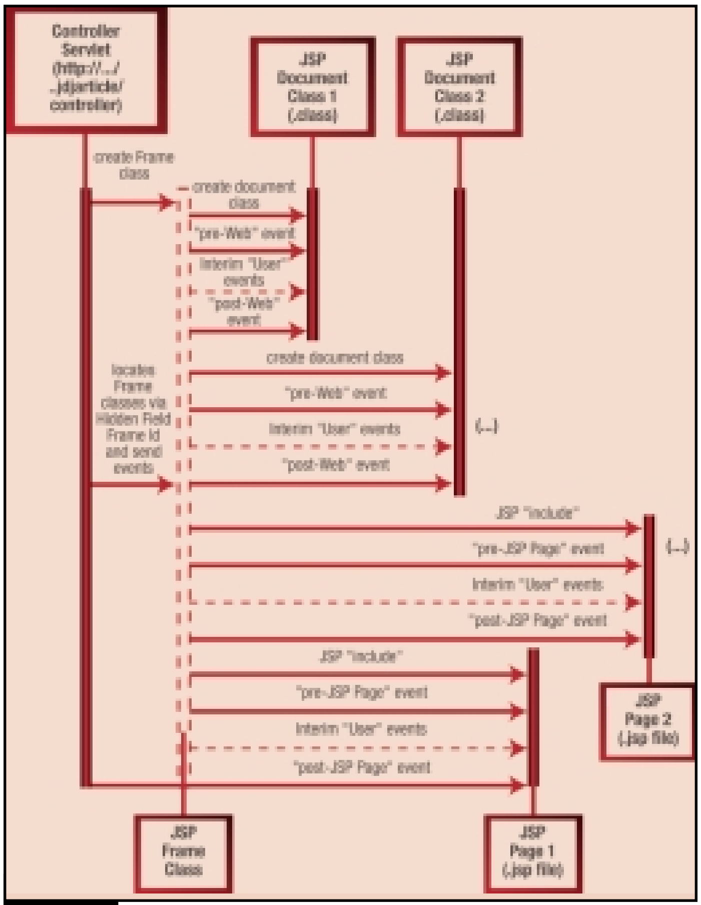


FIGURE 2: JSP framework event sequence diagram

4. Initiates the display of the frame to be displayed.
   a. The frame object that represents the frame to be displayed fires the pagePreProcess() event on each document object about to be displayed.
   b. It gets the document's associated JSP file and displays it.
   c. It fires the pagePostProcess() event on the just displayed Document object.

5. Initiates the validate() events on the document objects that make up the previous displayed frame.
6. Fires any user-initiated events on the document objects that make up the previous displayed frame.
7. Initiates the framePostProcess() events on the document objects that make up the frame that was just displayed.

Note that in number 6 we support a special type of event – a user-defined type. For example, a user can press a button on a page, and the controller will call the defined event for that method on the specified document. Figure 2 shows the sequence of events within the framework.

Two related problems remain unresolved. How does our framework allow pages to be used as a template (i.e., to have other JSP pages included within them), and how do we represent such a page in a URL so that our event mechanism actually works? Any page in our framework that wishes to include another page within it must use a special Taglib, include_document.

The include_document Taglib takes one attribute – location – which is simply a placeholder ID. Whichever JSP page is associated with that placeholder ID is included by a call to RequestDispatcher.include(). The association between the location attribute ID and the JSP page to be included is made for each frame to be displayed in an XML document. For example:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<doc location="MAIN" class="com.jdjarticle.jsp.document.MainDocu-
ment">
    <doc location="HEADER" class="com.jdjarticle.jsp.document.Head-
Document"/>
    <doc location="SUB_MAIN"
class="com.jdjarticle.jsp.document.SubMainDocument">
        <doc location="PAGE_01" class="com.jdjarticle.jsp.docu-
ment.FirstDocument"/>
    </doc>
</doc>
```

This XML document describes the documents each document will contain using the hierarchical nature of XML, and defines the placeholders each document will fill in their respective parent document/JSP pages.

URLs are then mapped to these XML documents in another XML document, called alias.xml.

```
<aliases>
    <alias name="/main">
        main.xml
    </alias>
    <alias name="/second">
        second.xml
    </alias>
</aliases>
```

This means that the URL controller/main will be mapped to the main.xml, which in turn allows the controller to create a frame object to display. Similarly, controller/second will be mapped to second.xml.

Summary

JavaServer Pages is a powerful technology that can provide many benefits to companies requiring dynamic Web front ends with maximum server-side flexibility. It's our hope that this article demonstrates ways in which the flexibility and features of JSP can be leveraged to provide more complex features, such as containers, document templates, and controller-based events, found in a typical GUI-style MVC design.

Authors Bios

Scott Grant is chief architect and lead developer for CascadeWorks, Inc., and a Sun Certified Java developer. He has 15 years of diversified engineering experience..

Joseph Campolongo, a senior developer with CascadeWorks, Inc., is a Sun Certified Java developer.

sgrant@cascadeworks.com or scottg2@home.com • jvc@speakeasy.org.

JDJ Mail

Letters to the Editor

To the Editor:

My company was forced to go to Solaris because serious bugs in the Linux threads library caused multithreaded (hundreds of threads) server-side applications to crash. Glibc is so buggy as to be useless for high-scale, multithreaded Java apps. I haven't seen a single JDK under Linux that won't crash when I run my server code. It works flawlessly under Solaris 8 Intel and also under NT/2000.

a disenchanted Linux lover—**Ron**
roncemer@gte.net

To the Editor:

I love reading ***JDJ***. It's packed with lots of useful information [about] developments in Java-related technologies. However, I'd like to point out some misinformation in the "Eiffel-Like Separate Classes" article in the November 2000 issue [Vol. 5, issue 11]. The author, Francisco Morales, mentions that he used the JavaCC tool (which used to be a tool from Sun Labs but is now supported by www.metamata.com) to implement the preprocessor for his tool. He claims that the JavaCC is based on ANTLR. That is untrue. Most likely JavaCC and ANTLR are descendents of Terence Parr's PCCTS. You can read all about it at www.metamata.com/javacc/javaccstory.html. In fact, JavaCC and ANTLR have taken different approaches to solving the problem of parser generators.

Also there's no reference to JavaCC authors and JavaCC-related literature. I hope the misinformation is corrected.

—**Sandip Chitale**
schitale@selectica.com

In my article I said the following: "JavaCC is based on ANTLR technology; also, it is an LL(k) predicate-based parser generator." This statement is based on a forum celebrated on August 18, 1997, which you can read at http://developer.java.sun.com/developer/community/chat/JavaLive/1997/jl0819.html. The speaker was Sriram Sankar, a software developer and manager at SunTest.

To the question: "How does JavaCC compare with ANTLR?" He answered: "JavaCC is based on ANTLR technology. We built JavaCC since we got quite used to ANTLR (in the C world) and did not want to go back to a LEX/YACC solution with Java."

So I think everything is clear. As a JavaCC user I took this discussion as a starting point for my work, and I think everything that I mentioned was right.

If you get more information on it, please don't hesitate to contact me.

Kind regards
—**Francisco**

To the Editor:

I read Alan Williamson's column, **Industry Watch**, in ***JDJ*** all the time. He's very funny. Good job and keep it going because we appreciate his insight!

Best regards and thank you for your time.

—**Christopher Tava**
christophertava@hotmail.com

To the Editor:

Just comments on your article "Why Linux Lovers Jilt Java" [Vol. 5, issue 12]. You ended with: "The really avant-garde are already poring over Microsoft's C#, which Microsoft fancies is a sort of super-Java blending the best of C++ and Java, and they're keeping their fingers crossed that it won't be a Windows-only creature."

You're right, I think the Linux community has much higher hopes for Java than C#. The "poring over" is only to satisfy an academic curiosity and possibly get some new ideas.

What would be really interesting is if MS did with C# what Sun is doing with Java – make it run anywhere, but then go farther and GPL it. This would effectively have MS using the free software community against Sun. I doubt this will happen though, the technical barriers are huge and MS has even bigger cultural barriers.

—**James Christopher Irrer**
<irrer@eecs.umich.edu>

To the Editor:

In your article on Java serialization "Secrets of Java Serialization" [Vol. 5, issue 11] you stated on page 80:

"You can subclass an object to make it serializable, but this only works if you're the one constructing it. If you need a deep copy of a web of objects, some of which are created inside libraries that you don't have the source for, serialization may be your only alternative."

This seems to imply that your Cloner.clone method works on any object, even those that don't implement the Serializable interface. However, when I tried to use your Cloner class to clone an object that didn't implement the Serializable interface, I got a java.io.NotSerializableException. The Java documentation on this says that ObjectOutputStream.writeObject will only work on objects that implement the Serializable interface….

Am I missing something? It seems your example doesn't work.

—**William Shulman**
williams@babycenter.com

Sorry, we wrote "serializable" but meant "cloneable"! You can "pseudo-clone" things not cloneable using serialization, but you can't, as you point out, serialize things that are not serializable.

We apologize for the slip-up.

Regards,
—**Gene Callahan**
gcallah@erols.com

To the Editor:

Just wanted to send you some kudos on your "Linux Focus" issue. I had time to read only a few of the articles, but it already looks great. I hope it helps other Java developers to embrace Linux. I've been doing most of my Java work on Linux for the past two years, and I wouldn't trade it for anything.

Anyway, no response neccessary, I've just always wanted to send an e-mail to ***JDJ*** for a great issue and a great magazine.

—**Charles Fulton**
fultoncr@ucarb.com

Must Be a Full Moon...

WRITTEN BY
ALAN WILLIAMSON

What a busy time it's been for us all. I'm not talking about the preparation of our titles. No, I'm referring to the fact that we seem to be in a mood for producing babies. First, Ajit Sagar, our resident XML guru, and his wife Seema gave birth to their first child this past summer. Next was Miles Silverman, SYS-CON Media's sales dude, in early November. Now me. Yes, I am now the proud owner of a brand new baby boy. Cormac Robert Williamson logged onto the world on St. Andrew's day (November 30) here in Scotland. Being his father's son, he has been hard at work already and by Day #2 he had his own Web site up and running at www.cormac-williamson.com. That's ma boy!

May They Rest in Peace

As some of you know, I've been running a wee miniseries here chronicling some of the less fortunate dot-com companies and their demise from the IP network. Over the past three months I've listed many of them. Sadly, the list only gets longer. One thing I'd like to point out: we seem to be in good company. *The Wall Street Journal* has published a list of failed dot-coms – the exact same companies I've been highlighting. Coincidence? You decide.

Should we as Java developers be worried? For example, according to the *WSJ*, five companies alone that have filed for bankruptcy account for around 700 people now in the market looking for work. Who knows what skill levels these people have? It would be a bit naive of me to assume that all of them were Java people. A very small percentage, I'd wager. So I think we needn't worry about our jobs just yet.

I've been thinking about why a lot of these companies are going down the tubes, so to speak. You could say to yourself, Why should I care? I don't work for them. Well, that's one way of looking at it, but, especially here at n-ary, when a dot-com failure hits the headlines, family and friends always look to us to ask why, and if it's going to affect us or not. Generally no, but these failures aren't doing the industry as a whole any good. For people that only see black and white, we can easily be tarred with the same brush, and this can be detrimental to everyone.

I sometimes think the media has a big hand in it. It appears to be in love with stories about Internet start-ups going down. The bigger the number lost, the better headlines it makes. In other sectors, if a company is to fail it makes it to the headlines only if more than a hundred jobs are at stake. Even at that, it's generally only the local headlines. It would appear the Internet entrepreneurs make for better reading.

It's well known that the majority of companies fail in their first two years of trading. Only a small minority make it past this milestone. Interestingly enough, I couldn't sleep one night and caught an analyst on the BBC News 24 channel talking about Internet years being something similar to dog years: one year of bricks 'n' mortar is equivalent to five to seven years for an Internet company. If this is true – and I can see some logic in it – these CEOs should be commended for making it past six months, let alone two years. The same report looked at how salaries were starting to come down in this sector as companies struggled with their cash flow every month. Running a company myself, I know only too well the need to keep a steady cash flow. A lot easier said than done at times.

I've been digging around looking at various reasons why these companies fail. Surprisingly, there isn't one obvious common denominator. Many reasons are cited: cash flow, overinflated expectations, high salaries, delivery problems, high cost of technology – the list goes on. But I guess this is business; every case is different.

Granted, I have to hold my hand up and admit some responsibility here. For what am I doing but drawing attention to failures? I'm as bad as the journalists I'm chastizing. So for that I bow my head in shame and slowly scuffle out of the room.

Wireless

What do you think of this wireless revolution we appear to be going through? Are you sitting on the fence waiting for the hype to die down, or are you developing w-applets for your Palms and Nokias? I've been following this world very attentively, and had my son not decided to make an appearance when he did, I would have been at **SYS-CON Media**'s Wireless DevCon Conference in San Jose last month. On the whole, I think it's a great time we're in, as we're just now seeing the move away from our desks. God only knows what my son, Cormac, will be used to when he heads off to university.

Reading various articles on where wireless is making inroads into the office environment, I can't help thinking we're missing something. Yeah, sure, we're networking our devices without the dreaded wires. So what? What difference does that really make? House phones had this a long time ago – the ability to have a base station and have the phone anywhere in the house. But not all is rosy in the world of wireless. Ask my father. He curses about the fact he can't find the phone any more; at least when it was tied to the wall he knew where to go and get it.

To this end, I can understand exactly where people like my father are coming from. So the wireless industry has to start innovating and stop making the devices that are already wireless. We need to see a new wave of devices that work with us and don't become a novelty or hindrance. I for one lose my mobile phone on a regular basis. I then have to ring it and play hide-and-seek to retrieve the bugger. I live in fear of the time I can take everything off my desk and take it with me – I'll never find it.

The biggest pain I find is power. Plugging in a small RJ45 cable is no big deal for me. This is easy. But the power? This is where the work for wireless should go. I'm not a physicist, so I have no idea if the Star Trek technology of having wireless power is possible. Can someone with a physics background come back to me on this? Wouldn't this be the ultimate dream? The ability to draw power from the airwaves, so to speak. This would impact people in every walk of life and have a profound effect on the cabling industry as a whole.

Imagine, if you will, your toaster. Not only would it have an IP address, but it would be a small plastic box that sat anywhere in the kitchen and browned whenever bread was popped in. No ugly power cables draping over the drain board. Hurrah!

That's the wireless world I'm looking forward to.

Way Cool

In December I was introduced to some very sexy Java at work. In fact, I was so excited that I simply had to collar the main man behind it for an interview, which, I'm proud to say, appears in this very issue of ***JDJ***. I'm referring to Steve Rock from EGBS. If you haven't read the interview, go and read it now. Bookmark this line and come back to me.

Read it? Isn't that just wild? Okay. For those of you that didn't bother going to read it, allow me to fill you in. What Steve and his team have built is a full media server that allows the producer, and viewer, to bookmark and tag particular areas within a video and attach either links or notes to them.

For example, say you were watching *The Matrix* with Samuel L. Jackson and you thought the sunglasses he was wearing looked pretty cool. Well, using Steve's technology, you could simply click on the glasses and be taken to a Web site for more information or to purchase them. This stuff is years ahead of its time. Only when we start watching on-demand content will we see this kind of technology come more into our lives.

This sort of thing opens up the world…and not just to the advertising community. You could, for example, click on a particular on-screen character and be fed information pertaining to that character's background and history. The uses for this sort of interactive tagging are endless and very exciting.

But what gives me the extra buzz is that it's all done in Java. This blows my mind. Allow me to tell you why. As a developer, and in this instance a Java developer, it excites me to be in an industry that can benefit and touch so many people's lives. To be able to sit down with your family of an evening and look at how they watch a particular TV program or movie, and then go to work the next day and develop a new interface or improved way of working. We are in an exciting industry and one that touches every other industry. We are so lucky that we can be in a position to explore many of our passions at the same time.

For Steve and his team, working on their Spectrum product allows them to get closer to the world of streamed content. It takes them into fields of producers and directors, and I for one, being a movie buff, wish them all the best.

If you and your team are putting Java to use in a different way, please drop me a line. I'd love to hear from you.

And with that, the score of *Armageddon* is coming to an end, and I had better close off and head on home to see how Cormac is doing.

Author Bio

Alan Williamson is CEO of the first pure Java company in the UK, n-ary (consultancy) Ltd (www.n-ary.com), a Java solutions company specializing in delivering real-world applications with real-world Java. Alan has authored two Java servlet books and contributed to the servlet API.

alan@n-ary.com

Consolidating Legacy Data

Solving legacy data integration problems
Part 1 of 2

WRITTEN BY
BRADY FLOWERS

Every company that's been around longer than a few months has probably created or purchased many different systems dedicated to specific areas of the business. For example, let's say customer files were set up years ago using off-the-shelf software. The software had hooks for customization, and some features were added. Over the years the customer list has grown very large, and the company has become dependent on this system. You know the word: *legacy*.

Of course, the developers who did the customization are long gone.

At some time between then and now new features were needed by the marketing department. The development staff was afraid to make changes in the poorly documented code and, besides, the marketing department's needs had little to do with the accounting department's use of their system. So armed with a C compiler, they added an entirely new system that provided the features they required. But it also ended up duplicating some of the features and data of the existing system because it needed that information and couldn't easily extract it.

Now we can add a new term to our vocabulary: *stovepipe processes*. Each system treats the customer independently, so it's difficult to obtain a complete view of the company's relationships with any given customer. Now, in the age of e-commerce, the company has decided that customers need to see their complete profiles online and you, the Java systems architect and developer, have the task of creating the Web application that will do this. Now it's your problem.

The issue resolves to this: "How do we obtain a single view of the customer across all these systems?"

Architecture for the Solution

The problem can be solved by building an architecture around MQSeries and MQSeries Integrator (MQSI). MQSeries, IBM's messaging and queuing middleware, provides connectivity across multiple systems, allowing different applications on different platforms to communicate with each other in a straightforward and reliable manner.

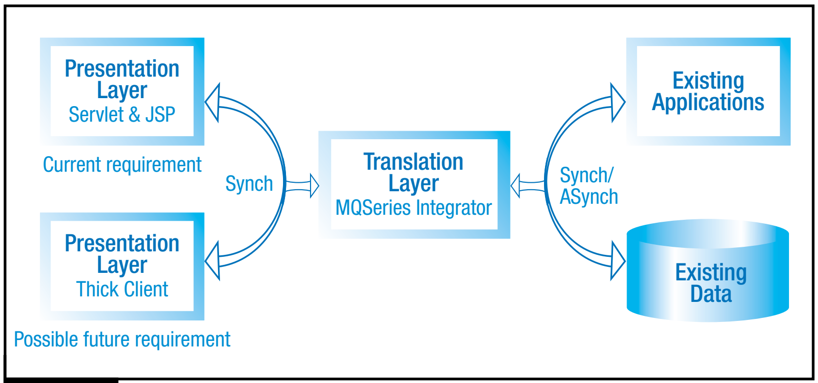


FIGURE 1: Solution architecture

MQSI allows you to isolate client applications that request data from the complexities of where and how legacy applications handle the data.

This architecture allows you to format client requests as required and then route them to the appropriate systems according to a central rules database. At the same time, this architecture is flexible and can be easily extended as new systems or business requirements are introduced.

Strategy

Figure 1 shows the design of our proposed solution. We'll adopt the following strategy:

1. Enable the back-end applications with MQ.
2. Develop MQSI message formats and flows that route and translate data between the legacy data formats and XML – our format of choice for the new Web application.
3. Deploy the servlet and JSP that will handle client requests on the WebSphere Application Server, Advanced Edition.
4. Develop the business logic and front end with VisualAge for Java and WebSphere Studio. This allows us to take advantage of their tight integration with the application server.

For the purposes of this article we'll assume that both back-end systems have already been MQ-enabled by their respective departments. Because the older system could stand only limited modification, a wrapper script was written to capture the binary output stream and place it on a queue. This binary record format can be described using the C language type definition shown in Listing 1.

For the newer legacy system, the group maintaining it managed to modify the code to translate its data structure into XML before placing it on the queue. Listing 2 shows an example record from this application. We'll configure a new message format in MQSI, define the message flows to accept the dual inputs, and merge them into a single output XML stream (see Listing 3).

VisualAge for Java contains connector technology for accessing MQSeries queues. It also provides XML parsers, which we'll need when we add our new business logic. Since we plan to deploy to WebSphere Application Server, our job will be made easier by the WebSphere Test Environment and live debugging engine that are in VisualAge for Java.

Once we've created the MQSeries access code and business logic in VisualAge for Java, we can take the Java code into WebSphere Studio and generate the servlet, HTML, and JSP files necessary to our Web application. We'll also use WebSphere Studio to deploy the application onto our WebSphere test server.

This column recently discussed using VisualAge for Java and WebSphere Studio in conjunction with WebSphere Application Server to create, test, and deploy end-to-end Web application solutions (***JDJ***, Vol. 5, issues 9 and 10). Therefore, we won't focus on that aspect of our project. Instead, in Part 2 of this article we'll look at MQSeries Integrator and some of the steps for creating data translations and message flows. But we'll be able to give you only a taste of the facilities provided by MQSI. For a complete discussion of end-to-end solutions, we urge you to examine the following resources.

Resources

If you'd like to explore the entire end-to-end setup of an MQSeries/MQSI environment for the development and testing of legacy integration applications consult the following sources:

1. IBM's *Patterns for e-business* Web site. The Patterns for e-business Development Kit available here is a self-configuring, end-to-end skeleton Web application. The PDK is a best practice implementation of the User-to-Business pattern: www.ibm.com/software/developer/web/patterns/.
2. Van de Putte, G., Brett, C., Sehorne, P., and Stubblebine, S. (2000). *Business Integration Solutions with MQSeries Integrator.* IBM Redbook: ibm.com/redbooks.
3. Sadtler, C., Ahmed, S., Fleifel, G., Jeynes, M., and Young, Y. (2000). *User-to-Business Patterns Using WebSphere Advanced and MQSI.* IBM Redbook: www.ibm.com/redbooks.

Author Bio

Brady Flowers is a Software IT Architect with IBM's WebSpeed team specializing in WebSphere, Java, and the rest of IBM's suite of e-business applications.

bradenf@us.ibm.com

Listing 1

```
#define CUSTNO_LEN 8
#define FNAME_LEN 24
#define LNAME_LEN 24
#define ADDR_LEN  24
#define CITY_LEN  24
#define STATE_LEN 2
#define ZIP_LEN  10

struct C_CUSTOMER {
 char custno[CUSTNO_LEN];
 char fname[FNAME_LEN];
 char lname[LNAME_LEN];
 char addr[ADDR_LEN];
 char city[CITY_LEN];
 char state[STATE_LEN];
 char zip[ZIP_LEN];
 double balancedue;
 int datedue_month;
 int datedue_day;
 int datedue_year;
}
```

Listing 2

```
<xml_customer>
 <CustomerNumber>123456768</CustomerNumber>
 <NewViewableInfo>...</NewViewableInfo>
 <NewPrivateInfo>...</NewPrivateInfo>
</xml_customer>
```

Listing 3

```
<customer_view>
 <CustomerNumber>123456768</CustomerNumber>
 <FirstName>James</FirstName>
 <LastName>Morrison</LastName>
 <Address>1234 Main St.</Address>
 <City>Anytown</City>
 <State>US</State>
 <PostalCode>12345-6789</PostalCode>
 <PersonalInfo>....</PersonalInfo>
</customer_view>
```

Mastering the JTable

Part 1 of 3

Have total control over your data

WRITTEN BY
BOB HENDRY

JBuilder Data Express controls enable JBuilder developers to use prebuilt objects to provide the user with an interface in which to view and manipulate data. For the most part, the use of Data Express components simplifies our task of programming data access functionality into our applets/applications. One drawback of using these components is that you're restricted to using only functions and changing properties that are supported by that specific control. In other words, although JBuilder simplifies your task, you can use only prewritten functionality.

What if you wanted total control over your data? What if you wanted to control every aspect of how your data is formatted, displayed, edited, and updated? The answer to this is knowing how to use the native Java JTable. Mastering the use of this class is your key to exercising total control over data within your applet/application – albeit with a considerable tradeoff in added complexity.

Over the next few issues I'll cover the use of the JTable and help you master this powerful Java component.

The JTable

Before Data Express there was the JTable, a Java Swing Component with the Swing JComponent as its immediate ancestor. The JTable is used primarily to provide users with a way to view and manipulate data in a columnar (or grid) format. With this functionality, users are allowed to edit and scroll through many records of data rather than editing one record at a time.

Although the JTable can obtain its data from many sources, it's most useful when it holds data retrieved from a database via JDBC. Interestingly, the JTable is the only native Java control that can be populated directly from JDBC that allows the user to manipulate data in a columnar format. As you might guess, the JTable is an extremely complex component – in my opinion, the most complex in the entire Java language. Since books can be written on its use, it's amazing that the JTable is the most underdocumented or written about Java component. Even the best of Java books barely give it a mention. I hope to help change all that.

Understanding the Basics

Like most things in Java, matters appear more difficult than they really are. So it seems with the JTable. Understanding its functionality first requires understanding related classes. As you'll soon learn, with all of its robust functionality the JTable relies on other Java classes to perform most of the work. Often the relationship between these related classes is a source of confusion; understanding it is key to understanding the JTable. Learning it can be daunting, but it's made easier if spoon-fed one piece at a time.

The first class that needs to be understood is the JTable itself. This class contains a two-dimensional view of the data to be displayed. In other words, when the user is looking at the GUI, what he or she sees is the JTable class. Simple uses of the JTable are relatively easy to set up. However, in complex applications use of the JTable without any of its supporting classes is very limited. For example, if you need to automatically populate the JTable with rows and columns from a JDBC data source, the use of a JTable alone will not suffice.

If used alone, the data contained within a JTable needs to be passed as an array of objects or as a vector to the JTable constructor. This may be okay if you want the JTable to contain static values. More than likely you'll want the data to come from a database. In a nutshell the JTable contains the visual portion of the data and generally doesn't control where it comes from or how it behaves (e.g., what happens if it changes). If you have the data ahead of time and want to simply display it to the user (without regard to changes in data) use the simple JTable constructors listed below:

JTable()

This constructs a "basic" JTable that's initialized with a default table model, a default column model, and a default selection model (these classes will be discussed later in this series). This constructor is of limited use because the number of rows and columns haven't been specified. Also, the JTable receives limited functionality because the default models will be used. Since you haven't built custom models (or don't yet know how!), don't expect the default models to do much. This constructor isn't used very often, so I won't offer a code example.

JTable(int numberofRows,int numberofColumns)

This constructs a JTable with numberofRows and numberofColumns of empty cells using the default models. Since no column names can be specified in the constructor, Java will produce generic ones in the form "A", "B", "C", "D", and so on. Ironically, you can add and remove columns after the table has been constructed, but you can't add or remove rows. Like the first constructor, this one doesn't directly accept data.

The code in Listing 1 illustrates the use of this constructor. In the listing, the class constructor builds a JTable (with three rows and two columns) and adds it to a JScrollPane. When a JTable is added to a JScrollPane, the JScrollPane automatically obtains the JTable's header, which displays the column names, and puts it on top of the table. When the user scrolls down, the table's header remains visible at the top of the viewing area. Next, the program's main method places the JScrollPane on a JFrame via a constructor call. Finally, the program uses the setValueAt method to set the value for a single cell in the JTable. Three names and ages are added. Here's the syntax for setValueAt:

```
public void setValueAt(object value,
int row, int column)
```

- ***Value:*** The new value to be placed in the cell
- ***row:*** The row in the JTable to be changed
- ***column:*** The column in the JTable to be changed

Notice that the setValueAt method doesn't care what data type the value is. In the above example I used the same method to set names (strings) and ages (integers). You may think that the setValueAt is an overloaded method. It isn't. It takes the value argument as type Object so the programmer doesn't have to differentiate between data types. Also rows and columns start at the number "0", not "1".

As noted earlier, data can't be included in the constructor JTable- (rows, columns). However, this isn't to prevent you from crafting creative schemes in your data-getting endeavors. The previous example illustrates that the setValueAt method can be used to set the value of a single cell within a JTable. So why can't the data come from a database rather than a hard-coded value? Well, it can. The current constructor doesn't support the use of database data but nothing is preventing you from doing the legwork yourself.

Consider the program in Listing 2. This listing is similar to the previous one, the main difference being that the data within the setValueAt method comes from data in a database. The focus of this series of articles is on the use of JTables and not necessarily JDBC. However, the code does warrant a brief explanation. For the sake of brevity I'll discuss only the differences from the first example.

```
try {

Class.forName("sun.jdbc.odbc.Jdbc
OdbcDriver");
     } catch(java.lang.ClassNot-
FoundException e) {
            System.err.print("Class-
NotFoundException: ");

System.err.println(e.getMessage());
     }
```

This first block of code loads the "Level 1" Java database driver. The Level 1 driver is often called the *JDBC-ODBC Bridge* and is used to connect to local databases that support the ODBC interface.

"Like most things in Java, matters appear more difficult than they really are"

```
String url = "jdbc:odbc:bradygirls";
Ex1Con=
DriverManager.getConnection(url, "",
"");
Ex1Stmt = Ex1Con.createStatement();
Ex1rs = Ex1Stmt.executeQuery( "SELECT
name, age FROM bradygirls ORDER BY
name");
```

Next, the string containing the URL is built; jdbc:odbc: is specified because the JDBC-ODBC bridge is being used. :bradygirls is the name of the ODBC data source. *Note:* For this to work, an ODBC data source named *bradygirls* must be set up on your local machine. This can be achieved via the control panel on Windows machines. In my case, I'm using an MS Access database.

After the database connection has been established a statement class is created, then an executeQuery method is fired. This method takes a string containing a valid SQL statement for an argument. The results of the SQL statement are read into a ResultSet class. Now we're ready to read the contents of the result set and put the data into our JTable.

```
int  li_row = 0;
while (Ex1rs.next()) {
myTable.setValueAt(Ex1rs.get-
String(1),li_row,0);
myTable.setValueAt(Ex1rs.getNum-
ber(2),li_row,1);
li_row ++;
} // while
```

The above while block will loop through the result set one row at a time and populate the JTable. The methods used to extract data from the result set are commonly called the *getXXXX() methods* (with the XXXX being the data type of the column in the result set).

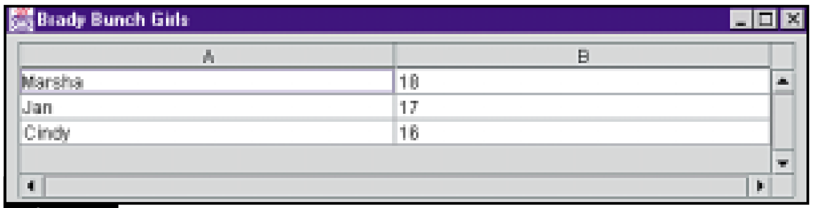
Notice that the first column in a result set is column "1". This is a direct contrast to the first column being "0" in a JTable. This discrepancy can be confusing for a while, but you'll get used to it. Believe me, it's not the only discrepancy you'll find in Java!

Although the above example uses a database to populate the JTable, keep in mind that it's provided as a work-around (instead of using Table Models). Table Models (discussed later) provide a better way to automatically populate a JTable directly from a database. But until you understand Table Models, this code example should keep you pretty busy – especially if you're new to JDBC. The results of the above program are displayed in Figure 1.

```
JTable(Object[][],rowscolData,Object[]
columnNames)
```

Unlike the previous two constructors, this one is used when the rows, columns, and headings are known at the time the JTable is instantiated. Data for the rows and columns are passed in a two-dimensional array of objects. The column headings are passed as a single array of objects. (FYI: There's a similar constructor that uses vectors in place of objects.) The following code snippet uses arrays of objects to instantiate and populate a JTable.

```
Object[] [] data =
{  {"Marsha", new Integer(18)},

     {"Jan", new Integer(17)},

     {"Cindy", new Integer(16)}
};
```



A	B
Marsha	18
Jan	17
Cindy	16

FIGURE 1: JTable results

```
String[] colNames = {"First
Name","Age");
JTable myTable = new JTable(data,col-
Names);
```

All of the simple constructors discussed here are easy to use. However, as I mentioned, these constructors also have a few significant limitations. For example, they automatically make every cell (and row) editable. This is misleading to the user because if you're allowed to edit a value, you're implying that changes can be saved; and remember, the JTable isn't even connected to a database. Furthermore, all of the simple constructors treat all data types as strings. This can be annoying in two ways. First, when aligning data, strings are generally left-aligned and numbers are right-aligned. Since the JTable treats everything as a string, the ages of the Brady Girls (Figure 1) appear to be aligned incorrectly. Second, the JTable has the ability to use other edit styles besides the ones that handle only text. For example, if a value to place into a cell is Boolean, the JTable has the ability to display the data in a check box. After all, a Boolean has only two values, so a check box seems appropriate. However, if you use one of the JTable constructors listed previously, a Boolean column will be displayed as a string. Finally, the largest limitation is that they don't automatically "link" the JTable to a database. Although there are a few limited ways to get around this, one way is to read database columns into an array of objects or vectors, then pass those objects to the JTable constructor. A better way is to implement your own custom table model, which will be next month's focus.

Default Behavior

Whatever constructor you use, be aware of the following default behavior:

- All columns in the JTable begin with equal widths, and the columns automatically fill the entire width of the JTable.
- If the container is resized (made larger), all the cells within the JTable become larger, expanding to fill any extra space.
- When a cell is selected (usually by double-clicking on it), the entire row becomes selected. The cell that was double-clicked becomes highlighted.
- Users can rearrange columns by dragging or dropping them to the left or to the right.
- Columns are resizable by dragging the column header to the left or to the right. This doesn't adjust the size of the JTable itself; the other columns will automatically resize to fill in unused space. Columns can be set to a default size by calling the setPreferredWidth() method for the column model. More on column models next month.

Some Fun with JTables

The program in Table 1 contains additional methods on a JTable that can be used to modify its appearance.

Coming Up Next Issue

As you can see, the JTable controls how the data is presented but has little control over how it's populated. This job is the responsibility of the TableModel, which defines where the data comes from, what the user is allowed to do with it, and what happens if it changes. This model is a Java class you create that extends the Java class AbstractTableModel. This class is fairly complex and can be a bit puzzling. I'll cover it in more detail in the next issue.

Another large piece of the puzzle is the TableModelListener. Its job is to execute when any of the data has changed in the TableModel. A TableModelListener is implemented whenever you create a class extending the Java class TableModelListener, or it can be implemented in an inner class. Of course, the functionality provided by the listener is completely up to you. Many programmers place code to update the database within a TableModelListener.

bobh@envisionsoft.com

METHODS

void	setAutoResizeMode (int mode) Sets the table's auto resize mode when the table is resized.
void	setCellSelectionEnabled (boolean flag) Sets whether this table allows both a column selection and a row selection to exist at the same time.
void	setColumnSelectionAllowed (boolean flag) Sets whether the columns in this model can be selected.
void	setColumnSelectionInterval (int index0, int index1) Selects the columns from index0 to index1 inclusive.
void	setGridColor (Color newColor) Sets the color used to draw grid lines to "color" and redisplays the receiver.
void	setIntercellSpacing (Dimension newSpacing) Sets the width and height between cells to newSpacing and redisplays the receiver.
void	setRowHeight (int newHeight) Sets the height for rows to newRowHeight and invokes tile.
void	setRowMargin (int rowMargin) Sets the amount of empty space between rows.
void	setRowSelectionAllowed (boolean flag) Sets whether the rows in this model can be selected.
void	setRowSelectionInterval (int index0, int index1) Selects the rows from index0 to index1 inclusive.
void	setSelectionBackground (Color selectionBackground) Sets the background color for selected cells.
void	setSelectionForeground (Color selectionForeground) Sets the foreground color for selected cells.
void	setSelectionMode (int selectionMode) Sets the table's selection mode to allow only single selections, a single contiguous interval, or multiple intervals.

TABLE 1 JTable methods

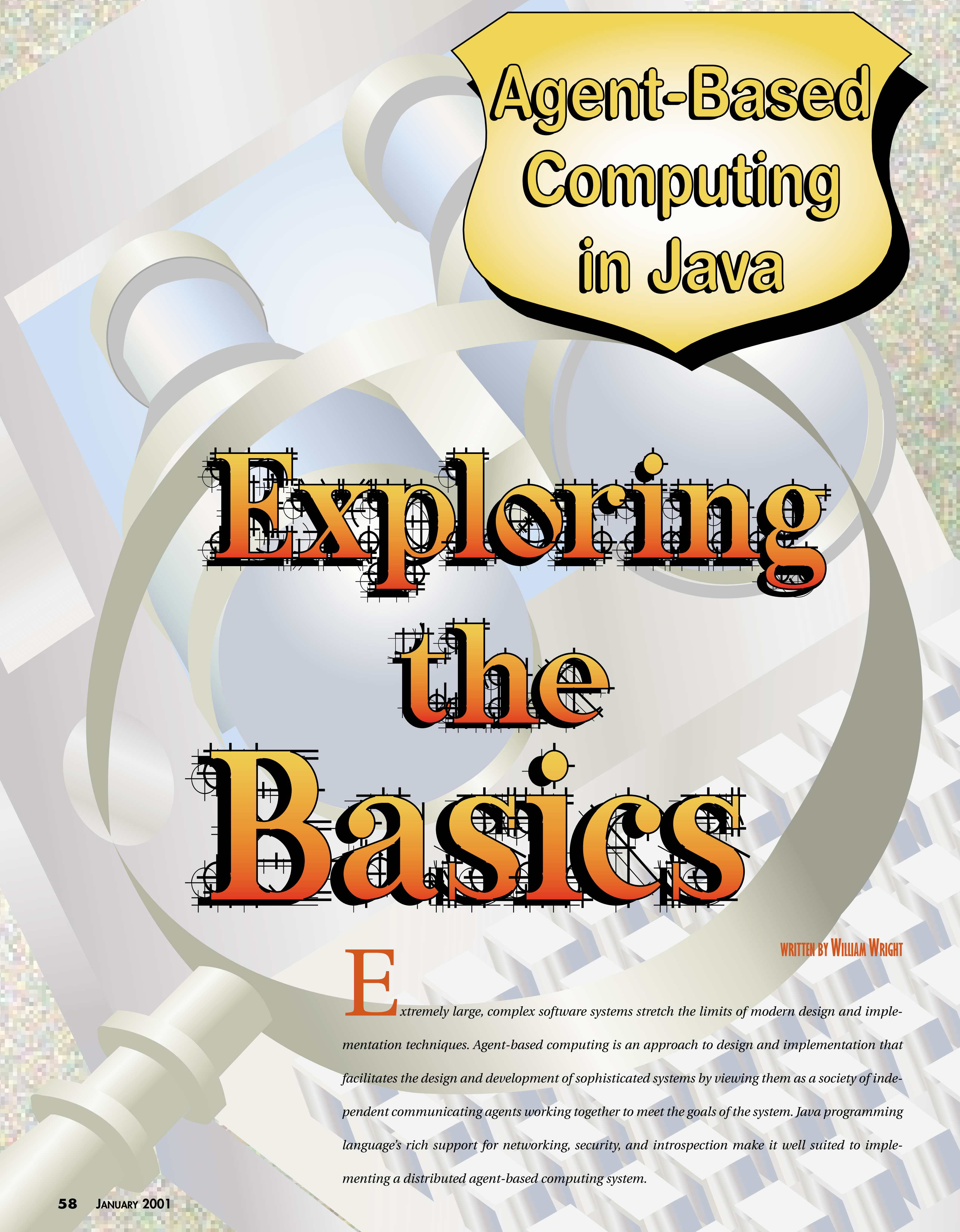
Author Bio
Bob Hendry is a Java instructor at the Illinois Institute of Technology. He is the author of Java as a First Language.

Listing 1

```
import javax.swing.*;
import java.awt.*;

public class BradyGirls extends JPanel{
   static JTable myTable;

BradyGirls(){
   myTable = new JTable(3,2);
   JScrollPane myPane = new JScrollPane(myTable,
   JScrollPane.VERTICAL_SCROLLBAR_ALWAYS,
   JScrollPane.HORIZONTAL_SCROLLBAR_ALWAYS);
   add(myPane);
   myTable.setPreferredScrollableViewportSize(new Dimen-
sion(500, 70));


}

public static void main(String args[]){
   JFrame myFrame = new JFrame("Brady Bunch Girls");
   myFrame.getContentPane().add(new BradyGirls());
   myFrame.setVisible(true);
   myFrame.pack();
   myTable.setValueAt("Marsha",0,0);
   myTable.setValueAt("Jan",1,0);
   myTable.setValueAt("Cindy",2,0);
   myTable.setValueAt(new Integer(18),0,1);
   myTable.setValueAt(new Integer(17),1,1);
   myTable.setValueAt(new Integer(16),2,1);
   }
}
```

Listing 2

```
import javax.swing.*;
import java.awt.*;
import java.sql.*;

public class BradyGirls extends JPanel{
   static Connection Ex1Con;
   static Statement Ex1Stmt;
   static ResultSet Ex1rs;
   static JTable myTable;

BradyGirls(){
   myTable = new JTable(3,2);
   JScrollPane myPane = new JScrollPane(myTable,
   JScrollPane.VERTICAL_SCROLLBAR_ALWAYS,
   JScrollPane.HORIZONTAL_SCROLLBAR_ALWAYS);
   add(myPane);
   myTable.setPreferredScrollableViewportSize(new Dimen-
sion(500, 70));
   }

public static void main(String args[]) throws SQLException{
   JFrame myFrame = new JFrame("Brady Girls Table");
   myFrame.getContentPane().add(new BradyGirls());
   myFrame.setVisible(true);
   myFrame.pack();

   //Initialize and load the JDBC-ODBC driver.
   try {
      Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
   } catch(java.lang.ClassNotFoundException e) {
           System.err.print("ClassNotFoundException: ");
                   System.err.println(e.getMessage());
   }
   String url = "jdbc:odbc:bradygirls";
   Ex1Con= DriverManager.getConnection(url, "", "");
   Ex1Stmt = Ex1Con.createStatement();
   Ex1rs = Ex1Stmt.executeQuery( "SELECT name, age FROM brady-
girls ORDER BY name");
   int  li_row = 0;
   while (Ex1rs.next()) {
      myTable.setValueAt(Ex1rs.getString(1),li_row,0);
         myTable.setValueAt(Ex1rs.getString(2),li_row,1);
         li_row ++;
      } // while
   }
}
```



Agent-Based Computing in Java

Exploring the Basics

WRITTEN BY WILLIAM WRIGHT

Extremely large, complex software systems stretch the limits of modern design and implementation techniques. Agent-based computing is an approach to design and implementation that facilitates the design and development of sophisticated systems by viewing them as a society of independent communicating agents working together to meet the goals of the system. Java programming language's rich support for networking, security, and introspection make it well suited to implementing a distributed agent-based computing system.

This article explores the basics of agent-based computing and examines an Open Source Java toolkit for building distributed agent societies.

Often one of the most contentious parts of agent-based computing is agreeing on the definition of an "agent". In the human world the concept of an agent is familiar to most people. Travel agents make travel arrangements on behalf of their clients. Real estate agents manage real estate transactions as representatives of their clients. Human agents generally possess some specialized expertise and use it as a representative of some client. Software agents are similar in that they usually contain specialized behavior and represent some entity as they carry out their actions.

Most people agree that software agents exhibit some behaviors that distinguish them from other types of software. In general, software agents are:

- ***Autonomous:*** They're capable of carrying out actions on their own; that is, there isn't necessarily a predictable response for every stimulus, and the agent can take action without an external stimulus.
- ***Goal-directed:*** Like their human counterparts, they have skills they utilize to accomplish their objectives. Each agent in a society of agents has one or more goals and strategies to achieve them.
- ***Sociable:*** They can communicate and negotiate with other agents in pursuit of their goals, so in a society of cooperating agents, one may be able to draw on the resources of others to help accomplish its goals. It can also serve as a resource to others as long as that service is in line with the agent's goals.

Agent-based computing is different from object-oriented computing in several ways. While objects and agents both encapsulate their state, objects have methods that will be executed if invoked by some other object. The object can't decide to offer that service to some objects but not others or to offer it at some times but not others. An agent can decide whether carrying out a request is in line with its goals and choose whether to perform the operation. This is why messages between agents are usually called something like "requests" rather than "invocation."

Rather than try to make the case for some particular definition of agent-based computing, let's look at some ways it can be put to use.

There are several reasons agent-based computing is particularly suited to the design and implementation of very large systems. The agent concept parallels the way many real-world systems are put together. Businesses, for example, are comprised of departments that act autonomously but cooperatively in pursuit of their goals. A business might be modeled in an agent society with agents representing these departments. The messages between them correspond to communications between departments. Many physical systems are also composed of autonomous interacting parts. An accurate simulation of these systems can be most directly built using agents to represent the physical components. The messages between agents could represent the forces acting on those components.

Decomposition into agents also benefits the software development process. For the same reason that object decomposition reduces the dependencies between software components, agent decomposition decreases the dependencies even further. In an agent-based system, all agents communicate using a common language, and no agent can directly access any state or invoke any method of another agent. This decoupling of components and rigorous definition of the communication language enables the parallel development of agents, decreasing the chance that a defect in one component will damage another. This is particularly valuable when the components of a system are designed and developed by different organizations. Each organization can implement its business logic in its own way, dealing with the other components only in terms of the interagent messages.

Highly complex systems also lend themselves to agent-based solutions. Systems that are effectively modeled as agents can be built from components that are themselves simple but, when combined, exhibit complex behavior. The development of a complex agent society can be eased by the gradual increase in the sophistication of the individual agents. Some agents in the society can perform their tasks in a simplified manner, while others are fully functional. This allows a complex society to be built and a preliminary evaluation of the system's effectiveness made before all of the components are completed.

Because the agents in a society are autonomously pursuing their own goals, the exact behavior of the society can't always be predicted. This emergent behavior is at the heart of agent-based computing. By working together, software agents can make a system that's greater than the sum of its parts. This unpredictability also makes the task of designing and developing agents more difficult.

An agent needs a strategy to deal with all possible situations, including the failure of a neighboring agent or the unavailability of a resource. Handling failure is one of the most difficult parts of distributed systems programming. Agent-based computing doesn't eliminate this difficulty, but an agent-based approach can be a structure for handling errors. We'll see more about failure handling in the example below.

Software agents are often developed to represent legacy systems that contain essential business logic. Often called "wrappers", these agents make the functions and data of the legacy system available to the society of agents. This is a powerful mechanism for reuse and one of the reasons for the popularity of agent-based systems in organizations with legacy systems that must be modernized but can't be easily reimplemented.

An Example

As an example of agent-based design, let's look at how a small part of a business might be implemented in an agent society. Say a manager (agent) wants to promote an employee named Alice. In Figure 1, the ovals represent agents and the arrows represent interagent messages. Several things have to happen to get Alice her promotion, and several agents carry them out, each using its own goals, expertise, and relationships:

1. The manager requests a promotion for Alice from the personnel department agent.
2. The personnel agent requests a raise for Alice from the payroll department agent. The payroll agent may then have to take several actions to update tax information or recalculate deductions. The payroll agent might be a wrapper for a legacy human resources system.
3. The personnel agent also requests a new office for Alice from the facilities agent. The facilities agent may also need to take several actions to satisfy the request. This agent might order new furniture or schedule painting.

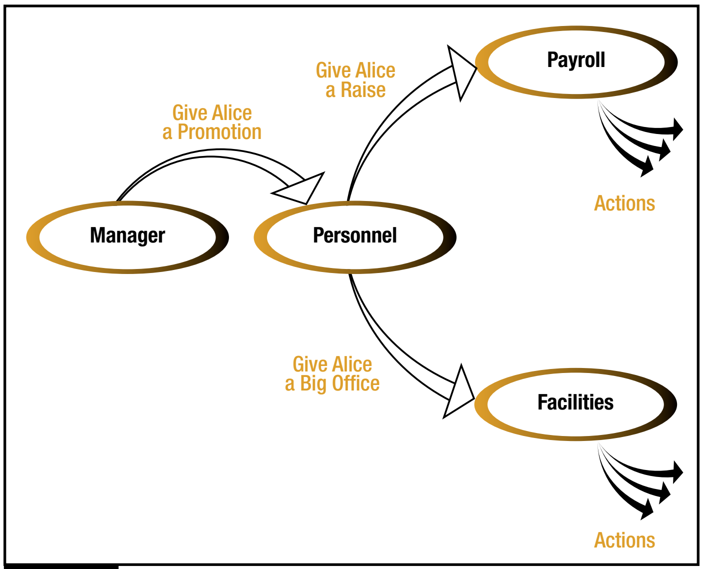


FIGURE 1 Alice gets a promotion

Notice that each agent has its own expertise (algorithm) for satisfying requests. The manager agent doesn't need to know how to talk to the payroll or facilities agents – he or she only needs to communicate with the personnel agent. In fact, the manager agent may not even know that the payroll and facilities agents exist. The scalability and robustness of an agent-based system comes from this information and behavior hiding.

As I mentioned, software agents also need to be able to handle failure. In the example in Figure 1, it's assumed that all requests are satisfied. This is a bad assumption. When agents make requests of one another, they need feedback on the success or failure of the request so they can decide whether to take further action. The communications shown in Figure 1 are really bidirectional communications as shown in Figure 2.

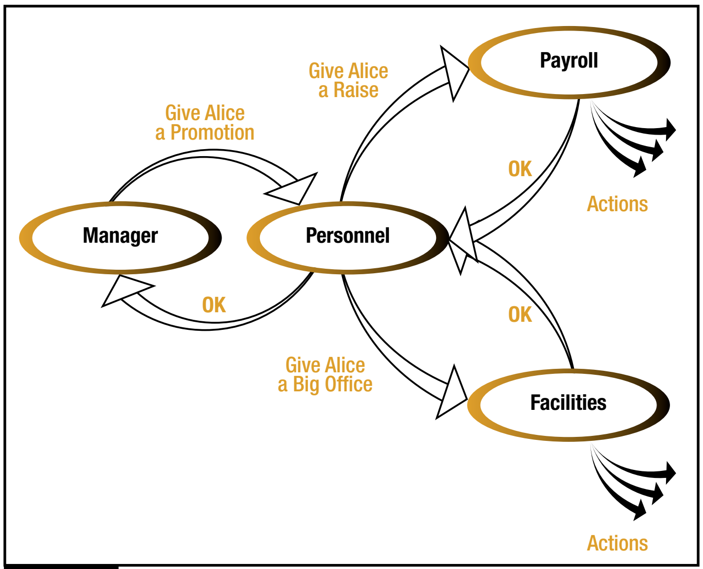


FIGURE 2 Bidirectional agent communication

What happens if a request can't be satisfied? The ability of agents to negotiate becomes important. In this example, if the facilities agent has no more available offices, the personnel agent needs to be notified that the request can't be fulfilled. This report can be a simple success/failure message, but if the agents share a richer vocabulary, the negotiation can be much more efficient. Instead of just saying "No," the facilities agent could say, "There are no offices available until January 1." With this information the personnel agent can make a better decision about how to handle the failure. Notice that the facilities agent contains all the logic for determining available offices. This calculation is invisible to the personnel agent, who only knows that the request wasn't satisfied and any other information the facilities agent chooses to share.

At this point, the personnel agent needs to use its expertise to solve the problem. A simple-minded agent could tell the manager agent that

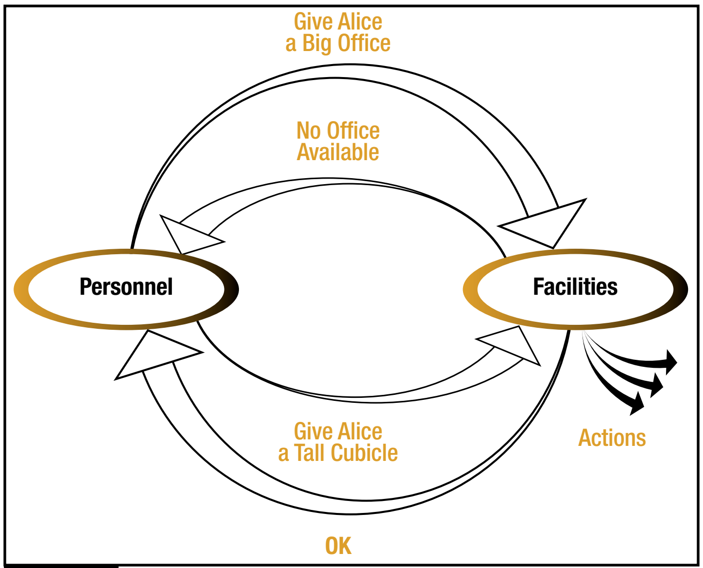


FIGURE 3 Negotiating with the facilities agent

Alice can't be promoted, but that's not a good solution. A better agent might make another request from the facilities agent for a different resource – a tall cubicle, for example – or request an office as of January 1. Figure 3 depicts this negotiation. The order of the messages is top to bottom.

Once the negotiations are complete the personnel agent can report to the manager agent that the promotion was successful using a vocabulary known by the manager agent. Then the manager agent can decide whether to take further action or accept the results given by the personnel agent.

Java Agents

Next let's look in some detail at a Java toolkit for building agent-based systems. The Cognitive Agent Architecture (Cougaar) provides mechanisms for building distributed agent systems in Java.

The Cognitive Agent Architecture grew out of a program by the U.S. Defense Advanced Research Projects Agency (DARPA) called the Advanced Logistics Project (ALP). While originally focused on logistics, Cougaar is a general-purpose agent architecture. The software is available under an Open Source license at www.cougaar.org. Cougaar provides a 100% Java framework for building and distributing societies of agents. It also includes a framework for building the agents themselves and some semantics for the interagent messages.

Cougaar agents communicate with one another by sending directives, which most commonly take the form of tasks; that is, a request for another agent to do something. Of course because they're autonomous, the agent that receives the task may be unwilling or unable to complete it and can report its disposition back to the sending agent. The agent that sent the task can also decide to take back or "rescind" the task. A task can only be rescinded if the receiving agent hasn't already completed the task or can undo what it did.

Cougaar agents are composed of business logic classes called *plug-ins,* and their behavior is determined by the plug-in components, which can respond to and generate directives. Each agent contains a blackboard that's shared by the plug-ins in that agent. Plug-ins can publish objects to the blackboard and subscribe to changes in it. Plug-ins can't communicate with other plug-ins except through the agent's blackboard, which isn't accessible by any other agents or processes. This enforces the modularity of both the plug-ins and the agents. The behavior of most plug-ins is determined completely by what objects in the blackboard they subscribe to and publish.

The reason it's called the Cognitive Agent Architecture is because it was designed to develop agents that model the human problem-solving process. Humans carry out several processes when trying to solve a problem. These include:

- ***Decompose the problem:*** Break the problem down into simpler sub problems and solve those. We saw an example in the personnel agent that knew that to do a promotion, a raise and a new office were needed.
- ***Delegate the problem:*** Assign the problem to someone or something and let them handle it. The manager agent delegated the details of Alice's promotion to the personnel agent.
- ***Consolidate multiple problems:*** Sometimes it makes sense to collect several problems and solve them simultaneously. If the facilities agent had several office moves, it might be possible to group the before hiring a moving company.
- ***Monitor progress:*** Observe the effects of the solution to the problem and reevaluate the solution if necessary.
- ***Gather data:*** Collect information needed to solve the problem.
- ***Report:*** Provide information to others about the solution to the problem.
- ***Act:*** Take action to directly carry out the solution.

Cougaar agents can have plug-ins that carry out these processes:

- ***Expander plug-ins:*** Take large tasks and break them into workflows of subtasks.
- ***Allocator plug-ins:*** Assign tasks to other agents or real-world assets.

- ***Aggregator plug-ins:*** Group tasks to be carried out together.
- ***Assessor plug-ins:*** Monitor external data sources, like sensors or data bases, to make sure that earlier decisions are still valid.
- ***Data plug-ins:*** Retrieve data from databases, Web sites, or other sources, for use in decision making.
- ***User interface plug-ins:*** Report the agent's progress to a human user.
- ***Execution plug-ins:*** Carry out actions that change the world outside the agent society, like updating databases, or commanding actuators.

Figure 4 shows how the personnel agent in this example might be implemented using Cougaar plug-ins. Plug-ins are represented as trapezoids. An expander plug-in receives the "Give Alice a promotion" directive from the manager agent. It decomposes that task into two subtasks: "Give Alice a raise" and "Give Alice a big office." An allocator plug-in delegates the raise subtask to the payroll agent, and another delegates the office move subtask to the facilities agent. This separation of responsibility within the agent makes it easier to upgrade or modify the agent's behavior. Just replace a plug-in with one that implements the new behavior.

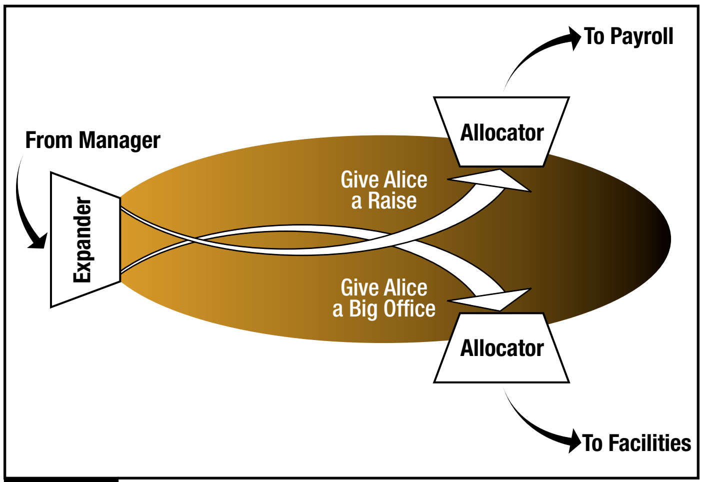


FIGURE 4 A Cougaar personnel agent

Cougaar uses several different Java technologies in its implementation and library of generic reusable plug-ins. It uses an RMI mechanism for interagent communication when agents are distributed across a network. Java introspection is used extensively to facilitate the communication between agents and plug-ins. Many wrapper agents use JDBC to access legacy databases and provide that data to the agent society. Cougaar also includes examples of wrapper agents for Web-based data that access HTML and XML data via HTTP.

If you're like me, things make more sense when you see some code. Let's look at some of the details of the Cougaar software. The basic building block of a Cougaar agent is the plug-in. In its simplest form a plug-in needs only to initialize itself and respond to changes in the set of objects of interest. There are two abstract methods in the plug-in's base class for these operations.

```
protected abstract void setupSubscriptions()
```

As its name implies, this method is implemented by the plug-in developer to perform initialization functions, including subscribing to the objects of interest. It's called just once when the plug-in is loaded. Plug-in developers define subscriptions by defining a small object with one method (a unary predicate) that returns true if the object should be included in the subscription. See Listing 1 for an example of creating a subscription.

```
protected abstract void execute()
```

The Cougaar agent executive calls this plug-in method when one of the plug-ins' subscriptions changes. Events that signal a subscription change include:

- New objects of interest being published
- A change in an object of interest
- The removal of objects of interest

The objects of interest might be published by another plug-in or transmitted from another agent as part of a directive. This is how a Cougaar agent's behavior is defined – through the subscriptions and publications of its plug-ins. The Cougaar infrastructure is also responsible for handling the transaction associated with the execute method. It ensures that no plug-in can modify the subscription objects during the execute method of another plug-in. Listing 1 is a simple example of a Cougaar plug-in.

Agents are social software, so they need to know about other agents. Other agents appear as objects in the local blackboard, so they can be accessed through a subscription like any other object. Tasks delegated from other agents appear in the blackboard, so they can also be elements of a plug-in's subscription. Common plug-in actions include taking incoming tasks and decomposing them into subtasks, collecting tasks to be handled as a group, or assigning tasks to other agents for further handling.

Applications of Agent Computing

Agent-based computing is applied in many domains. It maps well into the realm of autonomous robotics. A fleet of autonomous mobile robots has requirements similar to an agent-based system. The robots are individually goal-driven but need to communicate with one another. They behave autonomously, but cooperatively.

Agent-based computing has also been used to model network security. Work is underway to represent networks and computers as agents and simulate attacks by hacker agents. In this case, the agents are not all cooperative, but they all have goals and behave autonomously.

A marketplace is well represented as an agent society. In this case, agents can represent buyers and sellers. Some examples of this type of agent are in use already in some Internet auction sites. The real power of agent technology will come when suppliers' and customers' negotiation agents are tied into other back office agents and can adapt in real time to changes in the business environment.

Distributed agent computing is a great match for the Java environment. As the technologies mature, agent-based computing will be an important design method for large, complex systems.

Resources for Java Agent Computing

1. www.cougaar.org: Web site for the Cognitive Agent Architecture discussed in this article. In addition to the core Cougaar software, the site has documentation and examples.
2. java.stanford.edu: Web site for the Java Agent Template, Lite. JATLite was developed by the Center for Design Research at Stanford University and is available under the GNU Public License. It provides the connection and communications facilities necessary to enable messaging between agents but says nothing about the internal structure of the agent or the semantics of the messages. JATLite uses FTP or SMTP (e-mail) for interagent communication and has facilities for other message transport mechanisms to be implemented.
3. www.aglets.org: IBM recently released their Aglet Java mobile agent software under an Open Source license. An aglet is a mobile agent in that it can move from one host computer to another as it executes. In addition to the tools on the Open Source site, a free aglet software development kit is available from IBM.
4. mole.informatik.uni-stuttgart.de: A group at the University of Stuttgart has another Java mobile agent system called Mole. The software can be downloaded from their site. It's free for noncommercial use.

Author Bio

William Wright is a division engineer with BBN Technologies (a part of Verizon) in Arlington, Virginia. He has 10 years of experience with real-time systems development and object-oriented programming.

wwright@bbn.com

Listing 1: A Cougaar Plug-In

```
 // Cougaar classes are in the ‘alp’ packages
import alp.cluster.IncrementalSubscription;
import alp.util.UnaryPredicate;
import java.util.Enumeration;

public class ExamplePlugIn extends alp.plugin.SimplePlugIn
{
  private IncrementalSubscription
          allStringsSubscription;
  private UnaryPredicate allStringsPredicate =
          new UnaryPredicate() {
    public boolean execute(Object o) {
      return o instanceof String;
    }};

  /**
   * Establish subscription for Strings
   **/
  public void setupSubscriptions() {
    allStringsSubscription =
    (IncrementalSubscription)subscribe(
         allStringsPredicate);
  }

  /**
   * Handle changes to the subscription
   **/
  public void execute() {



 Enumeration e =
      allStringsSubscription.elements();
    while(e.hasMoreElements())
    {
      System.out.println("Got a string: "+
      e.nextElement());
    }
  }
}
```

Deploying a Java Server as an NT Service

Starting the server without a logged-in user

WRITTEN BY
NITIN NANDA

Generally it's desirable to deploy the Java server in such a manner that it automatically starts when the computer does, and stops when the computer shuts down. This could be quickly and easily implemented by writing an NT service that communicates with the Java server.

NT services can start before any user logs on the NT machine. After the user logs on, he or she may pause, stop, or restart the service using the Service Control Panel applet. When the user logs off though, the desktop and all processes assigned to the desktop are closed; however, the existing services will keep running.

Framework

NT services run under a system account so it's not easy to debug them, therefore it's not advisable to deploy the Java server directly as an NT service. A thin layer of the NT service that acts as proxy for the Java server by starting, interrogating, and shutting it down can do this. Moreover, interrogating the Java server to determine its current status and shutting it down involves interprocess communication (IPC).

Creating an NT service in Java that communicates with the Java server through remote method invocation (RMI) enables the service to easily communicate with the Java server. When the service is started it launches the Java server before any user logs on to the system. The service polls the Java server at regular intervals for its status and updates the Service Control Manager (SCM). Then it shuts down the server when the computer system shuts down, or the user can stop the server from the Service Control Panel applet. During this process, the service logs both informational and warning messages to the NT event log. The architectural framework is shown in Figure 1.

Within the framework I'll briefly enumerate the steps required to implement an NT service. This makes the architecture more comprehensible. The steps required to create an NT service, irrespective of the programming language, are:

- Register the service with the SCM
- Initialize the service
- Overload the service callback methods

Step 1: *Registering with the SCM.*

The services are started and stopped through the SCM; however, for this to happen they must be registered with the SCM.

Step 2: *Initializing the service.*

During the initialization process, the service tells the SCM its current state – stopped or running. It also indicates to the SCM what kind of service callback methods it will accept.

Step 3: *Handling the service callback methods.*

When the NT service is properly registered and initialized with the SCM, the

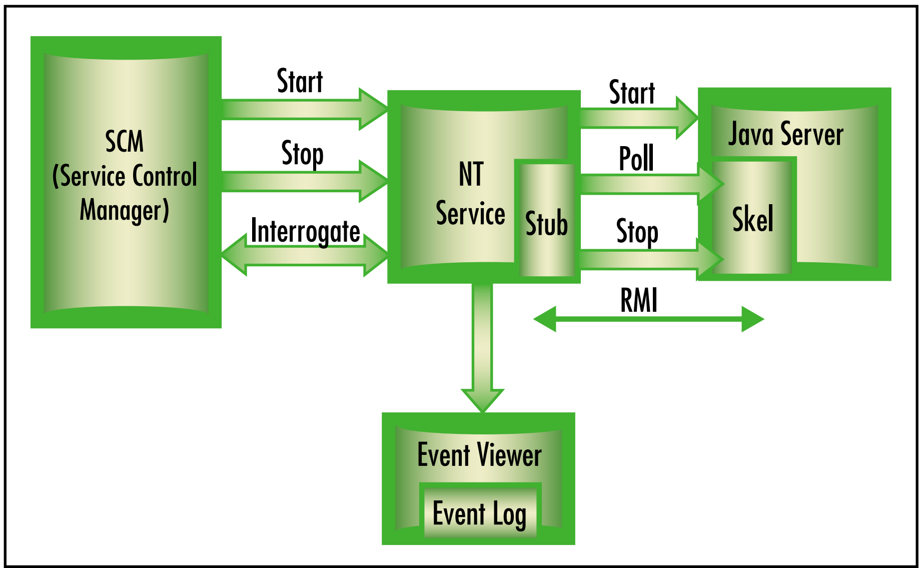


FIGURE 1 The framework

SCM begins to send messages to the service and call its callback methods. These methods handle shut down, stop, and more.

Communicating the Service with Java Server

Now I want to deploy a server called *myServer*. Let's expose an interface, iService, that myServer exposes to the NT service. iService has methods that the NT service calls to stop or poll myServer. Any Java server that's required to be deployed as an NT service implements the iService. The class diagram is shown in Figure 2.

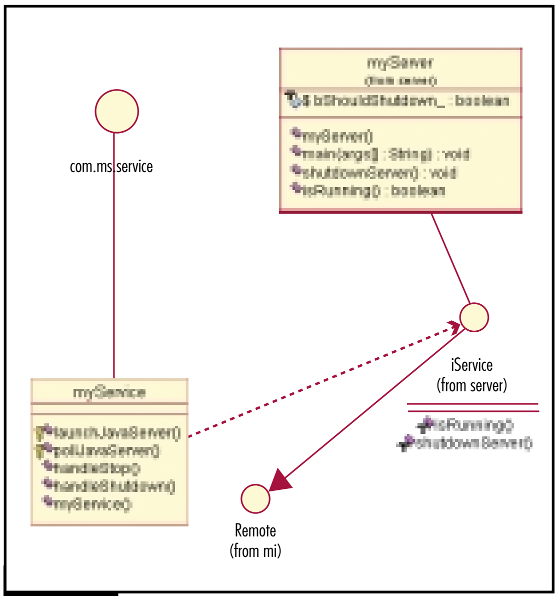


FIGURE 2 Class diagram

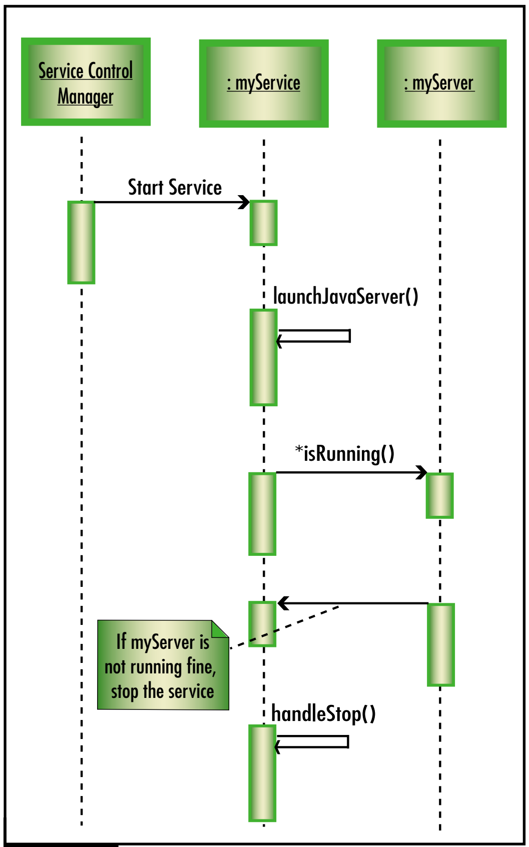


FIGURE 3 Relevant sequence diagram

iService Interface

The iService is a remote interface that has two methods: isRunning and shutdownServer. isRunning determines the current state of the Java server, whether it's running correctly or has crashed. shutdownServer prepares the Java server to shut down. It sets a boolean flag that tells the Java server to release all resources and finally shut down. The interface declaration is given below:

```
public interface iService
    extends java.rmi.Remote
{
 boolean isRunning()
  throws java.rmi.RemoteException;

 void shutdownServer()
     throws java.rmi.RemoteException;
}
```

Sample Java Server

Acting as an RMI server for the NT service, myServer implements iService and UnicastRemoteObject. In its constructor, the server binds itself as "myServer", which the NT service could look up to get its reference. The related code is given below:

```
Naming.rebind("myServer", theServer);
```

Any Java server deployed as an NT service implements the iService interface described above and handles service polling, start, and stop.

Handling Polling

myServer implements the isRunning and shutdown methods. isRunning simply returns true. If the server isn't running well, it throws a RemoteException that the service catches. Based on the return value or the RemoteException, the service updates the status of the Java server in the SCM.

```
public boolean isRunning()
throws java.rmi.RemoteException
{
     return true;
}
```

Handling Stop

To handle the Stop event, the Java server maintains a private boolean variable bShouldShutdown_. In the shutdown method the Java server updates its value to true. The code snippet is provided below:

```
public void shutdownServer()
     throws java.rmi.RemoteException
{
        bShouldShutdown_ = true;
}
```

Handling Shutdown

Within the main function of the Java server a loop regularly checks for the bShouldShutdown_ flag. When the bShouldShutdown_ flags become true, the server releases any resources being used and shuts down the server by calling the following code snippet:

```
while(bShouldShutdown_ == false)
{
      Thread.sleep (2000);
}
/* release server resources if any */
Runtime.getRuntime().exit(0);
```

Java Service

A quick and easy way to create an NT service in Java is to implement the com.ms.service interface that comes with the service.zip package included in the Microsoft SDK for Java. Let's call the service class *myService*. This service internally maintains the remote reference to the Java server it obtained by looking up the RMI server. After creating the service in Java, it can be converted to an executable by using the jntsvc.exe utility that comes with the MS SDK for Java. The executable can be deployed as the service using the myService.exe /install option. Since this service is written in Java, all possible Java features are available to it.

The service can be implemented by following these steps:

- Specify the callback methods
- Start the rmiregistry and Java server
- Poll the Java server
- Handle stop

Java Service Constructor

The constructor tells the SCM what callback functions the service is ready to accept. In our case we can intercept the shutdown and stop callback methods. These methods can be specified to the SCM by using the following code snippet:

```
setRunning(ACCEPT_SHUTDOWN |
ACCEPT_STOP);
```

The next step is to launch the Java server. Since it implements a remote interface and is an RMI server, the rmiregistry also needs to be spawned as a separate process. Spawning the rmiregistry and the Java server as separate processes within the service is done by calling a private method, launchJavaServer. Both these processes can be started using a runtime class. The relevant sequence diagram is shown in Figure 3.

```
prRMIRegistry = Runtime.getRuntime().
exec("rmiregistry");

prJavaServer = Runtime.getRuntime().
exec("java com.jdj.server.myServer");
```

Polling the Java Server

Polling is required to update the current status of the Java server – whether it's running or has stopped – in the Service Control Panel applet. This is accomplished by polling the Java server at regular intervals and retrieving its status over RMI. The polling loop determines the status by calling the isRunning method of the iService interface that the Java server implements. isRunning is called at regular intervals over RMI. This is accomplished by calling pollJavaServer, a private method, within the constructor of the service.

The polling mechanism, in the pollJavaServer method, retrieves the Java server reference by calling the lookup method.

```
String sURL = "rmi://" + "127.0.0.1"
+ ":1099/" + "myServer";
theJavaServer =
(myServer)Naming.lookup(sURL);
```

The polling loop polls the Java server at regular intervals to determine its status:

```
while(bContinuePolling)
{
  try
  {
    boolean bIsRunning =
       theJavaServer.isRunning();
    if (bIsRunning == false)
    {
      throw new Exception("Error
occurred while polling Java
Server.");
    }
    Thread.sleep(45000);
  }
}
```

Shutting Down the Java Server

The Java service gets invoked with handleShutdown or handleStop callback functions when the user goes to the Service Control applet and clicks the "Shutdown" or "Stop" button, respectively, for the service. The handleStop method sets the bContinuePolling to false so that the Java service stops polling the Java server, then shuts it down by calling the remote method shutdownServer. This is followed by closing the rmiregistry and Java processes that were invoked in the constructor of the service. The code snippet of handleStop method is given below:

```
protected boolean handleStop ()
{
  bContinuePolling = false;

  try
  {
    if (theJavaServer != null)
      theJavaServer.shutdownServer();
  }
  prJavaServer.destroy();
  prRMIRegistry.destroy();
  setStopped();
}
```

Writing to the Event Log

Informational or error messages from the service can be written to the event log. To write informational messages use System.out.println; to log error messages use System.err.println. An example of an error message in the code is:

```
System.err.println("Unable to
bind to the Java Server:");
```

An example of outputting an informational message is:

```
System.out.println("Started RMI reg-
istry.");
```

Compilation and Deployment

You can use the source code provided with this article on the ***JDJ*** Web site to deploy the sample Java server as an NT service. I'll briefly touch upon how to compile and register the service with the SCM. In the following example, the source code is present in <source dir> directory.

Compiling the Server

To compile the server, issue the command:

```
<source dir>\javac
com\jdj\server\*.java
```

To generate the stub and skeleton files, issue the command:

```
<source dir>\rmic
com.jdj.server.myServer
```

Compiling the Java Service

Set the classpath so that rmi.zip (can be downloaded from Microsoft's site), services.zip (comes with Microsoft SDK for Java), and myServer_stub.class are present in the classpath. Now issue the following commands:

```
<source dir>\javac myService.java
<source dir>\ jntsvc /svcmain:mySer-
vice /eventsource:myService mySer-
vice.class
```

The command jntsvc generates myService.exe file, which would be deployed in the SCM in the next step.

Deployment

Since the NT service runs under the system account, you need to set up the system classpath and system path. The system classpath must have rmi.zip and myServer_stub.class (for remote invocation of Java server) in the classpath. Also it must contain the directory <source dir> in which the Java server is installed.

The system path should contain the bin directory of the JDK because the NT service spawns rmiregistry.exe and Java.exe programs that are present in the JDK's bin directory. *Note:* Remember to reboot your computer after setting the system classpath and path.

Conclusion

Deploying a Java server as an NT service enables you to start the server without requiring a user to be logged in. The service runs under a system account. A thin layer of the service communicates with the Java server to start it, interrogate its status, and stop it.

This article uses Microsoft SDK for Java to implement the service and rmi.zip to enable RMI communication between the NT service and the Java server. This approach might be compatible only with Java servers that run on JDK 1.1.x. You can directly use the NT service that's provided. With a small modification in the server code (by implementing iService interface) your server can be deployed as an NT service.

If you want to use a non-Microsoft approach to create an NT service it's a somewhat lengthier approach. This would be created in C/C++ and talk to the Java server through JNI. The basic architectural framework described here would be applicable in the latter approach too.

Resources

- *To download MS SDK for Java:* www.microsoft.com/java/download.htm
- *To download rmi.zip:* ftp://ftp.microsoft.com/developr/msdn/unsup-ed/rmi.zip

AUTHOR BIO

Nitin Nanda is an associate project manager in an R & D center of Quark, Inc., based in Chandigarh, India. He's responsible for managing the development of a three-tiered product engineered in RMI.

nitin.nanda@mailcity.com

Deploying and Removing Patches for Java Applications

A cookbook procedure

WRITTEN BY
JOHN R. HINES &
CHRIS L. WHITE

The term *deploy* describes the process of installing the pieces of an application to a host and making whatever modifications are required to the host environment so the application runs correctly without further modifications. A *patch* is a group of Java class files, one or more documentation files, and one or more batch files that installs them to modify the behavior of a deployed program.

Deploying a Java application, particularly a distributed Java application, is a complex task, since applications typically require installing a number of JAR files and batch files (sometimes to different locations) as well as making changes in the host's environment. Most developers create a custom install program to simplify deployment. (Usually the developer employs a deployment program builder like InstallAnywhere to create the program.)

Generating the install program is time consuming, and it must be tested and verified before it and its related application can be shipped to customers. This is a particularly slow process if the install procedure must work on many platforms (Linux, Mac, UNIX, Windows, etc.).

The cost and time required to create and test an install program become significant obstacles to the timely delivery of upgrades and bug fixes. However, minor upgrades and simple bug fixes for a single platform can be quickly implemented with a patch. (*Note:* Patching other platforms usually requires only minor modifications to the patch batch file.)

Patches take advantage of a Java feature called the classpath, a list of locations where the Java launcher (Java.exe) searches for Java class files. The launcher searches the first location, then the second, and so on. If a suitable class is found, the launcher stops searching. If none is found, a "No Class Found" error message is generated. When the application is deployed, if the developer is wise enough to consider the possibility of needing a patch, the first location is empty, the JAR files are deployed to the second, third, and so on, locations. When stand-alone class files for a patch are placed in the first search location, they take precedence over the class files deployed in JAR files in that location and over stand-alone class files and class files deployed in JAR files in all following locations, effectively replacing them.

Classpath Details

The classpath is a string. In UNIX colons separate search locations. In Windows semicolons separate them. A location is either a directory location or a JAR file (including path) location. One special piece of terminology for defining locations: a dot indicates "the directory where the batch file that calls the launcher was started."

A typical classpath in Windows looks like this:

```
.;C:\MyClasses;C:\MyOtherClasses\Some
Jar.jar
```

In other words, search the directory where the batch file that called the launcher was started, then the directory C:\MyClasses, then the jar file C:\MyOtherClasses\SomeJar.jar. *Note:* When searching locations, the launcher searches for a fully qualified class file (package plus a dot plus class name) and uses the package name as a directory hierarchy.

Given the above classpath, if the launcher was searching for someRootPackage.someSubPackage.ThisClass, it would search for the file ThisClass.class in the directory someSubPackage under someRootPackage:

1. Under the directory where the batch file was started
2. If it wasn't there, under the directory C:\MyClasses
3. If it wasn't there, in the file C:\MyOtherClasses\SomeJar.jar
4. If it wasn't there, the "No Class Found" error is generated

Note: If the launcher finds a candidate, the last thing it does is verify that the file found does indeed contain a suitable package statement.

Setting the Classpath

There are three ways to set the classpath. A classpath environment variable can be defined at system start-up or in the batch file that calls the Java launcher. Neither is recommended if multiple Java applications requiring different classpaths are installed. The recommended way to define the classpath is to use the Java Launcher command line option –cp in the batch file that starts the launcher.

The –cp option allows the developer to specify a classpath that's valid for the Java launcher until it terminates. Consider this UNIX command line:

```
java –cp .:A.jar:B.jar com.myPack-
age.MyClass
```

- The launcher searches in the current directory (the ".") for classes.

- If it doesn't find the class, it searches in A.jar.
- If it doesn't find the class in either the current directory or in A.jar, it searches in B.jar.

A class file in B.jar is effectively erased by placing another version of the same class in A.jar, and a class file in either B.jar or A.jar is effectively erased by placing another version in the current directory. A.jar and B.jar can be patched by placing *.class files in the current directory. Removing the patch simply requires removing the *.class files.

Note: If java.exe is started without the –cp command line and if no classpath environment variable is specified, only the default location of the libraries is searched. (The default location is the lib directory under the same directory as the bin directory where java.exe is stored.)

Warning: A classpath variable containing many search paths significantly increases search time!

An Example

The procedure outlined in this example assumes the developer is using Windows NT for a development platform with a ZIP program installed and is deploying to a Sun UNIX platform. However, the general procedure is platform independent.

Suppose you need to patch the JAR file SomeJarFile.jar in the directory/home/myname/someproduct by replacing the file com.somePackage.MyClass.class with the file J:\MyClasses\com\some Package\MyClass.class. Also, suppose the program in the JAR file is started by a batch file in the same directory as the JAR file using the line:

```
java –cp SomeJarFile.jar
com.somePackage.SomeClass
```

A Name

First, assign a meaningful name to the patch, say patch2000082301. (The algorithm "patch" + year + month + date + patch number guarantees that an alphabetical listing of patch names is also a listing from oldest to newest, which simplifies patch management.)

A Directory

Second, create a directory with the same name as the patch, say, J:\patch2000082301, to store the patch and working files associated with it.

A ZIP File

Third, use a ZIP program to create a ZIP file with the same name as the patch, say, patch2000082301.zip.

A Readme File

Fourth, use a text editor to create a "readme" text file with a meaningful name, say, readme2000082301.txt. The file should contain a description of the patch and a list of the files included in it. For example:

```
readme2000082301.txt describes the
patch patch2000082301.
This patch is implemented by unjaring
patch2000082301.zip
into the directory containing the jar
file SomeJarFile.jar

To install this patch, copy
patch2000082301.zip
into the directory containing the jar
file SomeJarFile.jar
and unjar it with the following line:

  jar -xvf patch2000082301.zip

To remove this patch, go to the
directory that contains the jar file
SomeJarFile.jar and run
removePatch2000082301.ksh.

This patch consists of three files:
readme2000082301.txt – a description
of the patch
com.somePackage.MyClass.class – the
"fixed" class
removePatch2000082301.ksh – the patch
remover
```

Add this file to the ZIP file. Don't add any location information for this file!

Map a Drive to the Appropriate Root Directory

Fifth, map a drive, say, K: to J:\MyClasses. (If the J: drive is a UNIX drive, use the "ln" command). If mapping a new drive to the folder is too difficult, copy com and its subdirectories to the root of some drive. Mapping the drive creates the correct path for the class files in the ZIP file.

Add the Class File to the ZIP File

Sixth, use the ZIP program to add K:\com\somePackage\MyClass.class to the ZIP file. Be sure the ZIP file shows the location com\somePackage for the file. (WinZip is one program that does this.)

Add a Remove Batch File

Seventh, using a text editor, write a batch file with a meaningful name that removes the class files, say, removePatch2000082301.ksh. The file needs to contain only the lines:

```
echo batch file to remove patch
patch2000082301
echo must be in the same directory as
the jar file being patched.
rm com/somePackage/MyClass.class
rm readme08023.dat
rm patch2000082301.zip
```

Add this file to the ZIP file.

The batch file must not contain any ^M characters at the end of a line or they'll be treated as part of the file name and the RM command will not work correctly. Since this file is inside a ZIP, the "transfer binary" FTP option won't strip out the ^Ms.

Copy the ZIP File

Eighth, copy the ZIP file to the directory where the JAR file is deployed.

Modify the Start File

Ninth, using VI or some other editor, modify the batch file that starts the program so it reads:

```
java -cp .:SomeJarFile.jar
com.somePackage.SomeClass
```

This change forces the loader to first search for new class files under the directory where the JAR file is deployed before it searches the JAR file. (Hopefully, this change has already been made.)

Unjar the ZIP File

Tenth, at the command line in the directory where the JAR file is deployed, unjar the ZIP file:

```
jar -xvf patch2000082301.zip
```

Restart the Program

Last, the program being patched must be terminated if it's currently running. When restarted by the modified batch file, the new class files should be loaded so testing and verification can begin. If testing isn't successful, you can remove the patch using the batch file removePatch2000082301.ksh. If it's successful, you can claim a great victory and announce that the patch is deployed.

A Possible Problem

Caution: If a previous patch involving MyClass.class is already installed, this procedure will destroy the old patch class file. To avoid this possibility, rename the old MyClass.class file something like MyClass.class.oldpatch before you unzip the new patch file. (Saving this file is a good idea; you may find that the new patch doesn't work correctly, so you'll want the old file available.)

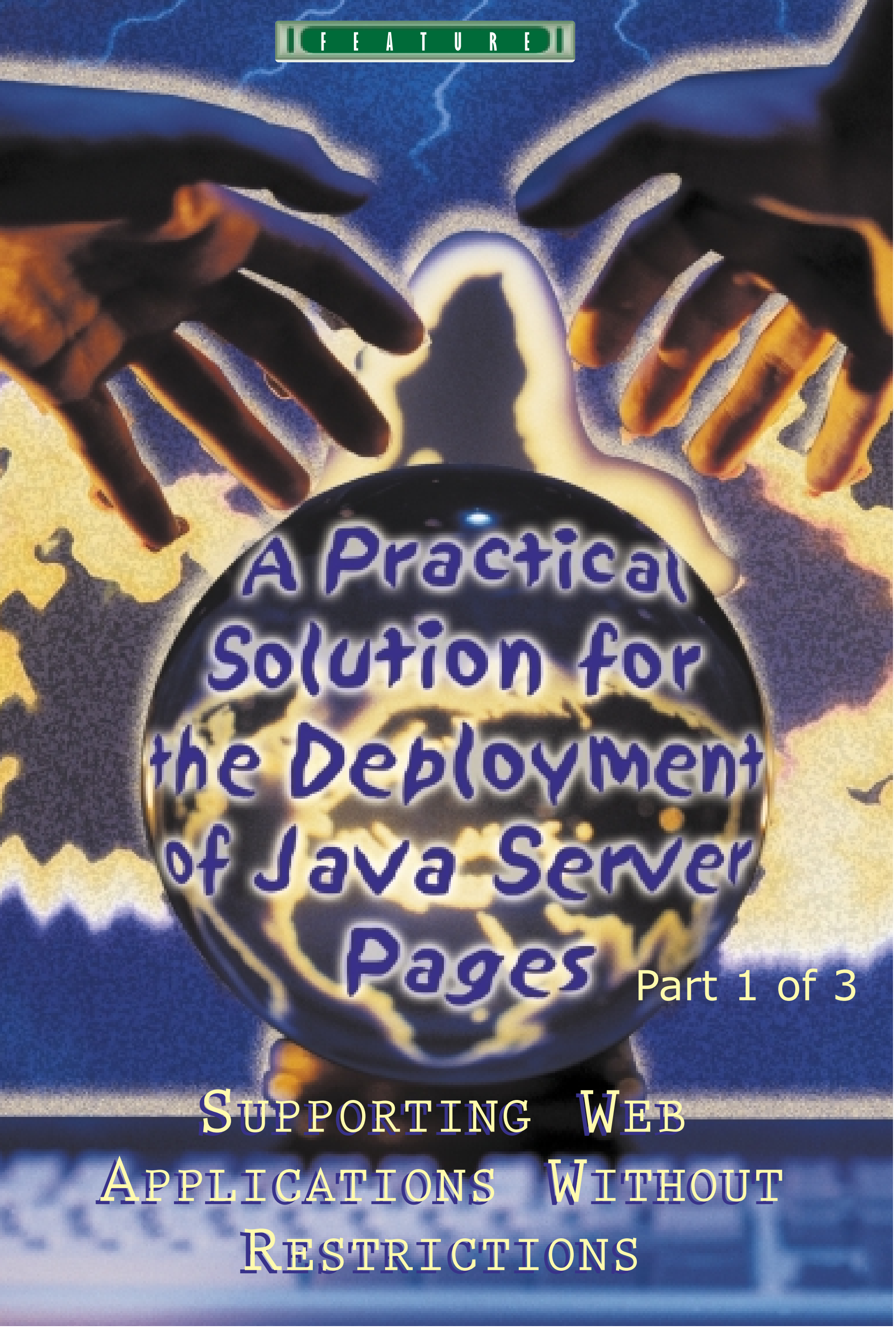
jrhines@softwareconsultingengineer.com

chris.l.white@wcom.com

AUTHOR BIOS

John R. Hines, P.E., is president of Software Consulting Engineer, Inc., in Dallas, Texas. He develops distributed Java applications during the day and teaches Java programming at Richland College at night. He's an enterprise architecture consultant for a large-scale project using Java and CORBA at MCI WorldCom.

Chris White is a software development manager for MCI WorldCom. He has delivered several large-scale projects using Java, CORBA, and XML technology over the past 10 years.



A Practical Solution for the Deployment of Java Server Pages

Part 1 of 3

Supporting Web Applications Without Restrictions

WRITTEN BY ALEXIS GRANDEMANGE

Sometimes it's worthwhile to go back and visit your former projects. It certainly was for me – using presentation as a commodity to be deployed according to network configuration is the concept that resulted from my visit.

The original assignment was to reduce the operating costs of a large banking agency network. How could this be achieved with a network of 23,000 personal computers scattered over 2,000 sites connected by a frame relay with a guaranteed bandwidth of 32Kb? We selected an intranet solution that radically reduced PC client/server applications to a single local program, the browser. To reduce bandwidth needs we deployed one Web server per site to perform only three tasks – handle presentation, maintain reference data, and invoke central system applications. This intranet design saved money by reducing the number of machines to operate from 23,000 to 2,000 by allowing them to be operated with a browser.

We can implement this concept to reduce the load of central farms with Java and J2EE because a fast local loop isn't always available. My goal, however, is to show specifically how we can do it better. The aforementioned approach didn't support a Web server's update on the fly and we had to contend with the central system's synchronization. The solution I present here addresses these issues by allowing Web servers to download their presentations the way browsers download applets. Figure 1 illustrates a possible organization. Let's summarize what we would need:

1. Inexpensive Java servers able to host JSPs and servlets
2. An API allowing them to invoke central system applications
3. A simple way to download a presumably large number of Java servers from any number of central repositories

The first two requirements can be fulfilled with off-the-shelf products, and the local Java server has to address only the following three requirements:

- Generate presentation
- Invoke central applications
- Maintain reference data

At least two Open Source products meet these needs: Tomcat (http://jakarta.apache.org/downloads/binindex.html) and Resin (www.caucho.com/download/index.xtp).

EJBs, Remote Method Invocation (RMI), or Java Message Service (JMS) can be used as the API to connect to central systems.

The last point, presentation downloading, implies development. This requires more explanation and thought and is the core of the article. Presentation downloading relies on a Java class loader and leverages on JSPs and servlets specifications, which I'll present first.

Standard

The Java Servlet Specification v2.2 defines a Web application as a collection of HTML pages, servlets, and classes that exists as a structured hierarchy of directories. The root of this hierarchy is the document root that serves files such as images or HTML. If your Java server waits for HTTP requests on www.iamakishirofan.com and you defined your Web application as gunnm, your users will be able to invoke zalem.html, stored at the root with the URL www.iamakishirofan.com/gunnm/zalem.html.

A WEB-INF directory contains a web.xml file that describes, among miscellaneous things, servlet and JSP definitions, initialization parameters, mapping to URL, and security constraints. It can also contain a

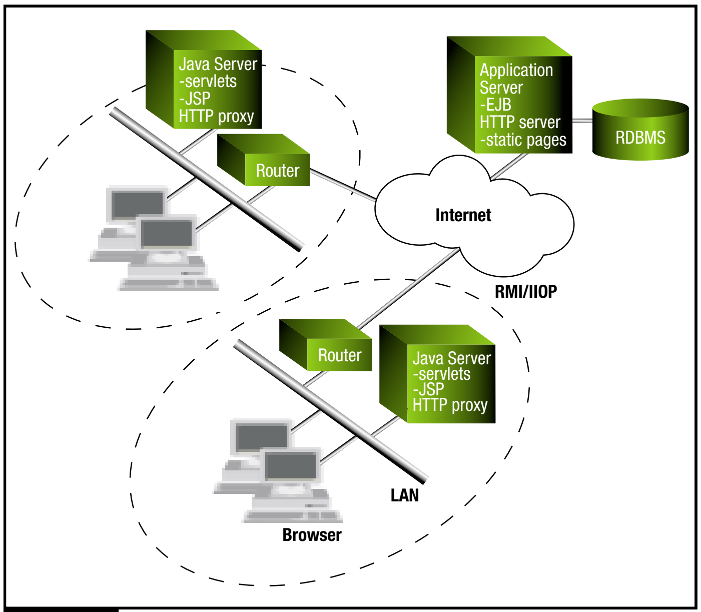


FIGURE 1 Topology

classes subdirectory in which classes, servlets, taglibs, JSP invoked beans, compiled JSP, and more are stored. A Web application should be packaged in a .war file – the JAR archive of the hierarchy.

This packaging is convenient as it gathers all related components in a single delivery. It has another important property: all servlets and JSPs of a .war are served the same ServletContext, which is different from the ServletContext of other packages. Servlets and JSPs can use this ServletContext to access .war data, such as resources and initialization parameters, or to store and retrieve application-wide attributes.

The servlet container loads and instantiates servlets. It initializes them before their first use by calling their init() method with an object that implements the ServletConfig interface. This provides access to servlet-specific data. When it decides to unload a servlet, the container invokes the servlet destroy() method and unreferences it. Each time the container has to route a request to a servlet, it invokes the servlet's service() method.

A compiled JSP is a servlet, even if it doesn't extend HttpServlet or GenericServlet as a normal servlet but as another class that's application server dependent. In the case of Resin, it's com.caucho.jsp.JavaPage and with Tomcat, org.apache.jasper.runtime.HttpJspBase. As you can see, compiled JSPs are no longer portable even if there are only minor differences. The specification requires a JSP to implement a standard HttpJspPage interface. A JSP indirectly handles container requests as depicted in Figure 2.

A compiled JSP implements a _jspService() method and, optionally, a jspInit() and a jspDestroy() method. The specification implies that, for instance, when the container invokes Servlet.init(), jspInit() is invoked somewhere in the implementation of the JSP base class. I've provided the Tomcat implementation in Listing 1. All Java servers I tested have similar code.

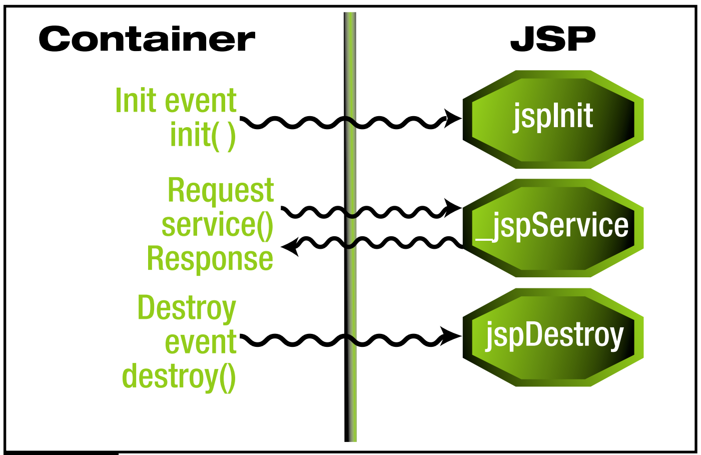


FIGURE 2 JSP contract

Choices

Back to our requirement – the solution I want to implement involves four participant types:

- ***Browsers:*** Submit HTTP requests
- ***Java servers:*** Process presentation and download the JSPs and servlets from a repository
- ***Repositories:*** Must be accessed with a URL. A suitable repositories list includes HTTP servers as depicted in Figure 3 and FTP servers
- ***Java application servers:*** Process EJB requests

I need to implement a piece of code in the Java server that's able to seamlessly retrieve JSPs and servlets from a central point, cache them, and support remote update. The solution depicted in Figure 3 is just common sense: I define a special servlet, JSPservlet, and package it in a .war file to handle all requests targeting its Web application. This servlet is responsible for loading target JSPs and servlets and forwarding them requests. To minimize data transfers, I handle archives (.jar) files only and cache downloaded archives, not only in memory but also on disk to survive a scheduled shutdown or a crash.

To simplify the development I don't handle JSP compilation. It doesn't mean the solution doesn't support JSPs, only that they have to be precompiled, not a real drawback. Compiling JSPs is the only safe way to ensure a JSP can compile, and I prefer to avoid downloading failing code. I also don't support single thread servlets that guarantee only one thread at a time will execute through a given servlet instance's service() method. The support of this feature would require instantiating a new target servlet when already created target servlets are processing a request. It would add complexity to the logic and have an adverse impact on scalability.

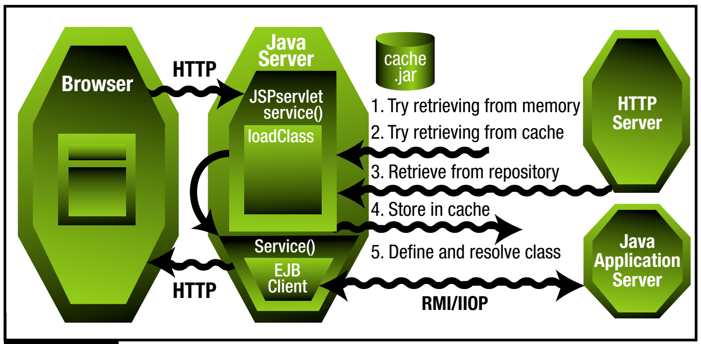


FIGURE 3 Solution

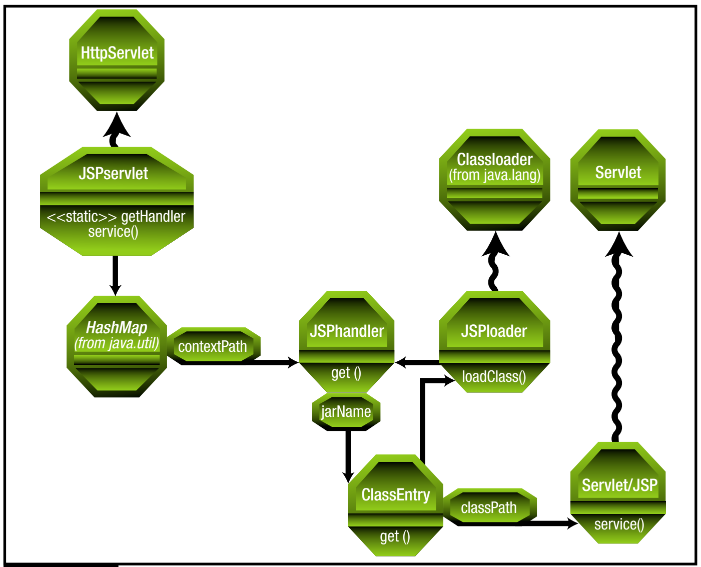


FIGURE 4 Class diagram

Listing 2 shows the deployment descriptor (web.xml) of the JSPservlet application and how to define that a JSPservlet must handle all requests targeting the application. You specify in <servlet-mapping> <url-pattern>/</url-pattern>, not <url-pattern>*</url-pattern> as you'd expect. Note that I use <init-param> to set every machine-dependent parameter. Deployers can then modify them to accommodate different installation and operating system requirements. cachePath is the directory in which downloaded JARs are locally stored, and remoteLocations indicates a property file in which remote locations are defined. For instance, if a JAR file named *myjar* must be downloaded from an HTTP server www.mysite.com, remoteLocations will contain an entry myjar=HTTP://www.iamakishirofan.com.

Implementation

Let's look at the class diagram in Figure 4. You see the aforementioned JSPservlet that relies on a JSPhandlers HashMap of JSPhandler. There's a JSPhandler instance per application that reads parameters and maintains ClassEntry objects, one per archive. ClassEntry maintains a target servlets cache and a JSPloader instance.

JSPloader is the class loader itself and maintains a class cache. It's also responsible for saving locally downloaded archives.

We can now see how the solution works. The Java server calls JSPservlet service(). To know which servlet is requested, JSPservlet.service() uses the request object. It first finds the appropriate JSPhandler with getHandler(), passing the application name it retrieves using the request getContextPath(). Then it gets a reference on the target with JSPhandler.get(), passing the path to the target returned by the request getPathInfo(). Eventually it uses this reference to invoke the target object service() method. As you can see in Listing 3 that's all for JSPservlet.

Listing 4 shows the implementation of JSPhandler. Its constructor retrieves parameter values from web.xml using ServletConfig. getInitParameter() and restores remote location properties from their persisted state. I chose to use the first part of the path as the archive name and the remaining part as the class name. Given the URL www.iamakishirofan.com/gunnm/ gally/nano/machine, if the application server is configured with the JSPservlet application on gunnm, ContextPath will be gunnm, the archive will be gally, and the servlet path in the archive, nano/ machine.class. This may seem a bit rough compared to the Web application flexible mapping but it's simpler to administer and implement. So JSPhandler.get() parses the pathInfo string and uses the archive part to find the corresponding ClassEntry in classEntries HashMap. It creates a ClassEntry if the search fails and invokes its get() method.

Now we can look at the ClassEntry implementation in Listing 5. Its constructor creates a JSPloader. Its get() method first tries to get the target servlet from its instance cache, servletObjects. No matter how many times a servlet is invoked, a single object is used and reused. If the object doesn't exist yet, it uses JSPloader to retrieve its class, invokes Class.newInstance() to instantiate it, and Servlet.init() to initialize it. It's extremely close to the Java server's implementations.

Class Loader

Before diving into the last and most complicated piece of code, JSPloader in Listing 6, let's recap what a class loader is and what our class loader is supposed to do. A class loader is an object responsible for loading classes. Given the class name, it can generate or load its binary code. It inherits from ClassLoader, which provides methods you can override (loadClass is the most flexible method). ClassLoader also implements a service method, defineClass, that converts the binary code in the Java class and resolveClass that links it. JSPloader must load classes from JAR files located either in the cachePath or at a URL.

Back to our example: it retrieves the archive from the local cache in cachePath/gally.jar or downloads it from a URL, which is the value of a gally property persisted in remoteLocations. In addition, when JSPloader downloads an archive, it must save this archive in its local cache, cachePath/gally.jar.

I prefer loading classes in a JSPloader constructor to minimize disk and network access duration and numbers. Another advantage is that forced loading can be performed outside peak hours by an administration JSP. JSPloader will then deliver a better response time as classes are already in memory. I found the memory use – same order of magnitude as the size of a downloaded archive – wasn't a showstopper. Note that I link a class only when requested, and ClassEntry instantiates objects only once, when they're first requested.

The JSPloader constructor tries downloading the archive from the local cache with loadClassDataFS() and then from its remote location with loadClassDataURL(). Both methods build a JarInputStream from an input stream that loadClassDataFS() gets from a FileInputStream and loadClassDataURL gets from a URL.openStream(). Since the JarInputStream handling is the same, I implemented it in a parseStream method.

parseStream loops around JarInputStream.getNextJarEntry(), which reads the next JAR file entry and positions the stream at the beginning of the data. Once parseStream has a JAR entry, it gets its name with JarEntry.getName() and uses a BufferedInputStream to read it. Then it converts it to a class with ClassLoader.defineClass and stores it in a classes memory cache. When it has to locally store a remotely downloaded archive, it uses a JarOutputStream; each time it's read an entry it rewrites it using JarOutputStream. putNextEntry() and JarOutputStream.write().

loadClass is invoked with two arguments, the name of the class and a boolean, resolve, that indicates if the class must be linked. Here I use the passive mode on purpose. Who invokes loadClass()? It depends. When ClassEntry invokes loadClass with the class name only, no magic happens. ClassLoader implements a loadClass(name) method that invokes loadClass(name, false). But the loaded class is associated with a JSPloader instance, which becomes the current class loader. If the loaded class uses another class, the Java Virtual Machine (JVM) will invoke JSPloader.loadClass to load it. This is why JSPloader.loadClass delegates class loading for the classes it doesn't find in its classes cache to the system class loader and its parent through the loadForward method.

The JSPloader.loadClass also delegates in two other interesting cases. If the class name starts with "java.", ClassLoader refuses to create it for security reasons. So I don't even try. The other case is "javax.servlet.Servlet". ClassEntry casts the target object it creates in a servlet. As I said, every class is associated with a class loader instance. In fact the JVM maintains the uniqueness of class_name, class_loader_object and not of class_name alone. So a cast of an object of class A loaded by class_loader_object1 to the same class loaded by class_loader_object2 fails. Therefore I check javax.servlet.Servlet and don't risk loading it from the archive.

Considerations

The order of the search has an obvious security impact. I prefer trying the class's memory cache first for speed and flexibility: I really depend on the Java server JDK for Java. I can download anything else, including the EJB, JMS, or RMI library code, but it has a security impact. If you don't trust your remote location, it's safer searching locally first.

My code is reasonably close to JDK 1.1 code: just replace HashMap with Hashtable and JarInputStream with ZipInputStream to run it with JDK 1.1. If local caching and JDK 1.1 have no value for you, consider URLClassLoader as an alternative to JSPloader. However, it's not really optimized for server-side use and you'd probably prefer the compatible NetworkClassLoader of Harish Prabandham provided in Tomcat. Its design is similar to JSPloader but instead of caching defined classes, it caches class data.

In Part 2 I'll describe how to handle images, support Web applications without restriction, and require updates from a browser. In Part 3 I'll demonstrate how to host downloaded classes in a sandbox, such as applets.

Conclusion

Through the class loader comprehensive mechanism it's easy to write a tool that's able to download servlets and JSPs from a remote location. It's even relatively easy to make it portable, though Java servers are probably the most hostile environment since they use class loaders intensively.

The idea probably has value for corporate intranets and B2Bs. Assume company B wants to provide access to its Web application to company A, which maintains a Java server. It simply configures its server to automatically download the code from B to enjoy reduced communication bills and better response time. It's a win-win situation since B doesn't have to process presentation. Now suppose A has many partners. As each downloaded archive is processed by a different class loader instance, it can use the same class names without collision. If A uses a different Web application for each partner, the partners won't share the same context. And A partners don't even have to know about A's Java server host and operating system. However, its real potential may be elsewhere.

If we could define a standard describing how to require a download and from where – for instance with XML over HTTP – even ISPs could host pages. Presentation would become a commodity like routing or a name service.

Author Bio

Alexis Grandemange is an architect and system designer. A Java programmer since 1996 with a background in C++ and COM, his main interest is J2EE with a focus on design, optimization, and performance issues.

agrandemange@amadeus.net

Listing 1: Apache implementation of _jspService and jspInit

```
public final void init(ServletConfig config)
  throws ServletException {
  this.config = config;
  jspInit();
}
public final void destroy() {
  jspDestroy();
}
public final void service(
  HttpServletRequest request,
  HttpServletResponse response)
  throws ServletException, IOException {
  _jspService(request, response);
}
```

Listing 2: JSPservlet web.xml

```
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, -
Inc.//DTD Web Application 1.2//EN"
"http://java.sun.com/j2ee/dtds/web-app_2_2.dtd">

<web-app>
  <servlet>
    <servlet-name>JDJloader</servlet-name>
    <servlet-class>JDJloader.JSPservlet
    </servlet-class>
    <init-param>
      <param-name>cachePath</param-name>
      <param-value>C:/temp</param-value>
      <description>local cache</description>
    </init-param>
    <init-param>
      <param-name>remoteLocations</param-name>
      <param-value>C:/temp/jdj.properties</param-value>
      <description>jar remote location</description>
    </init-param>
  </servlet>
  <servlet-mapping>
    <servlet-name>JDJloader</servlet-name>
    <url-pattern>/</url-pattern>
  </servlet-mapping>
</web-app>
```

Listing 3: JSPservlet code

```
public class JSPservlet extends HttpServlet {
  public static HashMap JSPhandlers = null;
  public static final synchronized JSPhandler
    getHandler(ServletConfig sc,
    String contextPath) throws ServletException {
    JSPhandler jh = null;
    if (JSPhandlers == null)
      JSPhandlers = new HashMap();
    else
      jh = (JSPhandler)JSPhandlers.get(
        contextPath);
    if (jh != null)
      return jh;
    jh = new JSPhandler(sc, contextPath);
    JSPhandlers.put(contextPath, jh);
    return jh;
  }
  public void service(HttpServletRequest request,
    HttpServletResponse response)
    throws ServletException, IOException {
    JSPhandler jh = getHandler(getServletConfig(),
      request.getContextPath());
    Servlet srv = jh.get(request.getPathInfo());
    srv.service(request, response);
  }
}
```

Listing 4: JSPhandler code

```
public class JSPhandler {
  String cachePath;
  HashMap classEntries = new HashMap();
  Properties remoteLocProp = new Properties();
  ServletConfig servletConfig;
  JSPhandler(ServletConfig sc,String contextPath){
    servletConfig = sc;
    cachePath = sc.getInitParameter("cachePath");
    if (cachePath == null)
      cachePath = "C:/temp";
    String remoteLocFile = sc.getInitParameter(
      "remoteLocations");
    if (remoteLocFile == null)
      remoteLocFile = cachePath + contextPath +
        ".properties";
    File f = new File(remoteLocFile);
    if ((f != null) && f.exists()) {
      try {
        remoteLocProp.load(new DataInputStream(
          new FileInputStream(f)));
      }
      catch(Exception e) {}
    }
  }
  final synchronized Servlet get(String pathInfo)
    throws ServletException {
    String fullName = pathInfo;
    if (pathInfo.startsWith("/"))
      fullName = pathInfo.substring(1);
    int idx = fullName.indexOf('/');
    String jarName = fullName.substring(0, idx);
    String classPath = fullName.substring(idx + 1);
    ClassEntry ce = null;
    if (classEntries.containsKey(jarName)) {
      ce = (ClassEntry)classEntries.get(jarName);
      return ce.get(classPath);
    }
    ce = new ClassEntry(this, jarName);
    classEntries.put(jarName, ce);
    return ce.get(classPath);
  }
}
```

Listing 5: ClassEntry code

```
class ClassEntry {
  JSPhandler handler;
  JSPloader jl;
  HashMap servletObjects;
  ClassEntry(JSPhandler jh, String jarName)
    throws ServletException {
    handler = jh;
    String jarURL = (String)
      jh.remoteLocProp.get(jarName);
    jl = new JSPloader(jh, jarName, jarURL);
    servletObjects = new HashMap();
  }
  final Servlet get(String classPath)
    throws ServletException {
    if (servletObjects.containsKey(classPath))
      return (Servlet)
        servletObjects.get(classPath);
    Servlet srv = null;
    try {
      Class jspClass = jl.loadClass(
        classPath.replace('/', '.'));
      srv = (Servlet)jspClass.newInstance();
      srv.init(handler.servletConfig);
      servletObjects.put(classPath, srv);
    }
    catch(Exception e) {
      throw new ServletException("ClassEntry.get("
        + classPath + ") " + e);
    }
    return srv;
  }
}
```

Listing 6: JSPloader code

```
public class JSPloader extends ClassLoader {
  JSPhandler handler;
  String jarURL;
  String jarName;
  HashMap classes = null;
  ClassLoader parent;
  public JSPloader(JSPhandler jh, String jarName,
    String jarURL)
    throws javax.servlet.ServletException {
    super();
    handler = jh;
    this.jarURL = jarURL;
    this.jarName = jarName;
    parent = getParent();
    if (!loadClassDataFS()) {
      if (!loadClassDataURL())
        throw new javax.servlet.ServletException(
        "JSPloader.JSPloader unable to load jar");
    }
  }
  private final boolean parseStream(
    JarInputStream jis, boolean toSave) {
    JarEntry je = null;
    boolean rc = true;
    try {
      JarOutputStream jos = null;
      if (toSave)
        jos = new JarOutputStream(
          new BufferedOutputStream(
          new FileOutputStream(handler.cachePath +
          "/" + jarName + ".jar")));
      while((je = jis.getNextJarEntry()) != null){
        String entryName = je.getName();
        if (entryName.endsWith(".class")) {
          if (toSave)
          jos.putNextEntry((JarEntry)je.clone());
          ByteArrayOutputStream baos =
            new ByteArrayOutputStream();
          BufferedInputStream bis =
            new BufferedInputStream(jis);
          int i;
          while((i = bis.read()) != -1)
            baos.write(i);
          if (classes == null)
            classes = new HashMap(100);
          byte[] buf = baos.toByteArray();
          String k = entryName.substring(0,
  entryName.lastIndexOf('.')).replace('/', '.');
          Class jarCl = defineClass(k, buf, 0,
            buf.length);
          classes.put(k, jarCl);
          if (toSave)
            jos.write(buf, 0, buf.length);
        }
        jis.closeEntry();
      }
      jis.close();
      if (toSave) {
        jos.closeEntry();
        jos.close();
      }
    }
    catch(Exception e) {
      rc = false;
    }
    return rc;
  }
  private final boolean loadClassDataFS() {
    String jarPath = handler.cachePath + "/" +
      jarName + ".jar";
    JarInputStream jis = null;
    try {
      jis = new JarInputStream(
        new FileInputStream(jarPath));
    }
    catch(Exception e) {
      return false;
    }
    return parseStream(jis, false);
  }
  private final boolean loadClassDataURL() {
    JarInputStream jis = null;
    try {
      URL url = new URL(jarURL + "/" + jarName
        + ".jar");
      InputStream is =
        url.openConnection().getInputStream();
      jis = new JarInputStream(is);
    }
    catch(Exception e) {
      return false;
    }
    return parseStream(jis, true);
  }
  private final Class loadForward(String name)
    throws ClassNotFoundException{
    try {
      return findSystemClass(name);
    }
    catch(ClassNotFoundException cnfe) {}

    try {
      return parent.loadClass(name);
    }
    catch(ClassNotFoundException cnfe) {
      throw cnfe;
    }
    catch(Exception e2) {
      throw new ClassNotFoundException(
        e2.toString());
    }
  }
  public synchronized Class loadClass(String name,
    boolean resolve)
    throws ClassNotFoundException {
    if (name.equals("javax.servlet.Servlet") ||
      name.startsWith("java."))
      return loadForward(name);
    if ((classes != null) &&
      (classes.containsKey(name))) {
      Class cl = (Class)classes.get(name);
      if (resolve)
        resolveClass(cl);
      return cl;
    }
    return loadForward(name);
  }
}
```

Download the Code!

The code listing for this article can also be located at

www.JavaDevelopersJournal.com

Performance Management Starts with IDL Design

An early focus on IDL design is key in the CORBA environment

WRITTEN BY
KHANH CHAU

At the enterprise level, building and deploying distributed object-oriented components involves a dizzying number of choices and considerations. In contrast to a single-process monolithic system, distributed computing provides the flexibility to delegate computing processing power to a large number of nodes, allowing us to build highly complex systems. Coupled with this flexibility are issues that arise from a system's distributed nature.

From a technical perspective, making the transition to *n*-tier architecture requires risk management of such issues as network latency, system responsiveness, service availability, load management, distributed caching, distributed garbage collection, and system management. Furthermore, for every novel solution to optimize system efficiency, more issues are created that must be addressed. Despite this gloomy outlook, the technical risks in building distributed OO components for large-scale enterprise systems can be managed by using fundamental design techniques.

For instance, by carefully analyzing the problem domain requirements, early design decisions can help minimize network traffic, thus improving overall system responsiveness.

This article outlines a common approach at the Interface Definition Language (IDL) level using an iterator pattern to govern the amount of data passing over the wire. Several issues are also addressed such as caching and distributed garbage collection, both of which can be solved using JDK features.

Performance Issues

Although CORBA abstraction helps shield underlying network infrastructure complexity, it doesn't guarantee the construction of a reliable high-performance system. To achieve some goals, the overall system architecture design must consider the underlying network infrastructure. According to the authors of *Enterprise CORBA*, three factors affect CORBA-based system performance:

1. Number of remote invocations
2. Amount of data transferred
3. Marshaling costs of different data types

Fortunately these issues can be mitigated if they're anticipated early in the design cycle. Observations indicate that a processing delay occurs when sending data over the wire. If a system design seeks to minimize network traffic caused by interactions among distributed components, the system performance improves accordingly. In CORBA-based systems the IDL plays an important role in component interactions as it defines interfaces in which servants comply. Hence, the logical place for applying design techniques is in the IDL design.

IDL Design

A common IDL design issue that's frequently overlooked is determining which interfaces are candidates for servants, and transient and persistent CORBA objects.

- A servant is a programming-language–dependent object that implements an object's operations (CORBA 2.4 specs). In the CORBA programming model servants are registered with the Portable Object Adapter (POA), which arbitrates the lifetime of the servants for the requests.
- In contrast to servants, transient CORBA objects aren't registered with the POA and are usually created by the servant during request processing. These transient objects don't live beyond the life of a process or (sometimes) the thread that created it. Their object references aren't published.
- Persistent CORBA objects associate with a persistent state and have special uses.

This article focuses on using transient CORBA objects to manage large data transfers. These data-throttling techniques are desirable when the potential exists for discarding data. For example, the user specifies a query that returns a large result. After viewing the first 20 items, another query is made that results in discarding the remaining data. In a single-process application such usage isn't a problem, but in distributed computing it wastes network bandwidth and CPU processing time.

For the basis of discussion, Figure 1 provides a reference of a trivial interaction between a client and a server. The client proxy is a remote proxy class that manages the connection and delegates client application requests to a remote invocation. As indicated, the client makes a remote request to a servant that services the request and returns an array or sequence of product objects. If the result contains *n* product objects, *n* elements are marshaled and sent back over the wire. At first glance this approach is acceptable unless the requirement is to remedy undesirables as previously outlined.

The following code shows an IDL snippet that defines the interaction. The IDL content is straightforward, as the interface contains only one operation. This operation defines the criteria used to return an array of products. We'll return to the IDL shortly after more discussion of the client-side implementation.

```
module productcatalog
{
 struct ProductItem{
   string productName;
  …
};
typedef sequence<ProductItem> Product
ItemList;

interface ProductCatalog {
  ProductItemList getProductItems(in
string group,
 in string category, in string sta-
tus)
         raises (SomeRemoteExcep-
tion);
         };
};
```

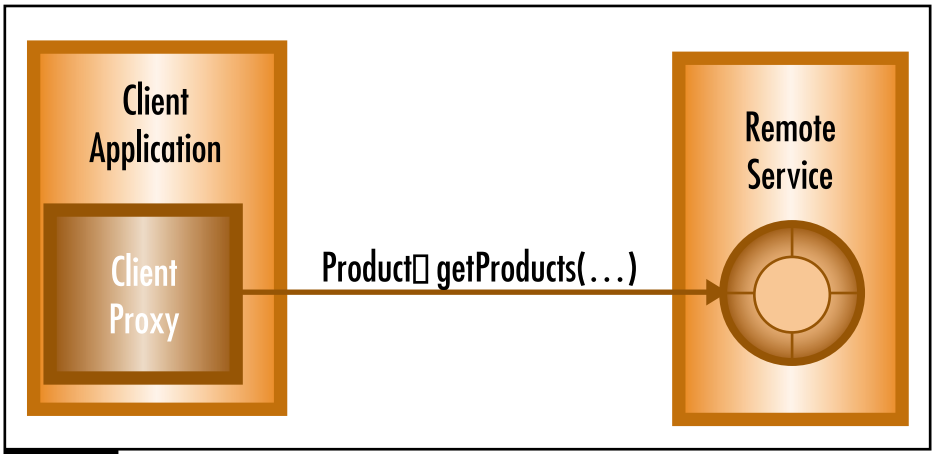


FIGURE 1 Scenario A — Return all products per request invocation

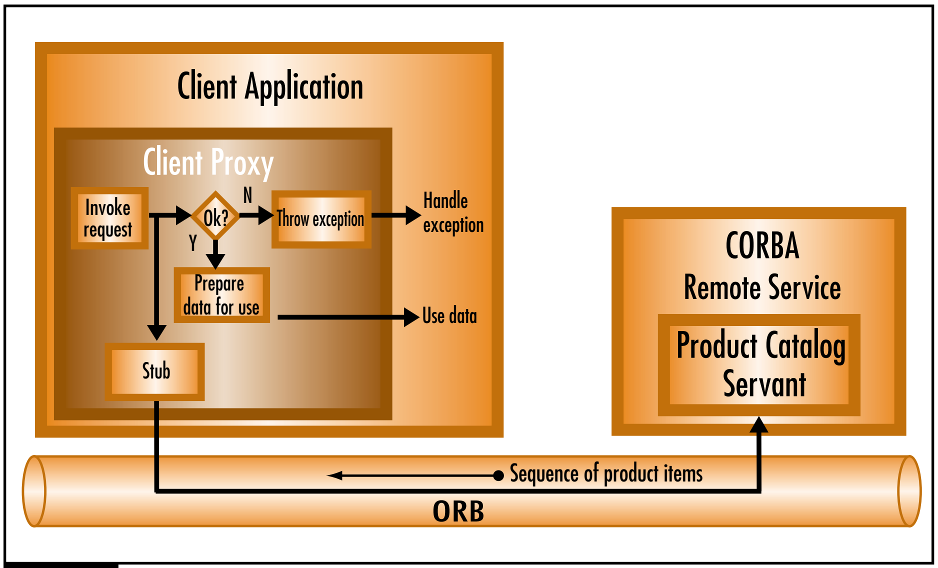


FIGURE 2 Standard procedure for handling requests

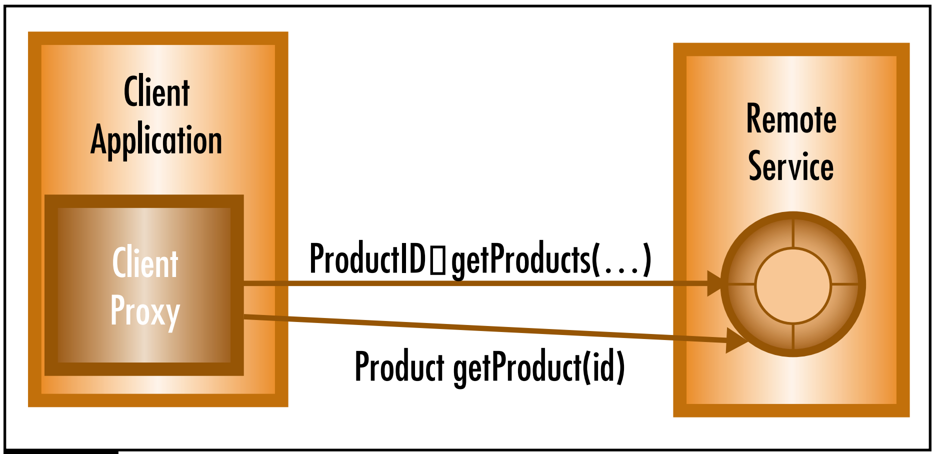


FIGURE 3 Scenario B — Return ID list, then invoke separate requests to get object using ID

Assume a remote service is functional. From the client perspective, making a request to retrieve product information involves a few steps (see Figure 2). The proxy object provides a wrapper that binds to a specific remote object instance and acts as an intermediary for managing remote invocation. Its responsibilities include:

- Preparing the request
- Delegating the request invocation to the stub
- Trapping remote exceptions (and preferably mapping them to meaningful user-defined exceptions)
- Preparing the received data structure for an object model that can be readily used by the client application

In the Product Catalog Proxy implementation (see Listing 1), the client proxy is implemented using a singleton pattern. This ensures that only one instance of the client proxy is created. For each request that requires remote invocation, the locateRemoteService() method is called to verify the remote object reference. If the connection is dropped or the object reference is no longer valid, the binding automatically occurs, enabling more resilience to failures. The implementation of locateRemoteService() isn't shown, as the programming model depends on the object location mechanism or is specific to an ORB product.

Now consider a slightly different IDL design that returns a list of product IDs instead of products (see Figure 3). The difference, in terms of network overhead, is that the ID sequence is much smaller in size than a products list. The assumption is that the UI can present the user with a list of product IDs, and the user selects one to return a complete product. This approach is an improvement over the previous one – data isn't wasted because information is transmitted according to user needs.

One shortcoming of the previous design is determining whether it's acceptable to display product IDs. Keep in mind that retrieving every product by enumerating over the product IDs array results in the same problem previously shown, except it's more severe because now there's *n*+1 remote invocations.

The next solution (see Figure 4) is to redesign the IDL to take advantage of user behavior and computing constraints. The design change is based on the following observations.

For most UI applications, such as Swing applications (thick GUI) and Web-based display, a limit exists on how many items are displayed at a time. Screen real estate limits the capability for viewing data. Usually, in the case of a thick GUI, a grid or table is used to hold a list of items.

For a Web-based presentation, search results can be broken down into pages and navigation controls that are used to display information.

In addition, it's safe to assume that given a screen of information the user needs a few seconds to digest results and determine the next action.

Developers can take advantage of the information by installing a mechanism to release the data across the wire as needed. *Note:* The basis of this need can be triggered by user interaction or autonomously (for example, by a read-ahead algorithm). For many uses, a simple implementation based on a user-initiated request is sufficient. The read-ahead approach adds complexity because it requires an algorithm and a caching mechanism. With this in mind, let's apply the following changes.

First, a mechanism is needed to regulate the flow of information. One way to accomplish this is to employ the iterator pattern. This technique requires rewriting the IDL so that large amounts of information can be broken down into chunks to be served via a remote iterator. To facilitate this implementation, it's beneficial to define the base iterator interface that specifies the behavior of the remote iterator. Listing 2 defines a BaseIterator and a BaseListIterator. This provides the basis for implementing a domain-specific remote iterator such as the ProductIterator as shown in Listing 3. Also notice that the method on the ProductCatalog is modified to return a ProductIterator instead of a Product ItemList.

On the server side the implementation of the ProductIterator plays an integral role in administering the amount of data for transfer. In essence, the ProductIterator is the transient CORBA object that materialized during the getProductItems(...). It's transient in nature because its content depends on a search result. If the result is collected and provided via a collection class such as ArrayList, ProductIterator implementation is simplified because ArrayList provides a ListIterator or Iterator on its content. Another change is in the implementation of the ProductCatalog servant. The getProduct Items(...) returns the ProductIterator object reference instead of an array of product IDs (scenario A) or products (scenario B). Since the product and product ID are CORBA data types and the server data presentation may use an enterprise object model, data mappings are required. By using the ProductIterator, the mappings are deferred to its implementation.

On the client side the proxy implementation fires off the initial request and caches the remote iterator object reference. Optionally, the proxy can immediately fetch the next *n* items before returning. Figure 5 outlines the interactions among the components. When the client application specifies the initial search request, the client proxy object performs prerequest work by checking its cache. If no information for the given criteria is available in the cache, a search request is invoked. The result is a remote iterator (a CORBA object reference) that's then stored internally in the client proxy. Subsequent requests by the client application cause retrieval of one product or a block of products. It's optional to maintain the products in the cache area. If caching is used, the internal mechanics may specify that the cache areas are cleaned up when a new search is initiated. Listing 4 presents changes in interesting areas.

As mentioned before, sometimes a

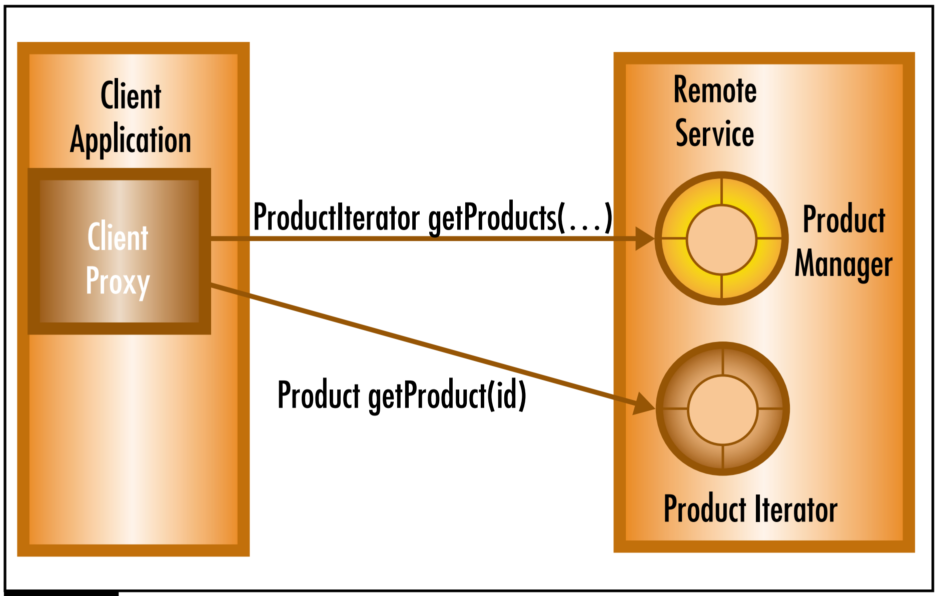


FIGURE 4 Scenario C — Return remote iterator, then use it to retrieve *n* objects

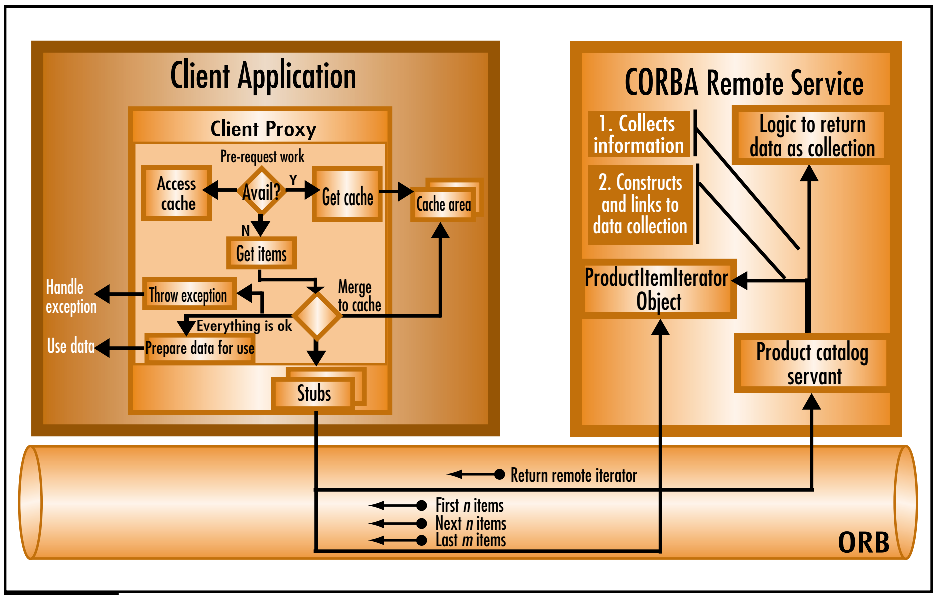


FIGURE 5 Remote iterator in IDL

Author Bio

Khanh Chau, an infrastructure architect with The Technical Resource Connection, Inc., is also an instructor and project lead for The TRC Java Developer Boot Camp program. He helped design, implement, and deploy a Spine Infrastructure Framework (SIF) Architecture for a major national real estate services company and is currently developing an enterprise e-commerce portal for one of Germany's largest insurance companies.

solution can cause more problems. Since the remote iterator is a transient CORBA object, it depends on the servant's lifecycle. A mechanism must be put in place so it doesn't get deactivated until the client is finished. On the other hand, we don't want the iterator to get garbage collected when there's no need. The latter issue relates to distributed garbage collection and can be solved with an eviction scheme. One mechanism is to trigger the remote service via a remote invocation to clean up during the initial request. Another solution is to implement a leasing mechanism via a distributed callback using the timer API, such as the one supplied by the JDK. The proper scheme depends on several design factors such as server caching, load balancing, fault-tolerance capability, static data, and dynamic data.

An interesting application is to apply the iterator approach the other way around. Consider a scenario where the client needs to submit a large block of data (such as an application for home insurance) to the server for storing and processing. It's possible to break up the form into data chunks, allowing the server to process the information in the background, thus freeing the client application for other tasks.

Conclusion

Enterprise system development isn't easy. Decisions made early in the design stage have a pronounced effect on the system's overall responsiveness. In the CORBA environment, an early focus on IDL design is key because a casual design doesn't take advantage of the strengths of the middleware. Creative IDL design, however, can create many issues during implementation time. Via utilization of a JDK API, such as timer and collection framework, developers can apply solutions to those issues that ensure a robust, highly available distributed system.

References

1. Gamma, E., Helm, R., Johnson, R., and Vlissides, J. (1995). *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley.
2. Mobray, J.T., and Malveau, C.R. (1997). *CORBA Design Patterns*. Wiley.
3. Object Management Group. (2000). CORBA/IIOP 2.4 Specification.
4. Slama, D., Garbis, J., and Russell, P. (1999). *Enterprise CORBA*. Prentice Hall.

Khanh.Chau@trcinc.com

Listing 1: Product Catalog Proxy Implementation

```
public class ProductServiceProxy {
 private static ProductServiceProxy instance;
 private ProductServiceProxy () {}

 public static ProductServiceClient getInstance() {
   if(instance==null)  instance= new ProductServiceProxy ();
   return instance;
}

public ArrayList getProducts(String group, String category,
String status)
throws SomeException{
   ArrayList productList = new ArrayList ();
   try {
  ProductService productService = locateRemotetService();
   if ( productService == null ) {
     // return empty list or throw user defined exception
   }
  // get product list
  try{
     // get products using criteria
    // when request returns data, prepare data and store
results in collection object
  } catch(SystemException se){  // handle exception}
    } catch ( Exception e) { //handle exception }
  return productList;
}
…
}
```

Listing 2: Base Iterators in IDL

```
#include "util/exceptions/idlexceptions.idl"
module iterator
{
 interface BaseIterator {
   boolean hasNext() raises (SomeRemoteException);
   short count() raises (SomeRemoteException);
 };
 interface BaseListIterator {
  boolean hasPrevious() raises (SomeRemoteException);
   short previousIndex() raises (SomeRemoteException);
   boolean hasNext() raises (SomeRemoteException);
   short nextIndex()raises (SomeRemoteException);
   short count() raises (SomeRemoteException);
 };
};
```

Listing 3: IDL Design of Service-Specific Iterator

```
#include "util/iterator/iterator.idl"
module productcatalog
{
 struct ProductItem{
  string productName;
 …
 };
 typedefsequence<ProductItem> ProductItemList;

 interface ProductIterator : iterator::BaseListIterator {
    ProductItem next()  raises (SomeRemoteException);
    ProductItemList nextBlock (in short size) raises (SomeRe-
moteException);
   ProductItem previous () raises (SomeRemoteException);
   ProductItemList previousBlock (in short size) raises
(SomeRemoteException);
 };

 interface ProductCatalog {
    ProductIterator getProductItems(in string group, in string
category, in string status)
      raises (SomeRemoteException);
 };
}
```

Listing 4: Changes to ProductServiceProxy Implementation

```
public class ProductServiceProxy {
 private int nItems=15;
 private ProductIterator remoteIterator;
 private static ProductServiceProxy instance;
  private ProductServiceProxy () {}

public static ProductServiceClient getInstance() {
   if(instance==null)  instance= new ProductServiceProxy ();
   return instance;
}

public ArrayList getProducts(String group, String category,
String status)
throws SomeException{
   ArrayList productList = new ArrayList ();
   try {
  ProductService productService = locateRemoteProductSer-
vice();
   if ( productService == null ) {
     // return empty list or throw user defined exception
   }
  // get product list
  try{
     remoteIterator=productService.getProducts(…);
  } catch(SystemException se){  // handle exception}
    } catch ( Exception e) { //handle exception }
  return this.getMoreProducts(nItems);
}
public ArrayList getMoreProducts(int nItems) throws SomeExcep-
tion{
      // retrieve the next 10-15 items using cache iterator
}
…
}
```

Download the Code!

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

Interview...with Steve Rock

VP of Electronic Global Broadcasting System

An interview by Alan Williamson

This is the first of a series of interviews from key people who are putting Java to serious use. We're proud to have Steve Rock from EGBS for a truly interactive chat session that spanned halfway 'round the world. EGBS is responsible for building and deploying a specialized video/media server that allows, among other things, both content producers and users to easily hot link particular areas of live video.

<Alan@JDJ> I'd like to welcome Steve Rock, vice president of EGBS. Thanks for taking the time to speak with us, Steve.
<Steve Rock> Thank you for this opportunity. I've read ***JDJ*** for the past three years. I find your magazine highly useful.

<Alan@JDJ> Can you give us a brief overview of what EGBS is up to? I hear you're doing some pretty cool stuff.
<Steve Rock> At EGBS we specialize in video production and streaming to broadband users on the Internet.

<Alan@JDJ> Is this using your Spectrum Server product?
<Steve Rock> Correct. As a test bed for developing our Spectrum Server Product Suite we've built two music Web sites that provide users with the choice of which music videos they wish to view, anytime, anywhere – HitMusic.com and IfItRocks.com.

<Alan@JDJ> Very slick sites. Tell us what makes Spectrum different from more conventional streaming technologies, such as RealAudio.
<Steve Rock> The basic streaming technology is similar. We use the QuickTime format because the Java API gives us great control over the internals of the video. However, what sets our Spectrum Products apart is that we've built a framework that allows us to build highly interactive applications in which the user can mark up the video, be it video bookmarks, drawing on the video, or launching URLs, which are based on the context, in an HTML browser as the video plays.

We've basically developed a suite of tools that allows production teams or even end users to treat a video as a time-based drawing template that can be wired to any type of interactivity.

<Alan@JDJ> So to illustrate this a little more, if we were watching the Charlie's Angels movie from a Spectrum Server and Lucy Liu picks up her mobile phone to make a call, the viewer can click on the mobile phone and be taken to the phone's Web site for more information without disrupting the flow of the movie. Is this correct?
<Steve Rock> Exactly. The user could also pause the movie as it's playing and add bookmarks with a note attached. By clicking on a bookmark in a list, the user can jump to that point in the movie. These bookmarks can be indexed and searched by keyword.

<Alan@JDJ> Wild. So the user, not just the producer, can bookmark particular movies?
<Steve Rock> Yes. Anything a producer can do, an end user can do. The user can even make drawings on the movie, such as an editor circling a piece of the movie he or she doesn't like, and add a description. This can be saved and forwarded to other editors.

<Alan@JDJ> What has the advertising/movie industry commentary been on this technology? It must be a godsend to them, especially with the trend in Hollywood for more product placements within mainstream movies. Your technology effectively gives them the power to capture users without forcing them to remember ugly URLs.
<Steve Rock> We've received fantastic feedback. We have several opportunities opening up for us; I can't provide the details at this point. The reason the industry is so excited is that with other tools on the market, to build all the interactivity I mentioned, *a lot* of manual production takes place. Everything is basically hand-coded. Our Spectrum Suite removes this manual programming process and puts the power into the end users' hands.

One exciting agreement we've just entered into is with a major film studio to produce a "Webisode" TV show on the Web that will provide the viewer with multiple paths for viewing each episode.This is a very exciting project for us because we'll be incorporating all the functionality we've been talking about.

<Alan@JDJ> Can you provide details for that? Or is it all hush-hush?
<Steve Rock> It's a joint venture with Screen Gems, a major film studio, to produce TV shows for the Internet.The question is, we have TV, why would you want to put a TV show on the Internet. We believe the Internet and TV will merge at some point, and this project is one of the first to expand on the completely passive TV experience to immerse the viewer into the show. They can choose how the plot progresses and what plot lines to follow.

We just had a press release go out on www.hollywoodreporter.com/archive/hollywood/current/webwatch/webwatch08.asp.

<Alan@JDJ> Let's get behind the functionality and discover what's happening in the background.
<Steve Rock> What would you like to know?

<Alan@JDJ> Well, call us old fashioned but being a Java magazine, tell us how you're using Java in this? Take us behind the scenes as it were.What's going on in the background? How are you using Java to achieve this level of functionality?
<Steve Rock> Java is immersed in every aspect of the project. One reason we call it Spectrum is because this technology can be scaled from a stand-alone Java application that runs off a CD-ROM to a full-blown Web site with database persistence storage.

<Alan@JDJ> Excellent. So are you using any J2EE APIs or the Java Media Interface?
<Steve Rock> The Web site is completely built upon J2EE. Our design has been modeled after the Sun Pet Store Demo but uses XML for all configuration.

<Alan@JDJ> Spectrum is a J2EE application. Is this primarily EJBs or Servlets?
<Steve Rock> The Spectrum Web component is a combination of EJBs, Servlets, and JSP pages. All the business logic is in the EJB tier, the Web tier is mainly for presentation, and the client interface for the application is Swing.

<Alan@JDJ> You mentioned earlier that you use the QuickTime format for the video. Is this stored within a database or fed from static files?

<Steve Rock> Well, the QuickTime format means a movie file is comprised of many tracks that are time-based. These tracks can be video tracks, audio tracks, text, etc.

<Alan@JDJ> Where are these tracks originating?
<Steve Rock> The video itself lives in a separate static file.

<Alan@JDJ> Are you reading this via the [java.io] interface in real time?
<Steve Rock>This is all proprietary to the QuickTime API for Java.

<Alan@JDJ> At what point do you add your content to the live stream, for example, the hot links?
<Steve Rock> I can't divulge all the details, but we can create QuickTime Events within the reference movie. So in viewing mode, as the movie plays, we can capture these events and based on an ID number tie them to a database. At this point we can do anything we want within Java.

<Alan@JDJ> I'm interested in looking at the overall system. You're in an environment where performance is critical. You're required to guarantee a throughput and I was wondering if you found Java to be up for the task?
<Steve Rock> Performance is critical, and Java poses no problem with that. The bottleneck isn't Java; it's usually the Internet connection.

What was more important to us was scalability from stand-alone systems to a full distributed system. Java is ideal for this because we build EJBs that almost mirror the same bean that's used in a stand-alone application.

<Alan@JDJ> So at the heart it's the same code?
<Steve Rock> The business components are basically the same code. Of course the user interface is completely different – Servlets and JSP for the Web site and SWING for the stand-alone application components. Using the model-view-control design helped us separate these components.

<Alan@JDJ> You seem to be using the complete, dare I say, spectrum of the Java API!
<Steve Rock> You got it!

<Alan@JDJ> Can you tell us if you have noticed a significant difference between the application servers you've deployed and tested with?
<Steve Rock> I hope I don't step on any toes with this hot topic, but the answer is a definite *yes!*

<Alan@JDJ> Honesty is never stepping on any toes – you're in an environment where performance is key. Tell us what you found.
<Steve Rock> iPlanet Application Server may perform quite well in production, but I don't know – I never got that far.

<Alan@JDJ> What are you running hitmusic.com with?
<Steve Rock> Their development and deployment environments were so lousy that we couldn't build anything in a reasonable amount of time.

We're currently running hitMusic with WebObjects. This app server is fairly easy to develop with, but doesn't hold up well in production. We have to reboot it every two days because of memory leaks.

<Alan@JDJ> With respect to specific app servers, are you using any vendor-specific libraries or are you pure J2EE?
<Steve Rock> We're trying not to use any vendor-specific code at all to give us the flexibility to change as needed.

I haven't tried moving our J2EE code across vendors yet, but it seems that required changes are minimal.

<Alan@JDJ> Did you look at any other technology before choosing to go with Java?
<Steve Rock> Not really. A few years ago I chose to invest my time in becoming an expert in Java, and I've been greatly rewarded. As VP of engineering, I've standardized our company on Java. I'd need a real strong case to choose another platform.

<Alan@JDJ> And in the words of Prince George from "Blackadder" …hurrah!

Well Steve, I'd like to thank you for taking the time to give us our first-ever JDJ-IRC chat interview.

For more information on the Spectrum Media Server visit: www.e-gbs.com/.

Author Bio

Alan Williamson is CEO of the first pure Java company in the UK, n-ary (consultancy) Ltd (www.n-ary.com), a Java solutions company specializing in delivering real-world applications with real-world Java. Alan has authored two Java servlet books and contributed to the servlet API.

alan@sys-con.com

PRODUCT REVIEW

Rose JBuilder Link

by Ensemble Systems

REVIEWED BY NEAL FORD

AUTHOR BIO

Neal Ford, vice president of technology at the DSW Group, is also the designer and developer of applications, instructional materials, magazine articles, and video presentations

nford@thedswgroup.com

JAVA DEVELOPER'S JOURNAL JDJ WORLD CLASS AWARD

Ensemble Systems Inc.
280-5200 Hollybridge Way
Richmond, BC, Canada
V7C 4N3
Web: www.ensemble-systems.com
Phone: 604 231-9510
Fax: 604 231-9545
E-mail: info@ensemble-systems.com

Test Environment
OS: Windows 2000
JProcessor: 650MHz Pentium III laptop
Memory: 256MB

One of the most important but least used techniques in software development is proper design before implementation. Everyone knows this, but it seems that no one does it. Insane development schedules, pointy-haired manager types who believe that the only "real" artifact produced by a developer is source code, and a host of other events conspire to keep development as a nonengineering pursuit. However, those who have used good design (in the form of use cases, sequence diagrams, class diagrams, and so on) find that it reduces the number of required changes late in the project, calls for fewer design changes after coding has started, and shortens development schedules.

The tool that's generally considered the Rolls Royce of CASE (Computer Assisted Software Engineering) is Rational Rose. It's the market leader and has a well-established reputation for supporting large-scale development projects. In fact, the "three amigos" who created the UML (Unified Modeling Language), Booch, Rumbaugh, and Jacobson, all work for Rational. Rose supports not only diagrams and other design artifacts, it also includes code generators to realize its diagrams in source code. It comes with built-in support for C++, Java, Visual Basic, SQL, and a few other languages. Alas, the Java code generation within Rose is a bit weak. It will generate classes based on diagrams, but its capabilities are very rudimentary. Of course, it would be nice if Rational directly supported JBuilder. This is the gap that Ensemble's Rose JBuilder Link seeks to fill.

Before talking about the product, however, a bit of background on RoseLink extensions is in order. Rose has extensive support for third-party add-ins. This may include code generators, version control packages, or anything else that developers might want to access from within Rose. In fact, an entire category of add-ins called *Links* exists just for code generation from models. Basically three types of links can be created:

- ***Forward engineering:*** Takes model diagrams and produces source code. However, once the code has been produced, the link has nothing more to do with it.
- ***Reverse engineering:*** Takes source code and produces models from it, showing all the relationships from the code in the diagrams. However, it can't take the resulting models and generate source code.
- ***Round-trip:*** Performs both of the above tasks – it can generate code from the model and vice-versa. The best round-trip links will make sure the code is synchronized, so the user canmake changes in both the model and the code, then synchronize the changes. Obviously, the last type of link is the most powerful and therefore the most desirable (and not surprisingly, the most difficult to write).

This brings us to Ensemble's Rose JBuilder Link, a full round-trip link with synchronization. To use it, you must have Rational Rose installed. It currently supports Rose 98, 98i, and 2000. Note that it won't work without Rose; it's written as an add-in tool so there's no standalone version, which is typical of Rose links. It's a standard InstallShield installation, actually a series of linked installs, each automatically following the other. So when you install it, you'll get a bunch of installation notices flying by. However, it's completely painless.

The next time you run Rose, you'll see a menu item under Tools for Ensemble Tools. The first time you run it, it will ask for a license. It looks like they've started using the same type of licensing arrangement that Rose uses, either a single license or a floating license. I'm less than enthusiastic about the way the licensing works in Rose (and consequently in Rose JBuilder Link), but I suppose it's a necessary evil. One of the options presented in the licensing manager is for a trial license, which lasts 30 days. This allows you to download a copy from their Web site and put it through its paces for a month. There's one quirk that's a side effect of the licensing. When it generates a license, it looks at the network characteristics of your machine.

I installed it while I was connected to the Internet via a dial-up connection and everything worked great. However, I tried to run it later when I wasn't dialed into the Internet and it refused to run because the license couldn't identify my machine. As soon as I logged back onto the Internet, the 30-day license reappeared. What's worse, when I went back into the office and could connect through the Internet using the office LAN, it still couldn't see the license. The only way I could use the license was if I were connected to the Internet in exactly the way I was when the license was generated.

Sigh. Maybe someday tools that are paranoid about licensing will get it absolutely right. For now, it generates minor inconveniences. I can't begrudge software vendors worrying about licensing and pirated software, but it's less than optimal if it gets in the way of legitimate use. The moral: before you run Rose JBuilder Link the first time and generate the license, make sure you're connected to the outside world in the way you'll be when you're working. I don't know if this same problem manifests itself with a "real" license. If it does, it's a serious shortcoming because it wouldn't allow me to use Rose JBuilder Link when I'm away from the office. After talking to Ensemble tech support, they assured me that this is a known, intermittent problem solely with laptops, and it only affects the trial version. They also stated that they are working on a solution.

I ran Rose JBuilder Link on a 650MHz Pentium III laptop with Windows 2000 and 256MB of mem-

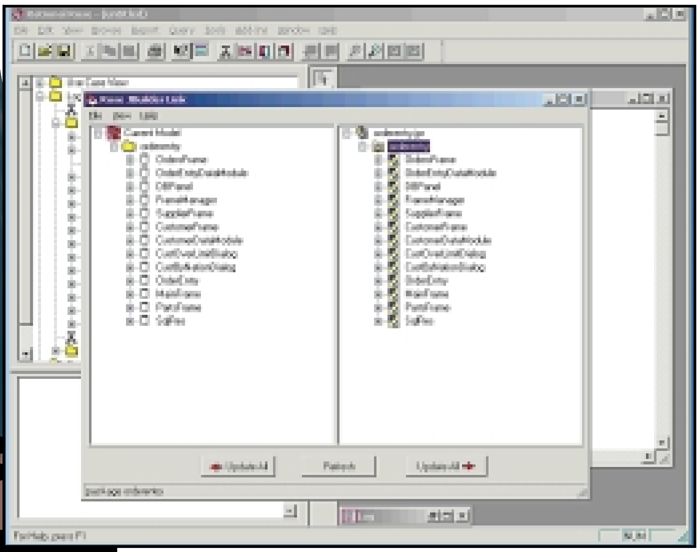


FIGURE 1 Rose JBuilder Link shows the object model and the JBuilder project side by side.

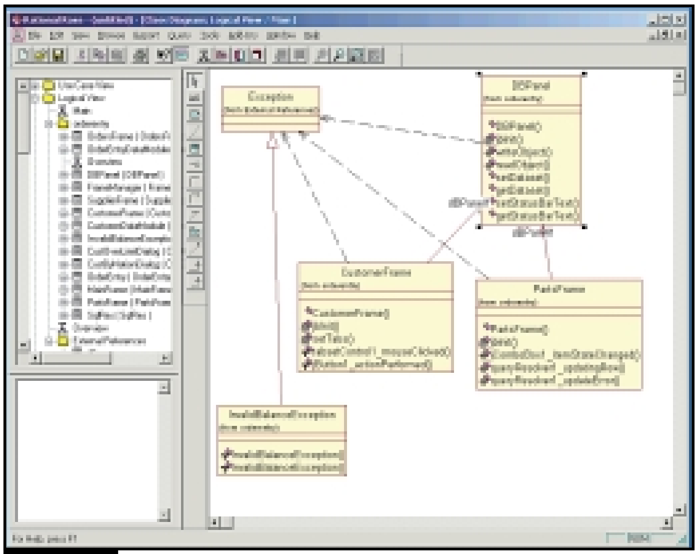


FIGURE 2 Rose JBuilder Link not only builds the classes, but correctly establishes the relationships between them.

ory. However, it takes up fewer resources than Rose itself, and most of its work is done with file generation; thus the I/O subsystem on your machine makes more difference than the processor. The bottom line: if your machine will run Rose, it should run Rose JBuilder Link with no problem. And if your machine runs JBuilder, you have more than enough machine to run just about anything else!

As far as the tool itself, it does an impressive job. As a stress test, I took a fairly complex JBuilder project and used Rose JBuilder Link to reverse engineer it to get class diagrams. The UI link has an intuitive user interface, showing the Rose model in an outline on the left and the JBuilder project on the right as seen in Figure 1.

Rose JBuilder Link reverse engineered the project flawlessly. When it was done, I had an object model based on the project. And it doesn't just pull the objects into Rose and create Rose references to them. It also establishes the relationships between the classes – those within the project and those that are part of the Java libraries. For example, Figure 2 shows a couple of the resulting classes that it created.

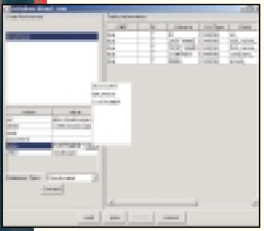
As you can see, both the CustomerFrame and PartsFrame classes use the DbPanel JavaBean for user interface chores. The class diagram correctly links these classes together with a dependency arrow. Also, the

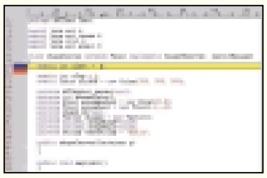
—continued on page 104

Kawa 5.0

Allaire Corporation

Kawa 5.0 is an integrated development environment for J2EE application development. A streamlined J2EE-compliant visual tool, it makes Java development accessible to many. Advanced capabilities include a debugger that supports multithreaded debugging and conditional breakpoints, as well as an extensibility framework for easy customization.





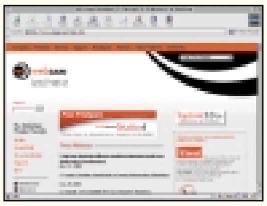
Kawa 5.0 complements Allaire JRun Studio and is available in two editions, Professional and Enterprise.

Available: Immediately
Contact: www.allaire.com

TopLink 3.0

WebGain

TopLink simplifies application development by bridging the gap between Enterprise JavaBean objects and relational databases. Its powerful runtime architecture allows developers to utilize existing corporate data for scalable application development.



TopLink for Java is compatible with major application servers, including IBM WebSphere and BEA WebLogic. It will be included in the Professional Edition of WebGain's flagship product, WebGain Studio.

For businesses, TopLink eliminates database redesigns and results in a reduction in time-to-market for application development, along with better predictability and scalability at runtime.

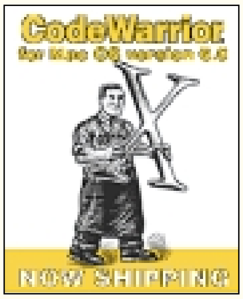
Available: Immediately
Contact: www.webgain.com

CodeWarrior

For Java Version 6.0

Metrowerks

CodeWarrior for Java is a set of development tools that can be used to create applications for wireless devices. The tools are the first to support the Mobile Information Device (MID) profile, targeted at wireless devices with limited memory and processor power.



CodeWarrior for Java provides a project manager and build system, class browser and code navigation system, text editor, debugger, emulators, and drag-and-drop GUI development tools. Developer productivity is enhanced with wizards for creating applets, applications, and JavaBeans.

The tools also offer extensibility and customization options, and can import and export project and target settings in XML. The PointBase database is included.

Available: Immediately
Contact: www.metrowerks.com

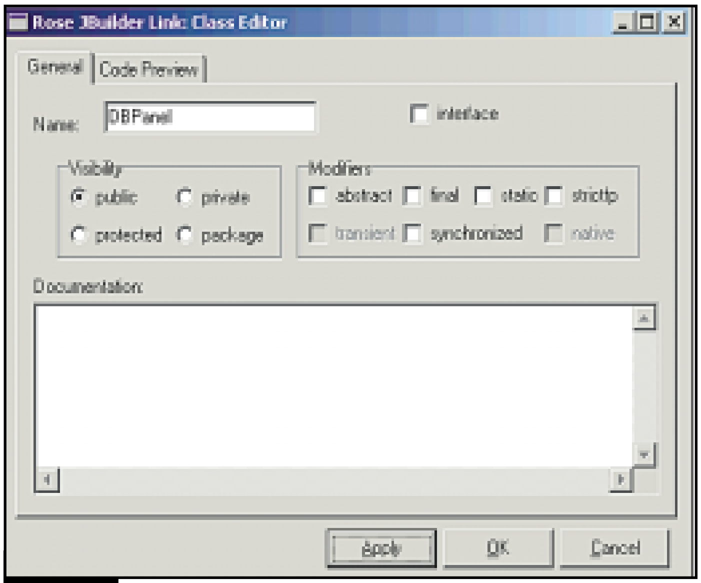
InvalidBalanceException class subclasses the built-in Java Exception class, which is shown by the generalization arrow. You'll notice that all the other classes also use the Exception class because it's thrown by JBuilder's jbInit() method. This diagram would have taken a long time to create by hand, given the complexity of the application.

Rose JBuilder Link will also take class diagrams and generate source code from them. I tested this with a fairly complex set of class diagrams that my company had created for a client. Rose JBuilder Link did a good job of generating the classes and creating stub implementations for them. It will also correct syntactic errors in the model that don't conform to Java rules.

For example, a class description in the model incorrectly (from a Java standpoint) declared an array. Rose JBuilder Link corrected the problem as it generated the code. This goes above and beyond the call of duty! When Rose JBuilder Link generates code, it doesn't place "magic markers" in it. A lot of code generators do place markers in the code that can't be removed by the developer without "breaking" the reverse engineering of the code. Rose JBuilder Link handles this by placing model documentation as code comments rather than brittle "magic markers."

Once you have both a model and a project side by side, you can refresh the views and Rose JBuilder Link will highlight the differences between the model and the code. This is nice in situations where the model and the code have changed simultaneously. Within its user interface, you can see exactly what's changed in each. Once you've decided which "side" (the model or the project) needs to be updated, use the button on the bottom to handle the update for you. All in all, the user interface is intuitive and useful.

Rose JBuilder Link also includes some other tools to make JBuilder and Java development easier. One of the hassles of doing Java object modeling in Rose is the clunky user interface you must use to see attributes of classes, such as scoping and modifiers. Because Rose must support many languages, the language-specific elements are confined to a single customizable dialog


Rose JBuilder Link: Class Editor

General | Code Preview

Name: DBPanel

☐ interface

Visibility
[✓ public] private
protected package

Modifiers
☐ abstract ☐ final ☐ static ☐ strictfp
☐ transient ☐ synchronized ☐ native

Documentation:

FIGURE 3 The class editor makes it easier to control Java-specific characteristics of a class.

box. One of the nice little tools that Rose JBuilder Link includes is an editor that allows you to set these characteristics without going through Rose's constrained dialog. This Class Editor tool is shown in Figure 3.

Rose JBuilder Link includes several other tools that space won't allow me to write about, including tools to help support server-side development such as Enterprise JavaBeans. In general, I think this is an excellent product, and it shows just how far code generation and reverse engineering have come. If you need to use Rose for object modeling (you really should be using something!), Rose JBuilder Link is a welcome time- and sanity-saving addition.

XJB 100

Zucotto Wireless Inc.

XJB 100 is the first Bluetooth protocol stack written in Java. It provides the functionality required for Java technology-based wireless networking using the Bluetooth protocol, and is portable to any platform running a Java Virtual Machine (JVM).

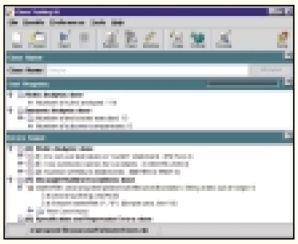
Zucotto Wireless first created a Bluetooth protocol stack in Java to bring wireless networking capabilities to its flagship line of Xpresso Java native processors and supporting software and hardware development kits. While the original Bluetooth stack was configured for a J2ME environment, the licensable XJB 100 can be configured for use on any Java platform, extending Bluetooth functionality to any device running a JVM. XJB 100 works with any Bluetooth baseband that's compatible with the Host Controller Interface (HCI) specification.

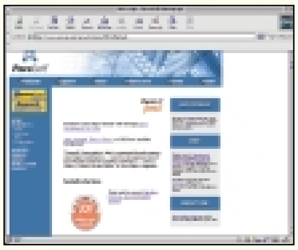
Available: First quarter of 2001
Contact: www.zucotto.com

Jtest for Linux

ParaSoft

Jtest is the first tool of its type available for the Linux platform. The product is a fully integrated automatic class testing tool for Java. It integrates every essential type of Java testing into one tool that automatically performs static analysis, and white-box, black-box, and regression testing. Jtest works on any Java class. Developers can use Jtest as soon as they've constructed and compiled each class of their project.



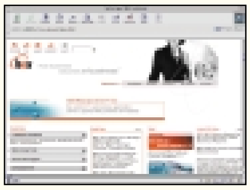


Available: Immediately
Contact: www.parasoft.com

WebLogic Server 6.0

BEA

BEA WebLogic Server 6.0 is one of the industry's most advanced Java application servers for building, running, and future-proofing high-volume, mission-critical e-business applications.



This new release extends the product's core functionality with the latest J2EE innovations, a wide breadth of industrial-strength "mainframe-class" functionality, and manageability benefits across the complete e-business application lifecycle.

BEA WebLogic Server 6.0 now incorporates BEA Tuxedo for improved transaction reliability. It also includes simplified manageability, usability, and installation with a new Web-based management console based on the Java Management Extension (JMX) framework, an integrated message system based on Java Message Service (JMS), and enhanced XML capabilities.

Available: Immediately via download
Contact: www.bea.com

TogetherSoft Acquires Object UK Ltd

(*Raleigh, NC / San Jose, CA / Southampton, England*) – TogetherSoft Corporation has acquired the Southampton-based software company Object UK Ltd, forming a new, wholly owned subsidiary, TogetherSoft UK Limited. Terms of the acquisition were not disclosed.

www.togethersoft.com

Bean-test 3.1 Available from RSW Software

(*Waltham, MA*) – New from RSW Software, a business unit of Empirix, is Bean-test 3.1, the most recent version of its popular EJB testing solution.

Features include multi-EJB load testing, automatic test-case generation, automatic custom objects support, IBM WebSphere 3.5 support, and batch processing.

www.rswsoftware.com

Rational Unveils Rational Suite v.2001

(*Cupertino, CA*) – Rational Software Corporation has introduced Rational Suite version 2001, which includes four

new products and enhancements to the existing 12 products in the Rational Suite family: Rational ClearCase LT, Rational Quality Architect, Rational TestManager, and the Rational Unified Process with Content for the Microsoft Web Solution Platform.

www.rational.com

Softwired Receives Awards from Symbian

(*Zurich, Switzerland*) – iBus//Mobile from Softwired has been awarded the prize for both the Symbian "Best Java Enterprise Application" and "Best Java Application Overall."

iBus//Mobile offers transparent support for wired and wireless networks and protocols, automatic transcoding to adapt content to wireless devices, message synchronization and queuing to support both online and offline operation, and personalization.

www.softwired-inc.com

STFB Inc. Launches Web-Based Accounting System for ASP

(*Pembroke Pines, FL*) – STFB Inc., an Internet software development firm, announces the release of Integral Accounting for JavaScript and ASP, the first off-the-shelf Web-based accounting solution for the application service provider industry.

The system includes a general ledger, accounts receivable, accounts payable, financial reporting, and an e-commerce shopping cart that is fully integrated into the accounting application.

www.stfb.com

Java Community Process Program Election Results Are In!

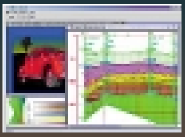
(*Palo Alto, CA*) – The Program Management Office of the Java Community Process (JCP) program, the community process for evolving Java technology, has announced its new executive committee (EC) members. The 30 members, who were voted into office by the Java technology developer community through the JCP program, will guide the development of the Java platform for a three-year term staggered to allow for five of the 15 seats of each Executive Committee to come up for election each year. The new EC members will take office on December 12, 2000.

Java Community Process Elections
Voting begins October 9, 2000

The result of the voting can be seen at http://betrusted.asapware.com/sun_results.adp.

Interactive Network Technologies Releases J/CarnacPro 2.0

(*Houston, TX*) – Interactive Network Technologies, Inc. (INT), a leading developer of high-performance graphics components, has announced availability of J/CarnacPro 2.0, its graphics toolkit based on Java 2D.



The product boasts significant performance enhancements, including new geometry and attribute editors for interactive editing of graphical objects, extensive printing and print preview API, updated tutorials, and improved zooming, caching, scrolling, and resizing.

To register for a 30-day evaluation, visit www.int.com/.

Pramati Releases Studio 2.0

(*Hyderabad, India*) – Pramati Technologies Ltd introduces Pramati Studio 2.0, the latest version of its server-side IDE. Pramati Studio 2.0 features full J2EE development life-cycle support, powerful wizards, JSP-EJB debugger, and a new component-based architecture that enables customers and partners to add new tools via the Studio API.

Developers can download an evaluation copy of Studio 2.0 from Pramati at www. pramati .com.

Oracle Introduces Oracle9*i*

(*Redwood Shores, CA*) – Oracle Corp. has consolidated traditionally separate, highly specialized business intelligence technologies into a single infrastructure for business information – Oracle9*i*. Using the built-in business intelligence capabilities of the Oracle9*i* application server, e-businesses can now make personalized business information available to anyone within the organization using any Internet-enabled device, including mobile phones and PDAs. E-businesses can also now make valuable business information available to partners and customers in a timely fashion.

www.oracle.com

Alcatel and Zucotto Form Alliance

(*San Jose, CA*) – Alcatel and Zucotto Wireless Inc. have announced a strategic alliance that will combine Zucotto's Java native processor with Alcatel's family of Bluetooth products to deliver efficient, optimized balance between high-speed and low-power consumption.

www.alcatel.com

www.zucotto.com

BSDi Releases BSD/OS 4.2

(*Colorado Springs, CO*) – BSDi announces the availability of BSD/OS version 4.2 Internet Server Edition. The latest BSD/OS release supports Java 2 Standard Edition and incorporates the KAME IPv6 and IPsec implementations.

New features include VLAN (802.1Q, 802.3AC), wireless Ethernet 802.11 (Aironet/WaveLAN), new sound drivers, additional RAID support (AMI MegaRAID, Compaq), Linux Application Platform (LAP) support updated for RedHat Linux 6.2, and FAT32 support for accessing the hard disk used by the Windows operating system in dual-boot configurations.

www.bsdi.com

PointBase to Deliver UniSync Technology Equipped with Built-In SyncML

(*Mountain View, CA*) – PointBase, Inc., will deliver its UniSync technology (universal synchronization) equipped with built-in SyncML technology to de- vice manufacturers, Java application developers, Internet companies, and wireless carriers who want to deploy SyncML-compliant products and services as early as Q2, 2001.

This new solution will give users local and remote data synchronization among SyncML-compliant products and services, regardless of the platform or manufacturer.

PointBase will add SyncML technology to selected implementations of its UniSync API, as well as provide professional services to its partners to quickly implement the SyncML specification.
www.pointbase.com

Fresco Embedded Browser by ANT Available

(*Los Angeles, CA*) – ANT Limited has announced that now its Fresco embedded browser runs on the Java platform. This new implementation uses the Java Native Interface (JNI) to interface with Java-based Internet devices, enabling ANT to provide OEMs with a small footprint, fully customizable browser for Java environments. ANT Fresco is written in C, which reduces development times by combining the ease of programming in Java with the speed and performance advantage of a native browser.
www.antlimited.com

Flashline.com Adds CRM and E-Business Components

(*Cleveland, OH*) – Flashline.com Inc. is expanding its Software Component Marketplace with the addition of leading-edge EJB-based components from Compoze Software, Diamelle Technologies, and Ohioedge. The products are designed to accelerate the development of e-business applications.

The company also announces that Adam Wallace has joined the company as chief information officer. He will work closely with the management team to guide the company's technical direction, and will lead all software development efforts to support Flashline's business initiatives across the company's multiple divisions. Wallace was previously CEO at XA.com, a developer of e-commerce software components.
www.flashline.com

SecureByDesign.com Offers New E-Mail Encryption Software

(*Houghton, MI*)– SecureByDesign.com has launched its J7 secure e-mail software for Java developers. The product conforms to the S/MIME standard, and allows developers to digitally sign and encrypt outgoing e-mail messages. It can also parse incoming secure e-mail.
www.securebydesign.com

Virtuas Releases jtagwireless

(*Englewood, CO*) – Virtuas Solutions, Inc., is offering its JSP wireless tag library, jtagwireless, for public download. The library is based on Sun Microsystems' JSP 1.1 specification and the WAP Forum's WML 1.1.

The tags are free, requiring only member registration. To download them, visit www.virtuas.com/downloads.html.

IONA's iPortal App Server Achieves J2EE Certification

(*Waltham, MA*) - IONA Technologies, the Enterprise Portal Company, has released version 1.3 of its iPortal Application Server. The server, which has been certified by Sun Microsystems' Java 2, Enterprise Edition (J2EE) Certification, is a cornerstone of the iPortal Suite, a standards-based integration, development, and Web presentation platform for the creation of e-business applications and enterprise portals.

It includes an EJB 1.1 technology-based container, a J2EE Web container that supports JavaServer Pages technology, servlets, and complete support for other enterprise technology specifications included in the J2EE platform.
www.iona.com

Sun Links Java, XML for Online Business

(*Palo Alto, CA*) – Sun has released APIs that connect Java software with XML, making it easier for software developers to create Web sites for e-business.

One interface, Java API for XML messaging, will allow businesses to send and receive XML messages using a new messaging standard called WebXML. Another interface is an updated Java API for XML processing, which now supports the latest XML standards. The interface integrates Java software with XML parsers. Free test versions are available at www.sun.com.

Sitraka Software Announces Support for StudioJ Customers

(*Toronto, ON*) – Sitraka Software (formerly KL Group) has announced an agreement with Rogue Wave Software to support the company's StudioJ users in adopting its own JClass components. Rogue Wave, which recently announced the retirement of its StudioJ GUI components, will be working with Sitraka to ensure StudioJ customers are able to make a smooth transition to using the JClass GUI components for their ongoing and future Java development needs.

As part of this agreement, Sitraka will offer StudioJ customers already receiving active support a free copy of JClass Chart or JClass LiveTable when they purchase a one-year gold support subscription – a package that entitles them to a year of free upgrades and unlimited support. StudioJ customers who don't have active support will be eligible to purchase JClass products at a discounted price.
www.sitraka.com

Sitraka Software Uses Innovative Pricing in Release of DeployDirector 1.3 E-Mail Encryption Software

(*Toronto, ON*) – Sitraka Software (formerly KL Group) has launched a new generation of DeployDirector, with a new pricing model that allows the enterprise customer to deploy applications more cost-effectively to an expanding workforce. DeployDirector 1.3 also introduces German and French localization and support as well as a free customer evaluation version available online.

The pricing model is based on the number of applications to be deployed rather than the number of "seats" the applications are being deployed to. A customer evaluation version can be downloaded free from www.sitraka.com/software/deploydirector/.

SHOW REPORT

XML DevCon becomes event to come to Silicon

San Jose, CA – XML enthusiasts from 25 countries – programmers, developers, engineers, software architects, system engineers, Web developers, product managers, project leaders, consultants, and educators gathered together at the San Jose DoubleTree Hotel November 12–15, 2000, for four days of technical sessions and a two-day interactive exhibition. Sponsored by SYS-CON Media, *XML-Journal,* and Camelot Communications, XML DevCon 2000's faculty included 33 authors, the pioneers who created markup languages and SQL, a W3C Fellow, ACM Fellow, Seybold Fellow, STC Honorary Fellow, and Turing Award winner.

Unprecedented All-Star Lineup of Speakers

The conference in San Jose was a resounding success. During the time I spent there, I tried to get an idea of how many attendees had also been to the XML DevCon held in New York last summer. As expected, most of them had waited for the show to hit their city. It looks like our idea of taking the "show on the road" is really paying off. Based on the input provided by some of the vendors I interviewed, it seems that most of the attendees were already familiar with the technologies and were seriously looking for tools and frameworks they could use in specific applications. As a representative from one of the vendors put it, "They're all now saying 'Show Me.'" This is not surprising, since the DevCon was held in the Valley, the home of the developer. However, I think this is true for the rest of the U.S., as well as the rest of the world – wherever enterprise-level applications are being developed. XML has definitely come of age.

The conference drew a mixed crowd – from experienced developers to program managers and business development folks.

Welcome Reception Sponsored by *XML-Journal*

The event kicked off with a welcome reception by ***XML-Journal***. Attendees gathered in a casual setting to exchange ideas with fellow XML developers and to network with the industry's leading XML influencers.

Featured Speakers

Industry icons such as Tim Bray, Charles Goldfarb, Don Chamberlin, and Jim Gray spoke at XML DevCon 2000 for the first time.

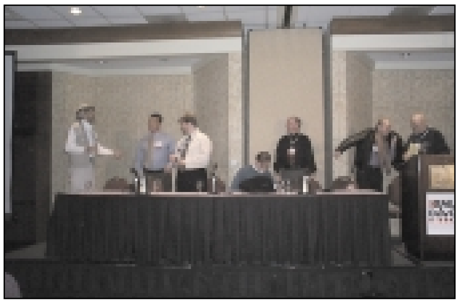
The conference opened with a keynote presentation by Bob Sutor, program director, eBusiness Standards Strategy, IBM Corporation. In his XML DevCon address, "XML and Web Services: Bringing Order to B2B on the Web," Sutor said that the primary mission is to assure the global availability of information through the interoperability of data and workflows. He announced IBM's XML strategy to (1) support standards through open source, (2) enable its entire product line, and (3) build e-business solutions.

Tim Bray's official launch of the Map.net Web site was the buzz of the conference. Using the continent of Antarctica as a visual reference, Bray's company, Antarctica Systems, has constructed a three-dimensional map of the World Wide Web. Built with the company's Visual Net software, the site presents users with a 3D landscape; the relationships between network elements are represented geographically.

Conference Tracks

Six conference tracks assured an interesting session for every time slot from Sunday through Wednesday. The tracks covered applied XML/e-business, Java/ scripting, wireless/messaging, servers/middleware, query/ schema/database, and developer techniques.

The conference delegates were taught by some of the leading XML experts in the industry, including Keith Bigelow, Barbara Bouldin, Kurt Cagle, Russell Castagnaro, Don Chamberlin, Parand T. Darugar, Max Dolgicer, Mohamed El-Mallah, Mary Fernández, Dave Frankel, Daniel P. Gill, Peter Haggar, Seth Hitesh, Molly E. Holzschlag, David S. Linthicum, Brett McLaughlin, Norbert Mikula, Duane Nickull, Ken North, Shelley Powers, Gerry Seidman, Simon St. Laurent, and Mark Volkmann.



largest XML Valley in 2000

XML DevCon FALL 2000

XMLDevCon2000 Show Floor Highlights

The exhibition floor had a good show of vendors and their products. Several of the tools offered by vendors are into their second or third release, indicating a maturing of the XML tool market. Here are some of the tools that garnered a lot of attention:

XML SPY 3.5

Altova previewed XML Spy 3.5, its Windows-based integrated XML document, XML schema, and XSLT stylesheet editor.

Alexander Falk, president of Altova, noted that XML Spy now offers "syntax highlighting, for text-based entry, along with an enhanced grid view providing intelligent entry features like drag and drop, and an integrated table view for repeating elements."

XML AUTHORITY 2.0

TIBCO Extensibility demonstrated a preview of XML Authority 2.0, the latest revision of their schema editor for Windows, UNIX, and (in beta) MacOS X, with extra support for Extensibility's own Schema Adjunct Framework, along with Oracle's iFS and Software AG's Tamino Starter Kit XML.

<XML>TRANSPORT AND <XSL>COMPOSER

Whitehill Technologies demonstrated <xml>Transport and <xsl>Composer, two tools for transforming information into XML and among XML vocabularies.

Keynote speaker Tim Bray

Mark Your Calendar

The exhibition area proved to be a good meeting place for vendors and prospective customers. I talked to several vendors who had gotten very solid leads as a result of the interaction they had with the attendees. Even more interesting was the fact that a few of the vendors mentioned that they had met each other for the first time and were exploring possible business relationships. To me, this signifies two things. One is that the XML market is still fragmented in the sense that there are several independent developments taking place in different companies that could supplement each other and offer more complete XML-based solutions. The other is that there is a need for conferences like XML DevCon and magazines like ***XML-J*** that can help bring interested parties together by offering venues for interaction and helping vendors introduce their technologies to developers and business managers.

SYS-CON Radio

SYS-CON Radio had a booth at the entrance of the exhibit hall and a booth inside the hall. The radio booth was a good location to see the crowd of attendees bustle by, trying to cram as much information as possible in the four days packed with keynotes, presentations, and vendor shows. At the booth we interviewed several vendors to get an idea of their impressions of the show. It was reassuring to see that some of the vendors are addressing serious issues in XML applications such as performance, data processing, and security.

Tune in to **SYS-CON Radio** to hear these interviews at www.xmldevcon2000.com/.

"XML DevCon's technical program and interactive exhibit floor will continue to set the standard in conference education for the software development community," said Ken North, conference chair. Highlights of the event may be found at www.xmldevcon2000.com.

If you did not attend, make plans now for the 2001 events in London (February 21–23), New York City (April 8–11), and San Jose, California (October 29–November 1).

ajit@sys-con.com

Participating Companies at XML DevCon Silicon Valley:

- Accelr8
- Altova-The XML Spy Company
- Arbortext
- Baltimore Technologies
- Be-Bop
- Cape Clear
- Code 360
- Cysive
- Eliad Technologies
- e-Numerate
- Fawcette Publications
- Geek Cruises
- Global Knowledge
- HiT Software
- IAM Consulting
- IBM
- Icon
- InfoGlide
- InfoShark
- Infoteria
- Ingeniux
- Insight
- Interknack.com
- Intershop
- IXIASOFT
- Learning Patterns.Com
- Learning Tree International
- MAILERS Software
- Merant
- Microsoft
- MindGap
- Nanobiz
- NeoCore
- Netfish Technologies
- Novell
- Oasis
- ObjectSpace
- Openlink
- Oracle
- O'Reilly
- Percussion
- Pervasive
- Planet 7 Technologies
- Popkin Software
- Progress SonicMQ
- Py-Bix
- PyBiz
- Radiant Logic
- Radview
- Random Walk
- RogueWave Software
- Seagull
- SearchXML Resources.Com
- Sequoia Software
- SilverStream
- SoftQuad Software
- Software AG
- Sun Microsystems
- SYS-CON Media, Inc.
- Talva Corporation
- TechTarget
- The Breeze Factor
- TIBCO Extensibility
- Westlake
- Whitehill Technologies
- Wrox Press
- XAware
- XML Global
- XML Solutions
- XYZFind
- Zkey
- Zot

Two Kinds of Java Engineers

You need to understand Java, not just know it

WRITTEN BY
BILL BALOGLU &
BILLY PALMIERI

Java Jobs is our new column. Each month we'll focus on different aspects of working with Java – in-demand skills, up-and-coming technologies, hot cities, salaries, rates, and other tips to help you plan and further your career in the Java marketplace. Whether you're a Java developer or manager, you'll find this column full of useful information and tips. We'd also like to hear from you about related topics that you'd like us to cover as part of this column. Please send your suggestions to jdjcolumn@objectfocus.com.

The intense demand for highly qualified engineers has not only produced serious challenges for hiring managers but also two kinds of Java engineers.

At our firm we provide skilled engineers to fill our clients' needs. This means reviewing resumes, screening, and qualifying candidates on a daily basis. The two basic types of engineers we come across most often are: those who *know* Java and those who *understand* Java.

Engineers who know Java may have a strong background in database-centric applications with skills in SQL, Oracle, or PowerBuilder. They may have done a lot of front-end or UI development.

Aware of the strong demand (and the high rates and salaries paid) for those with Java skills, these engineers will typically pick up a Java book, take a Java course, or "teach themselves Java."

"I know Java, it's just another programming language," is a comment we hear all the time from engineers with about six months experience "working with Java."

The know-Java engineers enter the job market asking for $100 an hour or $150K a year, and staffing agencies are often quick to place them on a contract or full-time position with a major client.

One thing is certain in the fast-paced, high-stakes, high-tech world. Hiring managers don't put out a call for engineers unless they needed one two months ago. They need an expert who can hit the ground running, solve their problems, and get the job done right.

The know-Java engineer often gets on a mission-critical project and (having oversold his or her Java skills) becomes quickly overwhelmed and can't do the job. This all-too-familiar scenario is a sure-fire career killer.

The know-Java engineer has now burned bridges with the hiring manager (whose precious time and budget have been wasted), the client company (who tracks these incidents in a database), the recruiting agency, and other co-workers on the project.

A series of short-term contracts (or worse, short-term, full-time jobs) listed on a resume is a serious red flag for anyone who's hiring, and it's hard to get a good reference from a manager you burned.

While it may take only a few weeks to learn the syntax of Java, it takes a lot of in-depth experience to understand it and know how to deal with critical issues and avoid pitfalls that are exclusive to Java.

Engineers who understand Java typically have an extensive OOP background, most likely with C++, and a strong grasp of object-oriented methodologies, development techniques, and design patterns. They have Java programming experience (typically more than two years) and have worked with EJB or J2EE among other skills, such as CORBA, distributed network computing, and multithreaded programming.

As a mid- to senior-level engineer, you're not being paid a high hourly rate or salary to simply know the language of Java, you're being paid for your expertise and experience in avoiding and fixing critical problems that come up in the development process.

The understand-Java engineer has experience in developing scalable, robust, high-performance, and large-scale applications. He or she knows what can and can't be done with Java. If there's a problem, he or she knows where to look and how to fix it swiftly and efficiently.

Hiring managers, faced with aggressive deadlines and budget constraints, often make the mistake of trying to cut corners by bringing in engineers (from rent-a-programmer shops at an "apparent" low cost) with minimal local experience, technically and work-culture wise.

Too often these managers end up paying for a know-Java engineer who's read the book and knows the syntax of the Java language, but is not experienced enough to look at the bigger picture. That relatively low contract rate of $75 an hour doesn't seem like such a bargain when after two months the manager has wasted $24,000 on an engineer who couldn't get the job done.

But what if I'm a know- Java engineer who wants to become an understand-Java engineer? How do I gain the experience I need without overselling myself and getting in over my head?

There are a few smart steps you can take to make the transition from junior to mid or senior Java engineer:

- Don't fool yourself. Realize what skills you have and what you need to learn.
- Look for a full-time position in a Java environment of a large company and plan to spend about two years there.
- A large company is much more likely to hire someone who's a bit junior and invest in training you.
- Bring all the skills and value you have to the company, and get all the Java development and object-oriented programming experience you can from them.
- Or try to find a long-term contract (six months or more) in a Java development environment offering your services at a reduced hourly rate.
- Make yourself a great value proposition for the company – you'll be getting an even greater long-term value from the experience you'll gain.
- Remember that your goal is not just to know, but to understand Java.

billb@objectfocus.com

billp@objectfocus.com

AUTHOR BIOS
Bill Baloglu is a principle at Object Focus (www.ObjectFocus.com), a Java staffing firm in the Silicon Valley. Prior to ObjectFocus, Bill was a software engineer for 16 years. He has extensive OO experience and has held software development and senior technical management positions at several Silicon Valley firms.

Billy Palmieri is a seasoned staffing industry executive and a principle of ObjectFocus. Prior to ObjectFocus, he was at Renaissance Worldwide, a multimillion dollar, global IT consulting firm, where he held several senior management positions in the firm's Silicon Valley operations.

SYS-CON INTERACTIVE

The World's Leading Java Resource

BPA INTERNATIONAL

JAVA DEVELOPER'S JOURNAL .com

SUBSCRIBE | JAVA FORUM | DIGITAL EDITION | SOURCE CODE

ADVERTISING | PRODUCT REVIEWS | SEARCH

FREE JDJ ARCHIVES

Free access to Java Developer's Journal Archives, every article published since our premier issue!

REGISTER HERE

JDJSTORE.COM
JDJBOOK STORE
BUYER'S GUIDE
READER SERVICE
FEATURE STORIES
EDITORIAL COLUMNS
JAVA JOBS
AUTHOR GUIDELINES
EDITORIAL BOARD
CUSTOMER SERVICE

Search JDJ

January 2001

What's Online This Month...

JavaDevelopersJournal.com

www.javadevelopersjournal.com is your source for industry events and happenings. Check in every day for up-to-the-minute news and developments, and be the first to know what's going on in the industry.

Participate in our daily Live Poll and let your opinion be heard.

JavaDevelopersJournal.com Developer Forums

Join our new Java mailing list community. You and other IT professionals,

industry gurus, and *Java Developer's Journal* writers can engage in Java discussions, ask technical questions, talk to Java vendors, find Java jobs, and more. Voice your opinions and assessments on topical issues, or hear what others have to say. Monitor the pulse of the Java industry !

Digital Edition

Don't have your print edition on hand? Can't wait for the next issue to arrive in the mail? Our digital edition is just what you need. As long as you have your computer with you, you can read *Java Developer's Journal* anytime, anywhere. Looking to research a specific topic? Search our archives – we've got every article that's been published since our premier issue!

JDJ Readers' Choice Awards

Vote for your favorite Java software, books, and services in our annual *JDJ* Readers' Choice Awards, January 10

through May 30, 2001. Winners will be announced at JavaOne 2001 and presented at the International Conference for Java Technology – Fall Conference.

International Conference for Java Development Spring 2001

The International Conference for Java Development, presented by ***Java Developer's Journal,*** will take place February 26–March 2, at the Marriott Marquis Hotel in New York City's Times Square. Click here to see what sessions are scheduled, who the keynote speakers are, and who's exhibiting. Register now for the largest Java software developer event coming to the East Coast in 2001!