No. 1 *i*-Technology Magazine in the World

JDJ

JDJ.SYS-CON.COM VOL.12 ISSUE:8

Multi-Core & Massively Parallel Processors

Coming soon to a theater near you...

RETAILERS PLEASE DISPLAY
UNTIL OCTOBER 31, 2007
$5.99US $6.99CAN
09>
0 09281 01751 6

PLUS...

- Introduction to Maven
- Developing Rich Internet Applications Using Swing

Altova® UModel® 2007 – The starting point for successful software development.

ALTOVA® umodel® 2007

Visualize works of software art

Draw on UModel® 2007, and picture better programs based on UML™.

Spied in UModel 2007 Release 3:

- Support for all 13 UML 2.1 diagram types
- Modeling of XML Schema in UML with code engineering capabilities
- Reverse engineering of Java and C# binary files
- Automated documentation generation in HTML, Word, or RTF

Altova® UModel 2007, the intriguing new force in the software design space, is the simple, cost-effective way to draw on UML. Use it to interpret or create your software architecture. Decode Java or C# programs into clear, accurate UML 2 diagrams, or outline applications and generate code from your plans. With all 13 diagram types, interoperability via XMI 2.1, an artful user interface, and more, UModel makes visual software design practical for programmers and project managers everywhere. Take the mystery out of UML!

Download UModel 2007 today:
www.altova.com

ALTOVA®

JAVA DEVELOPER'S JOURNAL

Editorial Board
Java EE Editor: **Yakov Fain**
Desktop Java Editor: **Joe Winchester**
Eclipse Editor: **Bill Dudney**
Enterprise Editor: **Ajit Sagar**
Java ME Editor: **Michael Yuan**
Back Page Editor: **Jason Bell**
Contributing Editor: **Calvin Austin**
Contributing Editor: **Rick Hightower**
Contributing Editor: **Tilak Mitra**
Founding Editor: **Sean Rhody**

Production
Associate Art Director: **Tami Lima**
Executive Editor: **Nancy Valentine**
Research Editor: **Bahadir Karuv, PhD**

To submit a proposal for an article, go to
http://jdj.sys-con.com/main/proposal.htm

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

888 303-5282
201 802-3012
subscribe@sys-con.com

Cover Price: $5.99/issue. Domestic: $69.99/yr. (12 Issues) Canada/Mexico: $99.99/yr. Overseas: $99.99/yr. (U.S. Banks or Money Orders) Back Issues: $10/ea. International $15/ea.

Editorial Offices
SYS-CON Media, 577 Chestnut Ridge Rd., Woodcliff Lake, NJ 07677
Telephone: 201 802-3000 Fax: 201 782-9638

Java Developer's Journal (ISSN#1087-6944) is published monthly (12 times a year) for $69.99 by SYS-CON Publications, Inc., 577 Chestnut Ridge Road, Woodcliff Lake, NJ 07677. Periodicals postage rates are paid at Woodcliff Lake, NJ 07677 and additional mailing offices. Postmaster: Send address changes to: Java Developer's Journal, SYS-CON Publications, Inc., 577 Chestnut Ridge Road, Woodcliff Lake, NJ 07677.

Worldwide Newsstand Distribution
Curtis Circulation Company, New Milford, NJ
For List Rental Information:
Kevin Collopy: 845 731-2684, kevin.collopy@edithroman.com
Frank Cipolla: 845 731-3832, frank.cipolla@epostdirect.com

Newsstand Distribution Consultant
Brian J. Gregory/Gregory Associates/W.R.D.S.
732 607-9941, BJGAssociates@cs.com

Doubtful Diagrams and Far Out Figures

Joe Winchester

In a recent presentation I attended, the speaker warmed up with a couple of bulleted lists that outlined the agenda of the session before moving onto his third slide, which was clearly the result of many days of work stitching together PowerPoint glyphs and figures in a sort of three dimensional loop that attempted to show the progression of software APIs around the evolution of networked computing. It was utterly baffling and the more I stared at it, the more I felt I was looking at some kind of latter day Escher drawing. I gazed around the room and saw most of the other attendees on their laptops, distracted by their chat or e-mail clients; however, among those of us who weren't using the session as down time in our schedule, I saw no one question the meaning of the figures or the information it was attempting to convey. The speaker was extremely proud of it, however, and lingered on the slide for about 20 minutes as he waved his arms and spoke of REST, ATOM, Ruby, and a plethora of other acronyms that apparently were all part of the Web 2.0 solar system.

This editorial isn't a diatribe against Web 2.0, far from it; it's against the vast overuse of meaningless diagrams that presenters use to confuse, impress, and befuddle their audience. Part of this, I believe, stems from the speaker's insecurity in understanding the subject matter, and seemingly complex figures are used as a sort of crutch to fall back onto to mask the lack of content. A lot of this probably dates back to high school math or science classes, where the apprentice PowerPoint maestro had long since lost the plot of what the teacher was discussing, but noticed that as some of the problems got harder, diagrams were introduced to help illustrate the ideas being taught.

To the confused yet cunning child, the association was clear: if the teacher could impress the audience with impressive-looking figures, then in later life when the student was given the task of being the presenter, if they just skipped the content part and showed figures with boxes and lines between them, then they too would assume the role of authority that their teacher commanded. It's sort of based on the bigger fool theory, which states that in life you don't need to be smart, you just need to be smarter than the next guy you're trying to fool. Rather like the apocryphal tale of two hunters running from an angry bear and as one of them stops to put on his sneakers, the other remarks how he'll never outrun the bear; the former replies that his intention is to merely outrun his partner. In presentations if the figure is meaningless and has no content, and if the goal is to impress the audience who are too nervous to challenge the emperor's naked ignorance, the speaker has outrun his pupils and achieved the respect originally given to the teacher.

Wikipedia defines a diagram as a "simplified and structured visual representation of concepts and ideas to visualize and clarify the topic." This definition should apply to software as well as any other discipline, and as a profession we need to do more to challenge ludicrous and meaningless figures that add no value, and strive more toward simplicity so that topics and ideas are made more consumable and understood, and the peddlers of complexity are run out of town with both their sneakers firmly tied.

Joe Winchester is a software developer working on WebSphere development tools for IBM in Hursley, UK.

joewinchester@sys-con.com

WebChart™ - 3D Overlay Donut Chart

NetAdvantage® for JSF 2007 Volume 1

WebGrid™ Export to CSV Format

Consistent Multi-Platform User Experience

WebCharting – Add high impact 2D/3D visualizations with over 20 different chart types and views

Improved Data Reporting – Easily export data within a grid to any application that supports the import of CSV such as spreadsheets and databases

Application Performance – Leverage our AJAX framework to turbo-charge your Web applications

Improve Productivity – Quickly and easily apply professional polish to your applications, using built-in professional styles, or customize to achieve your own

learn more: infragistics.com/jsf

Infragistics®
Powering The Presentation Layer

Infragistics Sales - 800 231 8588
Infragistics Europe Sales - +44 (0) 800 298 9055

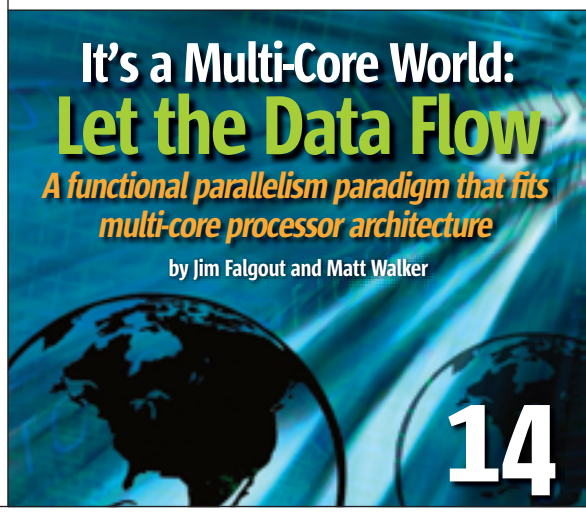
WINDOWS FORMS grids scheduling charting toolbars navigation
ASP.NET menus
WPF listbars
JSF trees tabs explorer bars editors

AUGUST 2007 VOLUME:12 ISSUE:8

JDJ contents

JDJ Cover Story

Feature



JDJ (ISSN#1087-6944) is published monthly (12 times a year) for $69.99 by SYS-CON Publications, Inc., 577 Chestnut Ridge Road, Woodcliff Lake, NJ 07677. Periodicals postage rates are paid at Woodcliff Lake, NJ 07677 and additional mailing offices. Postmaster: Send address changes to: JDJ, SYS-CON Publications, Inc., 577 Chestnut Ridge Road, Woodcliff Lake, NJ 07677.

Industry News

HI and Esmertec K.K. Partner to Integrate Mascot-Capsule on Esmertec's Jbed JVM Platform

(*Duebendorf-Zurich, Switzerland*) – Esmertec K.K., a reseller of Esmertec AG's Java solutions, and HI Corporation have announced that HI's 3D rendering engine MascotCapsule V3 and V4 have been integrated on Esmertec's Jbed Java platform by Esmertec K.K. The integration provides game developers with a powerful platform for creating and deploying high-quality 3D games on a broad range of mobile handsets, ensuring a richer gaming experience for consumers, and generating additional revenue for operators, game publishers, and handset manufacturers.

Jbed is Esmertec AG's best-in-class Java Virtual Machine platform for mobile handsets recognized by major handset manufacturers worldwide and already shipped in over 120 million mobile handsets. MascotCapsule is a de-facto standard 3D rendering engine in Japan and abroad designed for creating console-quality, high-speed 3D games on mass-market mobile handsets. The MascotCapsule series overall has been shipped with over 230 million handsets worldwide as of the end of December 2006. V3 and its Java API, com.mascotcapsule, have been used to develop mobile games worldwide. The latest release V4 is fully compliant with the Mobile 3D Graphics API for Java ME (JSR184) to exploit the power of hardware acceleration.

www.esmertec.com
www.hicorp.co.jp

Xcalia Introduces XIC 5.1 with Data Access Service

(*Palo Alto, CA / Paris*) – Xcalia, a provider of dynamic integration software, has announced Xcalia Intermediation Core (XIC) version 5.1 featuring Data Access Service (DAS). Companies can now utilize a single solution to enable access to heterogeneous data sources based on Java, .NET and Web services. Xcalia DAS also leverages the SDO and DAS standards (as specified by the OASIS and Open SOA organizations), ensuring that customers can architect their solutions in synch with evolving industry standards initiatives.

XIC Data Access Service (DAS) – XIC 5.1 now exposes a generic Web services CRUD interface to allow for more interoperability and easier access for clients using any technology. XIC can now be used by any Web service consumer programming language like BPEL or workflow engines. The XIC DAS is deployed in a Web services container and XIC 5.1 provides client APIs for both Java and .NET platforms. The XIC DAS acts as an SDO server. It publishes a logical business model that is manipulated by client applications through SDO. The XIC DAS defines the mapping between the logical business model and the supported data sources. That mapping is defined once in a single place and is shared by all client applications.
www.xcalia.com

Java Synergy: Perst Integrates with Apache Lucene

(*Issaquah, WA*) – McObject has announced that the newest release of the Perst open source, object-oriented pure Java embedded database system supports integration with the widely used open source, all-Java Apache Lucene information retrieval library, to provide text indexing and searching within Perst databases.

In McObject's new Perst for Java version 2.71, Perst functions encapsulate the Lucene indexing and searching APIs. Developers can designate fields within Perst object classes as full-text searchable, so that Lucene automatically adds these fields' contents to its index, and provides the ability to search this indexed material using information retrieval library features, such as support for single- and multi-term keywords, wildcards, proximity queries, phrase queries and relevance ranking. While existing functions in Perst and other databases support a degree of text searching, Lucene provides a much wider range of capabilities, available "out of the box" and proven in thousands of deployments.

www.mcobject.com/perst
http://lucene.apache.org/

BEA Delivers WebLogic Server Virtual Edition

(*San Jose, CA*) – BEA Systems, Inc., a provider of enterprise infrastructure software, has announced the general availability of WebLogic Server Virtual Edition, a Java application server packaged into a middleware appliance optimized for virtualized environments. The product is designed to help reduce total cost of ownership (TCO) and deployment complexity, and marks the first major milestone in execution of BEA's virtualization strategy and product roadmap announced in December 2006.

BEA WebLogic Server Virtual Edition combines the strength of WebLogic Server with BEA's LiquidVM, a Java Virtual Machine (JVM) that can help Java applications run more efficiently on virtualized hardware. BEA's approach to virtualization is designed to eliminate redundant and unused functionality in the software stack, helping to enable applications to run directly on a hypervisor. This middleware appliance design, which is a lightweight software packaging aimed at minimizing configuration and maximizing flexibility, can help achieve greater hardware utilization. The streamlined deployment afforded by this approach can help reduce operating costs and help enable the agile reprovisioning of computer resources to meet the requirements of today's dynamic SOA services and extreme transaction processing (XTP) applications. www.bea.com

SYS-CON MEDIA

President and CEO:
Fuat Kircaali *fuat@sys-con.com*

President and COO:
Carmen Gonzalez *carmen@sys-con.com*

Group Publisher:
Roger Strukhoff *roger@sys-con.com*

Advertising

Advertising Sales Director:
Megan Mussa *megan@sys-con.com*

Associate Sales Manager:
Corinna Melcon *corinna@sys-con.com*

Events

Events Manager:
Lauren Orsi *lauren@sys-con.com*

Events Associate:
Sharmonique Shade *sharmonique@sys-con.com*

Editorial

Executive Editor:
Nancy Valentine *nancy@sys-con.com*

Production

Lead Designer:
Tami Lima *tami@sys-con.com*

Associate Art Directors:
Abraham Addo *abraham@sys-con.com*
Louis F. Cuffari *louis@sys-con.com*

Web Services

Vice President, Information Systems:
Bruno Decaudin *bruno@sys-con.com*

Information Systems Consultant:
Robert Diamond *robert@sys-con.com*

Web Designers:
Stephen Kilmurray *stephen@sys-con.com*
Richard Walter *richard@sys-con.com*

Accounting

Financial Analyst:
Joan LaRose *joan@sys-con.com*

Accounts Payable:
Betty White *betty@sys-con.com*

Customer Relations

Circulation Service Coordinator:
Edna Earle Russell *edna@sys-con.com*
Alicia Nolan *alicia@sys-con.com*

SAP® TECHED '07:
ENTERPRISE SOA—PUT THE POWER TO WORK.

Knowledge, Networking, Expertise.

SAP® TechEd '07 will deliver the practical knowledge you need to implement a comprehensive enterprise SOA strategy. You'll learn about the latest SAP NetWeaver® platform technologies and SAP offerings like Duet™ software, SAP NetWeaver Master Data Management, SAP xApp™ Analytics, and SAP NetWeaver Composition Environment featuring Java EE 5 and ABAP™ through instructor-led hands-on workshops and in-depth lectures. Whether you're a beginner or an expert, SAP TechEd '07 offers you a unique opportunity to network with your technology peers – and to talk directly to experts from SAP and its certified partners.

October 1-5 – Las Vegas
October 17-19 – Munich
November 6-7 – Shanghai
November 28-30 – Bangalore

For more information,
go to www.sapteched.com

07 SAP TECHED

Multi-Core and Massively Parallel Processors

Coming soon to a theater near you...

by J. Stan Cox, Bob Blainey, and Vijay Saraswat

As software developers we have enjoyed a long trend of consistent performance improvement from processor technology. In fact, for the last 20 years processor performance has consistently doubled about every two years or so. What would happen in a world where these performance improvements suddenly slowed dramatically or even stopped? Could we continue to build bigger and heavier, feature-rich software? Would it be time to pack up our compilers and go home?

The truth is, **single threaded performance** improvement is likely to see a significant slowdown over the next one to three years. In some cases, single-thread performance may even drop. The long and sustained climb will slow dramatically. We call the cause behind this trend the **CLIP** level.

- ***C* – Clock** frequency increases have hit a thermal wall
- ***L* – Latency** of processor-to-memory requests continues as a key performance bottleneck
- ***IP* – Instruction-level Parallelism** is already fully exploited by current processor and compiler technologies.

To overcome these challenges the industry is looking to multi-core and multithreaded processor designs to continue the performance improvement trend. These designs don't look to improve the performance of single threads of execution, but instead to run many and sometimes massive numbers of threads in parallel. Wait just a minute though. Is concurrent programming that easy? Hasn't it been tried before?

This article will dive deeper into the current issues challenging processor performance improvement and include a high-level overview of the key microprocessor players: Intel, AMD, Sun, and IBM. Finally, we'll take a deep dive into the challenges, opportunities, and technologies available to Java programmers to take advantage of concurrent programming to leverage these new processor technologies. If you're not programming in parallel today, you will be soon.

Multi-Core Mania

Increases in processor clock frequency are slowing and in many cases are being decreased to reduce power consumption. One trend continues though. The industry continues to shrink the size of transistors, doubling the number of transistors on a chip about every two years or so. In 2007 most major chip manufacturers will begin the shift from a 60nm to a 45nm process. This will yield transistors about 1/2000th the width of a human hair! To provide a relative perspective, a silicon atom itself is about 1/4nm. Obviously continuing to halve the size of transistors will also reach a limit in the not too distant future. But that's a topic for another paper.

So, how will the industry use this new transistor budget to improve processor performance? Techniques such as superscalar execution, pipelining, and speculative processing with branch prediction have added significant complexity to microprocessor designs, but have also been successful at improving performance. Unfortunately, the latency to memory on cache misses and the high frequency of branches in most workloads is proving to be a limiting factor. Building ever-larger caches is one way to mitigate the memory latency problem but as cache size exceeds common working set size, there are rapidly diminishing returns for investing transistors in cache memory.

Instead, the industry is moving toward multi-core, multithreading, and specialization. Instead of improving the performance of a single thread on a single core, the transistor budget is being used to add multiple cores to a single chip. Further, in many cases each core is capable of running multiple threads to hide memory latency. When one thread is blocked by a long latency event, such as a cache miss, the processor simply switches to another thread to execute. Also, many chip designs now include special-purpose processing units that make effective use of transistors for specific tasks such as cryptography.

Taking a closer look at the processors themselves, the IBM Power is distinguished as being the first to introduce multiple cores on a chip in the Power 4 design in 2001. IBM recently introduced the Power 6 processor, which combines two high-performance cores on a chip with each core supporting two-way multithreading. Besides providing multiple cores, the Power 6 also achieved an amazing 4.7GHz clock rate showing that IBM remains serious about single-thread performance while keeping pace with the industry on multi-core. As Power 6 is destined to be included in high-end servers, IBM has also focused heavily on RAS (reliability, availability, serviceability) and virtualization.

In the x86 architecture camp, rivals AMD and Intel have both recently introduced multi-core processors. In 2006, Intel introduced chips with two cores while chips with four cores, based on 45nm technology, shouldappear this year. As part of the move to multi-core, Intel removed support for its version of multithreading known as "hyperthreading," although multithreading is

J. Stan Cox is a senior engineer with IBM's WebSphere Application Server performance group. In this role, he has worked to improve WebSphere application performance for J2EE, Web 2.0, Web Services, and XML . His current focus is WebSphere multi-core and parallel foundation performance. Stan holds a BS from Appalachian State University (1990) and an MS in computer science from Clemson University (1992).

stancox@us.ibm.com

expected to return in future designs. Not to be outdone, AMD later this year, will introduce its first four-core chip known as Barcelona. Both Intel and AMD continue to focus on single-thread performance as well, each introducing new innovations in instruction-level parallelism and caches. One key difference in their designs is the memory bus architecture. Intel is continuing with its symmetric front-side bus architecture. AMD, on the other hand, has introduced a NUMA-based design based on the open Hypertransport technology in hopes of alleviating the memory bus bottleneck.

Sun has adopted a more radical departure in design from prior generations of SPARC. At the end of 2005, Sun released the UltraSPARC T1 or Niagara processor. Niagara includes up to eight cores, significantly more than competing server processors. Sun was able to squeeze eight cores on the chip by shifting focus away from the best achievable single-thread performance toward high chip-level throughput. Niagara cores run at a relatively low clock rate and don't support out-of-order processing, branch prediction, or many other common ILP optimizations. Instead they depend on four-way multithreading to tolerate long waits for memory. The goal is to achieve high overall throughput through application concurrency. However, applications with lower concurrency may run significantly slower on Niagara relative to the other processors described here.

At this point, all of the key players are producing chips with multiple cores but diverging in core design, memory nest, and other important aspects. The key to success for processor designers over the next few years will be in the innovative use of their transistor budget. Architects will make strategic tradeoffs between single-thread performance, massive concurrency, cache sizes, power consumption, and specialized processing units. The companies that make tradeoffs in the most innovative ways to meet the demands of the market should emerge as the winners.

Parallel Programming

As a developer, it will be important for you to learn the skills necessary to develop applications that can run with high performance on these increasingly parallel processors. Since single-thread performance isn't likely to improve at historical rates, the developer will have to look to concurrency to improve performance for a given task. The goal of parallel programming is to reduce the time of a task by dividing it into a set of subtasks that can be processed concurrently. While this may seem simple enough, experience shows that developing correct and effective parallel programs is surprisingly difficult. To utilize parallelism in hardware effectively, software tasks must be decomposed into subtasks, code must be written to coordinate the subtasks and work must be balanced as much as possible. Still sound easy? Read on.

As you get started with parallel programming, the first rule to become familiar with is Amdahl's Law. Amdahl's Law says that speeding up your program is limited by the part that's not running in parallel. For example, if a profile reveals that 20% of the time is spent in code that can only run sequentially on one processor, then the best speed increase you can possibly get, even with perfect parallelization of the rest of your program is 5x, no matter how many processors you throw at it. Load imbalance is a similar problem. If you've divided your code into N subtasks, the time taken to execute them is not 1/N. Rather the time taken is the maximum of the execution times of the subtasks.

If getting your code divided into subtasks and ensuring that work is well balanced sounded hard, then let us introduce you to the coordination problem. Unfortunately, very few programs can be parallelized so simply. The reason is that those subtasks are likely to want to operate on the same data and some of the subtasks may have to wait for others to do their thing before proceeding. It's okay if two subtasks want to read the same memory location in parallel, but if one of them wants to write to the location, you've got trouble because you can't predict which subtask will get to it first.

For example, operations to insert and remove objects from a linked list must be executed so that updates to the data structures happen sequentially and don't corrupt each other. An incorrect ordering of accesses to a memory location is called a data race and it can be one of the most difficult bugs to find because your code might behave differently on each run and might even change once you decide to start debugging. To deal with this problem, most programming environments include mechanisms to ensure that a subtask has exclusive access to specific memory, commonly called **locks**. Unfortunately locks bring their own unique problems when multiple subtasks compete for access and, if used indiscriminately, can reverse all of your hard work in parallelizing your code by making subtasks wait too often or too long for exclusive access to shared memory.

Parallel Programming in Java

Fortunately for Java programmers, the language was designed from the beginning with concurrency in mind. Java includes support for threads that can be used to run parts of your code in parallel and "monitors," which are special kinds of locks acquired using the **synchronized** keyword. The java.util.concurrent package also makes managing threads much easier, provides a fast HashMap for parallel programs and "blocking queues" that can be used to pass messages efficiently between threads. If you're a J2EE developer, you're even more fortunate because J2EE application servers such as IBM WebSphere automatically manage parallelism for you. For example, multiple simultaneous requests to a Web site can be processed in parallel with multiple threads managed by the application server and running on multiple cores.

Creating a new thread in Java is as simple as extending the java.lang.Thread class and overriding the **run** method. Another approach is to instantiate a new instance of the Thread class providing an object that implements Runnable. See Listing 1 for a simple example of creating and running threads.

The java.util.concurrent package provides an alternate way to run code in parallel that takes away some of the burden of managing threads directly. Listing 2 shows an example making use of the ExecutorService API. While the code appears somewhat more complex than the first example, one key difference is that the number of threads executing isn't hard-coded into the application, only the amount of work done in each "task."

You may notice that the two samples we've looked at so far generate output that is a random interleaving of the words "tic" and "toc" and that the interleaving changes on each execution. That happens because the threads execute essentially without regard to what's happening in other threads[‡]. Now let's look at how Java helps you coordinate multiple concurrently executing threads. The primary mechanism used to coordinate access to shared data in Java is a **monitor**. In object-oriented programming, the class is a natural protection boundary for private instance data. So in

Bob Blainey is a Distinguished Engineer in the IBM Software Group, responsible for the technical roadmap for software in the era of multi-core and related next-generation systems innovations. Bob is an expert in programming languages and compilers having spent much of his career at IBM driving ever-greater performance and parallelism through program analyses and transformations. Immediately prior to his current position, Bob was CTO for Java at IBM. He is a member of the IBM Academy of Technology, an IBM Master Inventor, and, most impressive of all, manages to remain sane with two pre-teen daughters in the house.

blainey@ca.ibm.com

Java, every object is assigned a unique monitor. Methods declared using the **synchronized** keyword automatically enter the monitor as they get called and exit the monitor on returning. Only one thread can be inside a monitor at any one time, which means that if instance data is only accessed inside synchronized methods, then a data race can't occur. Listing 3 shows an example where multiple threads are updating a common counter value using a monitor to ensure that there's no data race between any two threads.

In some cases, use of monitors can incur too much overhead and simpler alternatives would suffice. For example, if hundreds of threads are involved in the counter example in Listing 3, performance can be dominated by the time taken to enter and exit the monitor rather than doing useful work. The java.util.concurrent.atomic package provides a few lightweight alternatives to monitors for safely updating shared locations in such busy situations. For example, in Listing 4, an AtomicInteger object is used to safely increment a shared counter without using synchronization.

In some cases, threads will want to wait (or **block**) for a particular condition before proceeding. For example, a thread operating on data in a stack will need to wait for another thread to add an entry when the stack is empty. One way to do that would be to have the thread repeatedly check the stack size. Java provides an easier and more efficient way to do this, however. The consuming thread can call the **wait** method on the object and, when another thread adds an entry, it can call the **notify** method which will wake up an arbitrary waiting thread. Listing 5 shows the use of a monitor and the use of wait and notify to operate on a simple stack of integers.

Java offers many more useful features to help you in your parallel programming tasks. We encourage you to explore them and learn more about parallel programming in Java through the excellent resources listed at the end of this article.

What Does the Future Hold?

Even with all of this native support in Java, parallel programming can still be very difficult. First of all, Java provides the means to parallelize an application but does nothing to help you design a parallel program in the first place. Furthermore, even when you have a good parallel design, there are many challenges in achieving good performance and avoiding concurrency bugs. One common problem is for synchronization to cause an excessive number of threads to block. In the worst case, all threads can block leading to deadlock. Another more devastating problem is a race condition that can lead to data corruption. These kinds of problems can bring the entire application down with a memory fault or worse can intermittently produce incorrect results. These and other similar problems often go undetected in development environments, showing up for the first time when under stress in a production environment. Further, these problems can be difficult to diagnose and debug, leading to delays in providing fixes. While there are some good tools available to help with problem determination, the debugging problem remains vexing and standard techniques such as adding print statements may actually make the problem disappear or move!

X10

Of interest to Java developers is the recent announcement by IBM and several academic researchers of a new programming language called X10**.** X10 is a set of extensions to Java providing higher-level constructs specifically for parallel application development. The X10 programming environment is now an open source project at http://x10.sourceforge.net/.

The goals of X10 include managing both concurrency and the distribution of data and providing constructs to greatly simplify the task of concurrent programming. A central concept in X10 is the notion of a place. A place is an abstraction for a collection of related data and activities that operate on that data. A computation may have many places. Places serve as units of distribution – for instance, different places may be located at different nodes of a cluster. An object is created in one place and lives in that place throughout its lifetime. However, all places in a computation are part of the same address space. That is, an object located in one place may contain references to objects located at another place.

Objects are operated on by activities. Activities are much like threads in Java, except that they may be very lightweight – for instance, an activity may execute only a few instructions in its lifetime. An activity may read and write variables, invoke methods, execute control statements, catch and throw exceptions – in short, perform the actions that any Java thread can perform. X10 makes it very easy for a programmer to write code that creates a new activity: the statement **async S** specifies that the statement S is to be executed in its own separate task, which executes in parallel. Listing 6 shows that achieving the parallel Java tasks shown in Listing 1 is quite simple in X10.

Along with spawning activities, X10 supports the notion of joining activities, that is, determining when a collection of activities has terminated. The statement **finish S** specifies that statement S is to be executed and, if during the execution of S any activities are created, these activities must terminate before any following statement begins executing. Thus a programmer may use **finish** to specify an order on activities.

Unlike Java, X10 doesn't support locks. Instead, X10 provides a very simple construct for the programmer to specify atomicity of execution. The statement **atomic S** is executed as if in a single step (with all other activities frozen). Listing 7 shows that achieving the atomic increment shown in Listing 3 is also easy in X10.

The wait/notify behavior shown in Listing 5 is accomplished in X10 using the simple keyword **when**. Listing 8 shows the same simple integer stack implemented in X10. Notice that the pop() method uses **when** to cause the thread to wait for a specific condition. The notify is implicit in the action of the push and doesn't require explicit coding by the programmer.

We have only scratched the surface of X10 features for concurrent programming. More thorough and complex examples of parallel programming in X10 are provided at http://x10.sourceforge.net/.

Summary

While performance improvement for single threads may slow significantly over the coming years, proces-

Vijay Saraswat joined IBM Research in 2003 after a year as a professor at Penn State, a couple of years at start-ups, and 13 years at Xerox PARC and AT&T Research. His main interests are in programming languages, constraints, logic, and concurrency. At IBM, he leads the work on the design and implementation of X10, a modern object-oriented programming language intended for scalable concurrent computing. Over the last 20 years he has lectured at most major universities and research labs in U.S.A. and Europe. Vijay got a B Tech degree from the Indian Institute of Technology, Kanpur, and an MS and PhD from Carnegie-Mellon University. His thesis on concurrent constraint programming won the ACM Doctoral Dissertation Award in 1989, and a related paper won a best-paper-in-20-years award in its area.

vsaraswa@us.ibm.com

Business Objects

RARE OCCURRENCE.

For a limited time, upgrade to Crystal Reports® XI for only $99. Create brilliant reports in minutes and speed report integration so you can focus on what you do best – core application coding. A great price with this depth of features is a rare occurrence, indeed.

- .NET, Java™ and COM SDKs
- Unlimited free runtime for internal corporate use
- Includes Crystal Reports Server – embed report management services
- Includes crystalreports.com – share reports over the web
- Unlimited installation-related support

NOW $99

UPGRADE or $395 NEW

Act fast. Go to www.businessobjects.com/rareoccurrence or call 1-888-333-6007 today.

sors will provide significantly expanding concurrency. For software performance to continue to improve, developers must begin thinking and coding in parallel. Fortunately, the Java programming language was designed for concurrency. Java provides all of the basic constructs for threading and locking. However, parallel programming is still very difficult due to inherent complexities such as dividing sequential tasks into balanced subtasks and avoiding race conditions and deadlocks. IBM has recently introduced a new programming language called X10 that runs on top of Java and significantly simplifies many of the tasks of parallel programming.

Resources

1. http://www.ibm.com/developerworks/power/newto/
2. http://www.sun.com/processors/niagara/
3. http://www.ibm.com/power
4. http://www.intel.com/multi-core/index.htm
5. http://multicore.amd.com
6. Brian Goetz. "Java Concurrency in Practice." http://www.briangoetz.com/pubs.html
7. Brian Goetz series of articles on developerWorks. http://www.ibm.com/developerworks/java/library/j-jtpcol.html
8. http://www.oreilly.com/catalog/jthreads3/
9. X10 http://x10.sourceforge.net/

Listing 1: Parallel Threads in Java

```
Thread t1 = new Thread (new Runnable() {
   public void run() { while(true) { System.out.println("tic"); }}
});
Thread t2 = new Thread (new Runnable() {
   public void run() { while(true) { System.out.println("toc"); }}
});

t1.start(); t2.start();
```

Listing 2: Pooled Parallel Threading in Java

```
ExecutorService svc = Executors.newCachedThreadPool();
Future tic[] = new Future[10], toc[] = new Future[10];
for (int i=0; i<10; i++) {
   tic[i] = svc.submit (new Runnable() {
            public void run() { for(int i=0; i<10; i++) System.out.
println("tic"); }
   });
   toc[i] = svc.submit (new Runnable() {
            public void run() { for(int i=0; i<10; i++) System.out.
println("toc"); }
   });
}

   try {
            for (int i=0; i<10; i++) {
                     tic[i].get(); toc[i].get();
            }
   } catch (InterruptedException e1) {
            return;
   } catch (ExecutionException e2) {
            return;
   }
```

Listing 3: Synchronized Updates

```
public class Counter {
   private int ctr = 0;
   public synchronized void Increment(int k) { ctr += k; }
   public synchronized void Decrement(int k) { ctr -= k; }
}
```

Listing 4: Atomic Increment

```
import java.util.concurrent.atomic.AtomicInteger;

public class AtomicCounter {
   AtomicInteger ctr = new AtomicInteger(0);
   public void Increment(int k) { ctr.addAndGet(k); }
   public void Decrement(int k) { ctr.addAndGet(-k); }
}
```

Listing 5 : Wait and Notify

```
private Node top = null;

public synchronized void push (int x) {
   Node newNode = new Node(x,top);
   top = newNode;
   notify();
}

public synchronized int pop() {
   while (top == null) {
            try { wait(); }
            catch (InterruptedException e) { }
   }
   int result = top.data;
   top = top.next;
   return result;
}
```

Listing 6: Parallel tasks in X10

```
 while (true) {
    async System,out.println("tic");
    async System,out.println("toc");
 }
```

Listing 7: Atomic Increment in X10

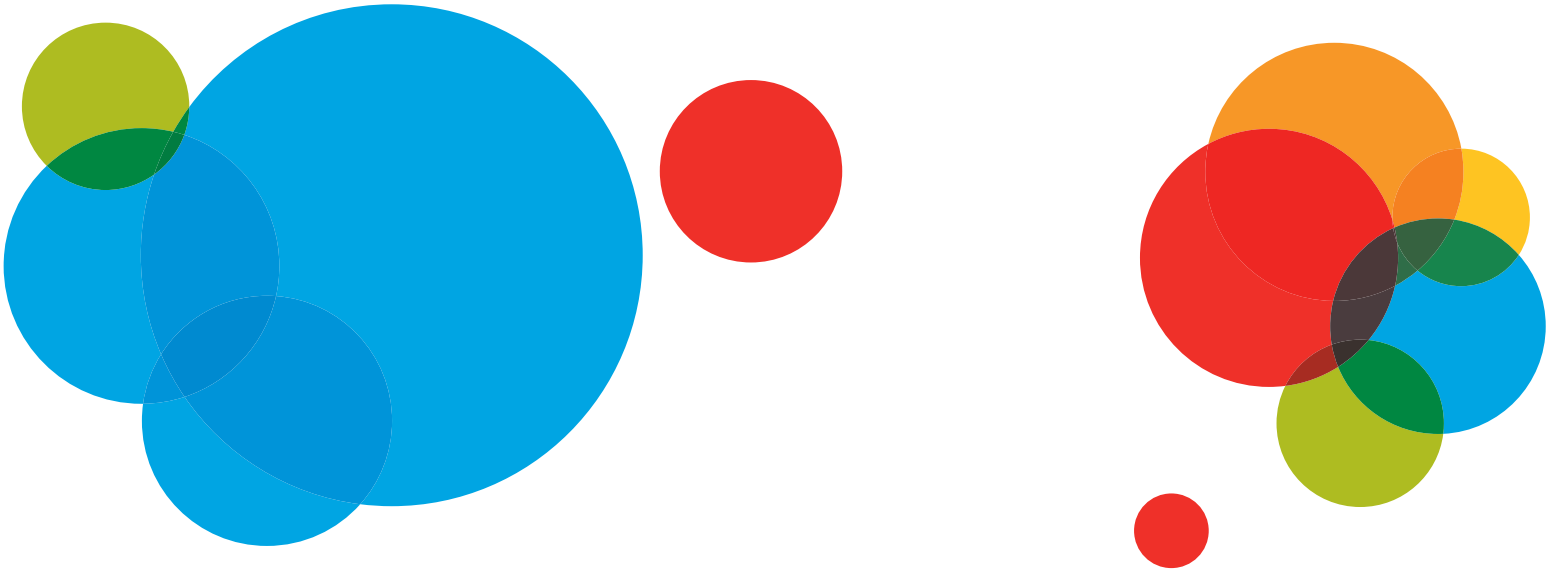
```
   atomic ctr = ctr + 1;
```

Listing 8: Wait/Notify in X10

```
private nullable<Node> top = null;

public void push (int x) {
   Node newNode = new Node(x,top);
   Atomic top = newNode;
}

public int pop() {
   when (top != null) {
            int result = top.data;
            top = top.next;
            return result;
}
}
```



JBoss Enterprise Application Platform is the market leading platform for next generation enterprise Java applications. Built on open standards, JBoss Enterprise Application Platform integrates JBoss Application Server, Hibernate, and Seam into a complete, simple enterprise solution for Java applications. Integrated, simplified and delivered by the leader in enterprise open source software.

It's a Multi-Core World: Let the Data Flow

A functional parallelism paradigm that fits multi-core processor architecture

by Jim Falgout and Matt Walker

The multi-core buzz is everywhere. Pick up a newspaper and the local electronics mega-store is advertising multi-core desktops and laptops to the consumer. Interesting, but what does it mean to the everyday Java programmer? Maybe nothing. If you live in the application server world writing EJB-based applications your application server does most of the heavy lifting for you. It handles concurrency just fine. But that doesn't cover all applications. Multi-core technology will especially affect applications that must process large amounts of data in a non-transactional (outside of a database context) manner. For this class of applications, the implications of multi-core are huge.

Why? Well first, notice the processing speeds of multi-core processors. They're not getting faster. In fact, they may be slowing down. As manufacturers add more cores to a chip, the processing speed of each core is usually slowed down to prevent overheating. The 80-core chip that Intel demonstrated recently wasn't 80 cores of x86 architecture but a simpler architecture. This may be an industry trend as more and more cores are squeezed onto a chip. The processor architecture may become heterogeneous, with a few full-power "legacy" cores and many specialized cores. The IBM Cell architecture already employs this scheme, with a single PowerPC core at the center and eight SPU cores connected using high-speed interconnects.

One implication you should take away from all of these processor changes: your single-threaded application may actually slow down on a multi-core system. If you need faster runtimes to meet shrinking SLA windows, you 'll have to multithread your applications, now! No problem, right? Java has included the java.util.concurrent package since Java5. This library contains many powerful constructs from which you can build a fully concurrent and scalable application. But, that isn't always easy or straightforward. The java.util.concurrent package is a set of building blocks that you must master and put to good use. There are several good books on this subject. We highly recommend *Java Concurrency in Practice* by Brian Goetz, for one.

There's a technology that's been around for years called *dataflow* that can solve the multi-core dilemma. How? This article will go into detail about dataflow, but the gist is this: dataflow provides a functional parallelism paradigm that fits well into multi-core processor architecture. A dataflow instance consists of a directed graph of processing nodes connected with FIFO data queues. This pipelined architecture lets applications be built from small-size reusable components stitched together with queues. The diagram below gives a small example of dataflow graph. Since it's a pipelined architecture, it naturally takes advantage of many processing cores. But more on that later.

First, we'll cover the overall nature and architecture of dataflow technology, specifically focusing on dataflow in software. Then we'll cover the history of dataflow technology, how it was first conceived and has matured and morphed over the years. Then we'll discuss several implementations of dataflow technology that exist in the marketplace today, highlighting a Java implementation. Read on for a peek into a technology that may have been ahead of its time but appears poised for the new multi-core world.

Yet Another Programming Paradigm

You may be asking: why another programming paradigm? First, the languages we have available to us don't directly support the needs of application builders. As mentioned before, the java.util.concurrent package provides most of the constructs needed to build scalable applications. However, these are lower-level building blocks. It can take many months to become familiar with these constructs and apply them right.

Second, the programming frameworks that have traditionally been used to build highly scalable applications have been targeted at the academic and scientific community. Frameworks such as MPI and OpenMP have been used to solve very large complex problems, taking advantage of some of the world's largest computers and computer grids. However, the confluence of the information explosion with the availability

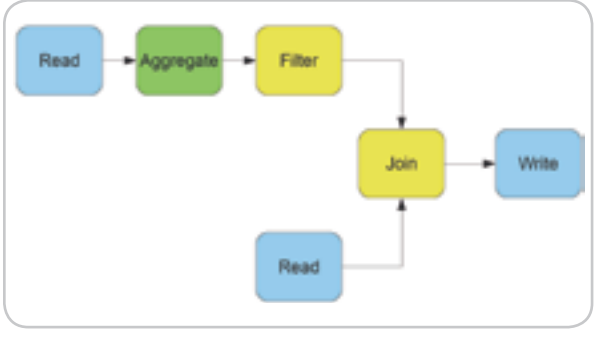


Figure 1 A simple dataflow diagram illustrating the base concepts

Jim Falgout is a solutions architect for Pervasive Software, where he applied dataflow principles to help architect Pervasive DataRush. Jim is active in the Java development community; this May, he presented a technical paper titled "Unleashing the Power of Multi-Core Processors: Scalable Data Processing in Java Technology" at JavaOne.

jfalgout@pervasive.com

of very inexpensive, off-the-shelf hardware has put high-performance computing within reach of even small and medium-sized businesses. These businesses have ever-increasing demands to process more data in shorter time periods. What they don't have is a staff of concurrent programming experts.

On the one extreme we have Java and all of the functionality that it provides. The building blocks to create scalable applications are there, but a cost must be paid to tap into this functionality. On the other extreme are frameworks such as MPI and OpenMP. Again, they provide high functionality, but have traditionally been used by the academic and scientific computing world. They are not easy tools to use. Something is needed to bridge this gap to provide high-performance, highly scalable data processing to the business world. Dataflow technology can be one way to bridge that gap.

Dataflow Programming Model

Dataflow is an alternative to the standard von Neumann model of computation. Typically, we think of a program as a series of instructions each executed one after the other by a processor keeping track of its progress with an instruction pointer. In dataflow, on the other hand, channels transmitting data in one direction join computations to one another. Conceptually, you can think of this structure as a directed graph with data channels as edges and processes performing computation on the data as nodes. The processes each operate only when data is available — the data flowing through the network is all that's needed to organize the computation. The immediate advantage is that many of the processes can be operating simultaneously, thus allowing dataflow applications to take advantage of hardware with multiple processor cores. Notice the concurrency happens external to the process; the developer doesn't have to bother with threads, deadlock detection, starvation, or concurrent memory access to build parallelism into his application.

This type of implicit parallelism stands in stark contrast to the concurrency mechanisms of many other programming paradigms. Gone are the locks of concurrent programming in imperative languages like C, which lack composability — two correct snippets of code using locks may not be correct when they're combined. Dataflow, on the other hand, allows composability: as long as the I/O contract is correct, sub-graphs may replace nodes or be spliced between them in the original dataflow network. This facilitates both program correctness, since sub-graphs can be tested as they're constructed then linked together to form larger programs, and code reuse, since commonly used sub-graphs can be copied from one application to the next.

Dataflow process networks bear some relationship to the dataflow variables of declarative programming languages like Oz. A dataflow variable is simply an unbound variable whose value can only be determined by a separate thread operating in parallel. If the dataflow variable is referenced before it's been bound, the referencing thread pauses awaiting the value. Combined with a single-assignment store (variables can be bound once at most), these variables lead to the nice property that it doesn't matter in what order we evaluate simultaneously executing expressions. Likewise, the outcome of a dataflow network is determined uniquely by its input, regardless of the order in which processes fire. Firing order impacts queue sizes and performance, of course, but this can be dealt with elsewhere besides explicitly within the program itself, dramatically simplifying the task of the dataflow programmer.

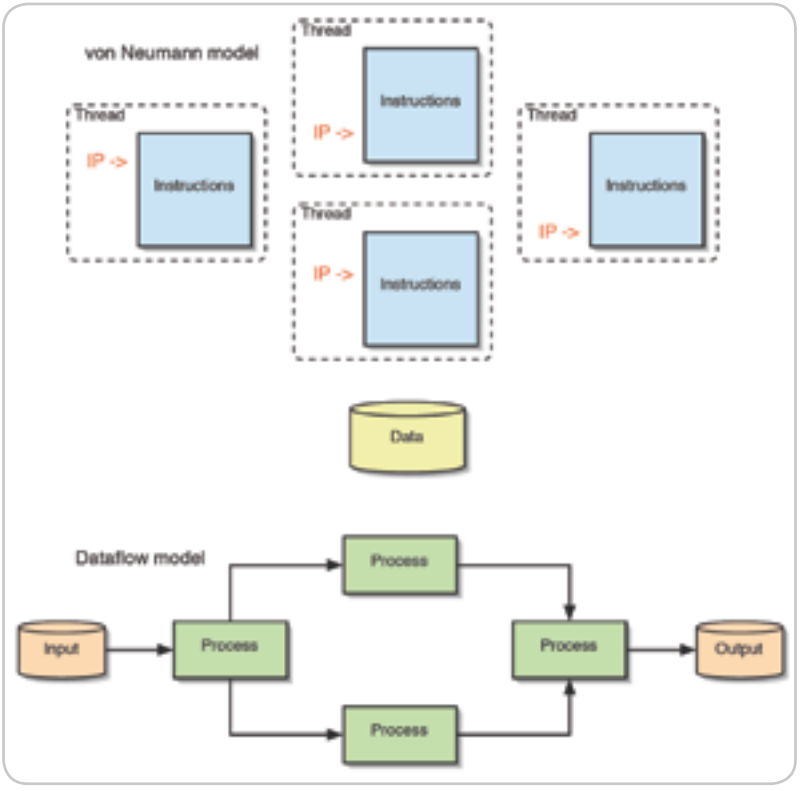


Figure 2 The programming paradigm difference between the von Neumann model and dataflow

The History of Dataflow

In the early 1970s, many people grew skeptical of the von Neumann architecture's ability to cope with parallelism. The global instruction pointer and memory could both become bottlenecks in concurrent software if it wasn't carefully designed. Dataflow architecture arose as the only compelling competitor. Designed with concurrency in mind, it eliminated the global instruction pointer and memory by organizing the computation based on the flow of data through a network of processes. However, these radically different architectures proved difficult compiler targets for traditional imperative programs. Dataflow programming and languages arose in response to this need.

At this time, Jack B. Dennis developed the static dataflow model and applied it to the design of computer architectures. His model limited nodes to primitive computations, and the edges were seen as representing data dependencies among the various operations occurring in the network — they held only one data token at a time. The work of Gilles Kahn extended this idea in two ways. First, the edges of his dataflow graphs were unbounded first-in, first-out queues, providing for a flexible rate of flow across each node. Second, he allowed each node to be a complete sequential process, which is often called large-grain dataflow. This approach tends to be more effective in creating efficient software since the threads implementing processes are given a larger fraction of work, reducing the amount of time spent switching between them. Further, the model freed the concept of dataflow from defining the language of its processes. Now it was conceivable to implement them in standard programming languages such as C or Java but still have the network of code operate according to dataflow principles.

Though the potential for distinguishing between the process language and the language or mechanism for coordinating these processes was recognized early on, it wasn't until the

Matt Walker is a researcher in the Pervasive Software Innovation Lab, seeking a deeper understanding of concurrent programming techniques to improve the Pervasive DataRush framework for dataflow programming in Java. He holds a master's in computer science from UT.

mwalker@pervasive.com

> Dataflow is an alternative to the standard von Neumann model of computation"

late 1980s and early 1990s that the idea drew much attention. Thomas M. Parks presented a Kahn process network scheduling policy in his PhD thesis that ensured bounded queues for infinite inputs, making practical implementations realizable. Simultaneously, projects focused on the software engineering aspects of dataflow began showing up. While not technically dataflow because it doesn't obey the dataflow execution model, J. Paul Morrison's flow-based programming explored the reusability of large-grain processes implemented in common programming languages. He applied these ideas to large systems in the banking industry, empirically measuring an improvement in programmer productivity. More modern frameworks combine the engineering benefits of languages such as Java with the dataflow execution model.

Application and Existing Implementations

Now that you have some background in dataflow technology, you're probably starting to see applications for it. The traditional arena for dataflow is signal processing, since that's where it got its start. And dataflow is still used in that area of the industry today. This is especially true in the academic world. A quick search for dataflow on the Web will show many universities with research activity in the area of signal processing using dataflow technology.

Along those lines, the LabVIEW toolset created by National Instruments has an architecture based on dataflow technology (http://www.ni.com/labview/). The outstanding user experience offered by this toolset lets a user build up a dataflow graph of data collection and data processing nodes very quickly. It appears to have used dataflow concepts as more of a functional paradigm than for performance. However, with the advent of multi-core and the availability of processing power at much more affordable price points, the LabVIEW toolset is poised to provide highly scalable processing due to its use of dataflow architecture.

There are other applications for dataflow technology beyond signal processing. The pipeline nature of dataflow implementations provides a natural fit for data processing applications. Since the data is pipelined in a dataflow architecture, massive amounts of data can be processed in a highly scalable way. This ability implies that dataflow techniques can be applied to many industry problems, including:

- Data mining and data warehousing
- Data analytics and business intelligence
- ETL (extract, transform, and load) processing
- Data quality
- Fraud detection

One of the hybrid approaches to dataflow implementation has been used to create massively scalable data processing engines. Way back in the 1990s, a start-up company called Torrent created a C++-based dataflow framework named Orchestrate. This framework implemented dataflow techniques and could run across a cluster of homogeneous systems. Several Torrent customers created business intelligence applications using this dataflow framework. Torrent was eventually acquired by Ascential, which was then acquired by IBM in 2005.

Another data processing engine using dataflow architecture has been built with 100% Java technology. This engine, available from Pervasive Software, uses more of the style of flow-based programming (www.pervasivedatarush.com). It's currently available as part of a free public beta program sponsored by Pervasive. See the sidebar for more information.

Conclusion

As the information age and the multi-core wave continue to collide, more pressure is exerted on software developers to provide access to increasingly inexpensive computing horsepower. High-performance computing was once the domain of system experts, government agencies and universities. Now it's in demand by large as well as medium-sized companies with massive amounts of data to process and shrinking time windows. Compute power that used to cost millions is now available in systems that can be ordered on the Web for $20k.

All of this leads to the need for a better programming paradigm: a paradigm that encourages the developer to build highly performing and scalable software without the burden of low-level system knowledge. Dataflow is one such paradigm. Dataflow, being a pipelined architecture, is inherently scalable. And as we've pointed out, dataflow concepts have been around for many years. They've undergone change and growth as the ideas have matured in the academic community.

There are also several commercial implementations of dataflow in the marketplace, a sign that dataflow technology is real and has many benefits to bring to the software development community. We encourage you to investigate dataflow concepts and determine how they can fit into your software architectures going forward.

Resources

- Johnston, Hanna, and Millar. "Advances in Dataflow Programming Languages." 2004. http://portal.acm.org/citation.cfm?id=1013208.1013209
- Najjar, Lee, and Gao. "Advances in the Dataflow Computational Model." 1999. http://ptolemy.eecs.berkeley.edu/publications/papers/99/dataflow/
- Parks. "Bounded Scheduling of Process Networks." 1995. http://ptolemy.eecs.berkeley.edu/publications/papers/95/parksThesis/
- Harris, Marlow, Jones, and Herlihy. "Composable Memory Transactions." 2006. http://research.microsoft.com/~simonpj/papers/stm/#composble
- Van Roy and Haridi. "Concepts, Techniques, and Models of Computer Programming." 2004. http://www.info.ucl.ac.be/~pvr/book.html

Pervasive DataRush: A Dataflow Implementation

by Jim Falgout

The article on dataflow concepts introduced you to dataflow technology and its relevance to multi-core processing. Here we'll discuss an implementation of dataflow technology built with Java. The application framework is called Pervasive DataRush and is currently in beta release from Pervasive Software.

DataRush is an application development framework. Its purpose is to enable the user to build data processing applications that can easily take advantage of multi-core processors to produce highly scalable software. DataRush implements many dataflow concepts and extends some of these concepts to provide several ways to dramatically increase scalability of applications.

DataRush implements a scripting language, based on XML that provides the means to create reusable components and data processing applications. This language, called DFXML (dataflow XML), is simple in syntax and very flexible. DFXML is used to compose what are called assemblies. An assembly is a composite operator and can be composed of other assemblies, processes, and customizers. A process is the lowest level of operator. It's written in Java and performs the work in the dataflow graph. A customizer is a compiler helper also written in Java that enables the dynamic nature of the DataRush framework. DataRush includes a component library that consists of more than 50 pre-built, ready-to-use components provided by the framework. Included are components that provide connectivity to data and data processing components such as sort, join, merge, lookup, group, and so on.

The architecture block diagram in Figure 1 depicts the high-level architecture of the DataRush framework.

The user utilizes an IDE such as Eclipse to create DFXML assemblies and Java processes and customizers (http://www.eclipse.org). The DataRush assembler is used to convert DFXML scripts into a binary form. The DataRush execution engine is then invoked to compile the binary files into a dataflow graph for execution. This compilation step is run for every engine instantiation, providing a very dynamic graph-generation capability. Once the dataflow graph is generated, the DataRush engine creates threads and dataflow queues representing the graph and executes the graph. Execution monitoring is provided via JMX.

Dataflow Implementation

The Pervasive DataRush framework implements many of the basic structures of dataflow. Processing nodes (processes in DataRush) are built in Java and interface using dataflow queues. The dataflow queues in DataRush are typed and support native Java types besides string, date, timestamp, and binary.

The dataflow queues in DataRush are somewhat comparable in functionality to the blocking queue implementations in the java.util.concurrent package introduced in the Java 5 release. They're both memory-based queues that block readers on empty queues and block writers of full queues. The DataRush queues, however, must support deadlock detection and handling. Due to support for multiple queue readers and the fact that processes can have multiple inputs and outputs, cycles of dependencies can be created in a dataflow graph. These cycles can lead to deadlock, whereby writers and readers are waiting in a way that needs intervention for the graph to continue working. A deadlock algorithm in the DataRush engine detects deadlock situations and handles it, normally by temporarily expanding the size of the problematic queue.

Besides the pipeline scalability that a dataflow architecture already provides, the Pervasive DataRush framework has built-in support for two other types of scalability: horizontal partitioning and vertical partitioning. Horizontal partitioning replicates a section of dataflow logic and segments the input data into chunks, flowing the data concurrently through the replicated dataflow sections. Figure 2 depicts this scenario using a lookup component as an example. In this example, the lookup operator is replicated with a data partitioner spreading the data load evenly to each lookup instance. This lets each lookup operator run in parallel, fully utilizing multiple cores on the system. Vertical partitioning supports running different dataflow logic in parallel on each field of an input stream. Figure 1 shows the high-level architecture of the Pervasive DataRush framework including design and execution components. The user utilizes an IDE such as Eclipse to create DFXML assemblies and Java processes and customizers. Figure 2 exemplifies horizontal partitioning, one of three types of scalability, which can be implemented using Pervasive DataRush. Horizontal partitioning replicates a section of dataflow logic and segments the input data into chunks, flowing the data concurrently through the replicated dataflow sections.

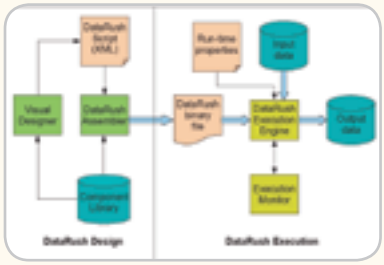


Figure 1 DataRush architecture block diagram

Figure 2 Horizontal partitioning

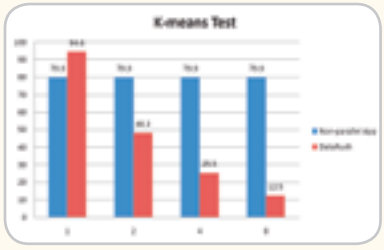


Figure 3 K-means results

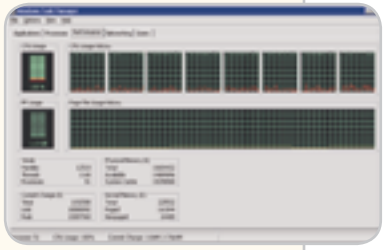


Figure 4 CPU snapshot

Why Java?

As the article on dataflow points out, there have been many instantiations of dataflow technology over the years. Most of them have been implemented in C or C++. This makes sense due to the prevalence of C and C++ when the systems were built. When DataRush was first being developed, the decision was made to use Java as the programming language. This decision was based on several factors: portability, flexibility, extensibility, and scalability – and you can throw in productivity for good measure. The decision was also based on the high level of industry investment in JVM technology. Over the past few years, we've seen significant performance improvements with each JDK release. Also, the amount of open source libraries available is astounding. With such a rich environment, the decision has proved to be a good one.

The question always arises about Java and performance. What we've found, with the introduction of the java.nio package and other JVM performance enhancements, is that native speeds can be obtained from Java. This is especially true for frameworks like DataRush in which a static set of classes (the process nodes) are utilized over a relatively long period of time. This scenario provides an environment well suited for JIT compilers.

A Simple Benchmark

To demonstrate the scalability of the DataRush framework, we developed a simple benchmark implementing a one-pass K-means algorithm. The algorithm takes two double-typed values as points and clusters the points into like groups. The benchmark measures the performance of running K-means on 100 input columns over 10 million rows of data. For this particular test, the input data is generated. As can be seen from Figure 3, the performance of the benchmark test improves as more CPU resources are made available. These benchmark results of a K-means test run on an 8-core machine demonstrate how a non-parallelized application fails to scale as more compute resources are added. A snapshot of the CPU utilization is also provided, showing that the DataRush framework was able to keep the machine heavily utilized for the duration of the test. Figure 4 shows CPU usage during the K-means benchmark, the Pervasive DataRush platform has scaled to take full advantage of all 8 cores available on the machine used for this test.

Conclusion

The DataRush application development framework implements dataflow concepts that enable Java programmers to create highly scalable applications that can process many million rows of data. The framework is currently in beta release and can be downloaded at http://www.pervasivedatarush.com. DataRush is built completely in Java and so is easy to install and begin using right away. A user interface in the Eclipse IDE is being developed, so please check back with the site periodically for updates on that development. The site also includes more information on DataRush and forums for discussion and questions.

Introduction to Maven

Part III – Application development management using Maven 2 and Eclipse

by Murali Kashaboina and Geeth Narayanan

In the parts 1 and 2 of this article, we demonstrated how to download and install Maven 2, how to install the Maven 2 plugin for Eclipse, and how to go about setting up a project directory structure using Maven 2. We used a simple use case for displaying employee details on the Web given an employee ID, but deliberately made the design a bit complex by introducing design concepts such as XML binding, EJBs, and JCA connectors to illustrate a few of the many features offered by Maven. In this final installment of the article, we continue with the remaining modules in our example and illustrate a few more developmental tasks that can be accomplished fairly easily using Maven that otherwise would demand significant time and effort to accomplish.

As described earlier, the 'connector' module yields a RAR artifact containing a JCA connector that uses 'xmlBinding' classes to return Employee information. This means that the 'connector' module will have a dependency on the 'xmlBinding' artifact and any artifact that can provide JCA classes. We'll use Maven's 'Add Dependency' feature in Eclipse to add these dependencies as described below.

1. In the Eclipse 'Package Explorer' pane, right-click on the 'connector' module's POM file and in the menu, select 'Add Dependency' Maven2 option as shown in Figure 1.
2. A dialog window to search for artifacts in the Maven repository will be displayed. In the search box, type *'com.somecompany.'* The search results will be displayed in the text area as shown in Figure 2.
3. Expand the *'com.somecompany xmlBinding'* result entry, select the *'xmlBinding-1.0.jar'* option, and click the 'OK' button. The dependency on the 'xmlBinding' artifact will be added in the POM file.
4. To add JCA specification classes, we'll use a Geronimo JCA specification artifact as a dependency. Following the same approach as described before, search for 'geronimo' in the repository search window and select 'geronimo-spec-j2ee-connector-1.0-M1.jar' as a dependency to be added as shown in Figure 3.
5. The two dependencies will be added to the POM file. The scope element should be added to the *'geronimo-spec-j2ee-connector'* dependency element with a value of 'provided' indicating that JCA classes will be provided by the underlying J2EE application server. Below is the modified POM.

```
<project>
  <parent>
    <artifactId>EmployeeInfo</artifactId>
    <groupId>com.somecompany</groupId>
    <version>1.0</version>
  </parent>
  <modelVersion>4.0.0</modelVersion>
  <artifactId>connector</artifactId>
  <packaging>rar</packaging>
  <name>connector</name>
<dependencies>
    <dependency>
    <groupId>com.somecompany</groupId>
    <artifactId>xmlBinding</artifactId>
    <version>1.0</version>
  </dependency>
  <dependency>
    <groupId>geronimo-spec</groupId>
    <artifactId>geronimo-spec-j2ee-connector</
      artifactId>
    <version>1.0-M1</version>
    <scope>provided</scope>
  </dependency>
 </dependencies>
</project>
```

It's time to add connector implementation Java classes. The source files that we developed for the connector are shown in Figure 4.

Download and review the source files to understand the complete implementation. However, the important classes to note are *'EmployeeInfoCCIConnection,'* *'EmployeeInfoCCIInteraction,'* and *'EmployeeInfoSPIManagedConnection.'* The sequence diagram as shown in Figure 5 gives a high-level view of the method calls that a client will invoke to retrieve employee information using an employee ID. For brevity's sake, the sequence diagram is kept simple.

The RAR artifact for the *'connector'* module can be created by adding *'maven-rar-plugin.'* By default, this plug-in will look for RAR meta-information such as the *'ra.xml'* descriptor file under the 'src/main/rar/META-INF' directory. The 'ra.xml' descriptor is in Listing 1.

Static Analysis of Code using Maven

Errors in the code are typically detected via code reviews, unit testing, system testing, integration testing, and user-acceptance testing. Code reviews certainly help in early bug detection by enforcing language-specific programming standards and best practices. However code reviews are carried out manually and hence the process can be cumbersome and inefficient particularly in case of large projects. Static analysis is a tool-based automated code review mechanism typically used to find code defects early in the build phase. Static analysis ensures early bug detection and remediation by comparing source code with predefined language patterns. It also helps in enforcing coding conventions and thereby improves code quality.

PMD is a static analysis tool for Java code. PMD packages a number of ready-to-run rules that can identify unused variables, unnecessary object creation, and empty catch blocks in

Figure 1

Murali Kashaboina is a lead architect at Ecommerce Technology, United Airlines, Inc. He has more than 10 years of enterprise software development experience utilizing a broad range of technologies, including JEE, CORBA, Tuxedo, and Web services. Murali previously published articles in WLDJ and SilverStream Developer Center. He has master's degree in mechanical engineering from the University of Dayton, Ohio.

murali.kashaboina@united.com

the source code. Custom rules can also be incorporated. PMD can be executed using Maven by including *'maven-pmd-plugin'* in the POM file as shown in the following snippet. The PMD plug-in lets you automatically run the PMD code analysis tool on your project's source code and generate a site report with its results. For more information on the PMD Maven plug-in, refer to plug-in documentation available at http://maven.apache.org/plugins/maven-pmd-plugin/.

```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-pmd-plugin</artifactId>
   </plugin>
```

PMD plug-in can be invoked at the command line within the module directory by running the command below. However, with the current implementation, the plug-in will generate reports only in case of 'jar,' 'war,' and 'ejb' POM packaging types. Since the current POM is a 'rar' packaging type, the packaging type should be temporarily changed to 'jar' so that PMD plug-in can generate the reports.

```
mvn pmd:pmd
```

When this command is executed after temporarily changing the packaging type to 'jar,' Maven will invoke the PMD plug-in that will run the code analysis and create reports under the *'target/site'* directory with the main report in the *'pmd.html'* file. Figure 6 is the report generated for the *'connector'* module source code. Make sure to reset the packaging type back to 'rar.'

Packaging and Installing 'Connector' Artifacts

With the current implementation of the plug-in, the plug-in won't automatically create a separate JAR file for the module classes. If the JAR file for the current classes is created separately, there's an option to include the JAR in the final RAR file. Note that the packaging type for the current module is 'rar' and so no separate JAR file is created for the *'connector'* classes. This means that we'll have to explicitly add a separate plug-in to create the JAR file. This is very simple to do just by adding *'maven-jar-plugin'* to the POM file. Note that the JAR plug-in should be put before the RAR plug-in since the plug-ins are executed in the order they appear in the POM file. The plug-in configuration added to the POM file is in Listing 2.

Note that by default both plug-ins get executed during the 'package' phase of the

Figure 2

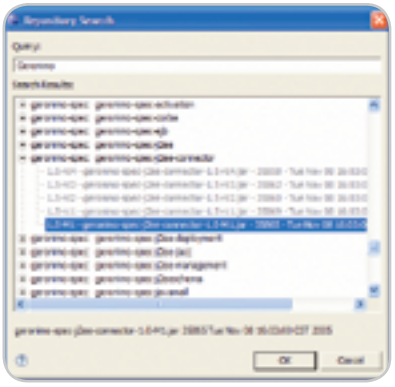


Figure 3

Maven lifecycle. To deploy the *'connector'* RAR artifact to the local repository, right-click on the module's POM file in Eclipse and then in the 'Run As' options, select 'Maven2 install.' Maven will execute the configured plug-ins to build both JAR and RAR files in the 'target' build directory and will copy over the primary artifact, the RAR file, to the local repository along with the POM information.

Attaching Additional Artifacts

Sometimes, you may have to deploy additional artifacts to the Maven repository. For example, in the case of *'connector'* module, we can attach the JAR file as an artifact along with the primary RAR artifact. This is fairly easy to do by using *'build-helper-maven-plugin'* from the *'org.codehaus.mojo'* plug-in group. The snippet below shows the configuration for the plug-in to attach the JAR file as an additional artifact.

```
<plugin>
  <groupId>org.codehaus.mojo</groupId>
  <artifactId>build-helper-maven-plugin</artifac-
tId>
  <executions>
    <execution>
      <id>attach-artifacts</id>
      <phase>package</phase>
      <goals>
        <goal>attach-artifact</goal>
      </goals>
    <configuration>
      <artifacts>
        <artifact>
          <file>${project.build.directory}/${project.
              build.finalName}.jar</file>
          <type>jar</type>
        </artifact>
      </artifacts>
    </configuration>
    </execution>
</executions>
</plugin>
```

Please refer to the plug-in documentation at http://mojo.codehaus.org/build-helper-maven-plugin/index.html for additional information. After adding '*build-helper-maven-plugin*' to the POM file, redeploy the *'connector'* to the Maven repository and you should see the JAR file along with the RAR file deployed in the local repository as shown in Figure 7.

Setting Up an 'ejb' Module

We'll move on to create an 'ejb' module with a default Maven project. To do this, go to the 'EmployeeInfo' directory on the command prompt and execute the following Maven command:

```
mvn archetype:create -DgroupId=com.some-
company -DartifactId=ejb –Dversion=1.0
```

Refresh the *'EmployeeInfo'* project in Eclipse and update the Maven source directories. Delete the sample Java class and its JUnit Test case. As in the case earlier, we'll use Maven's project inheritance model as described before and remove redundant version, group, and common dependency information from the *'ejb'* POM file. Since this module will yield an EJB artifact, we'll change the packaging type to *'ejb'* in the POM.

The *'ejb'* module provides a stateless session bean that internally invokes the JCA connector to retrieve the Employee info XML object. This implies that the *'ejb'* module will have dependencies on both *'xmlBinding'* and *'connector'* artifacts. However, since the *'connector'* module already depends on the *'xmlBinding'* module, it's sufficient to include dependency on the *'connector'* artifact alone since Maven will transitively resolve dependencies and thereby transparently include dependency on the *'xmlBinding'* artifact. Note that this module also needs compile-time EJB specification classes and classes from the J2EE connector specification as the Session bean refers to some of J2EE connector classes internally. We'll add dependencies

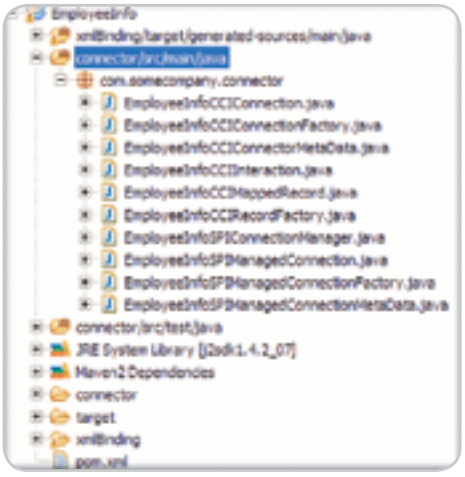


Figure 4

Figure 5

Figure 6

on *'geronimo-spec-ejb-1.0-M1.jar'* and *'geronimo-spec-j2ee-connector-1.0-M1.jar'* with a scope value of 'provided.'

Finally, we'll add *'maven-ejb-plugin,'* which gets executed during the 'package' phase and can build both the EJB JAR and EJB client JAR. Note that any client module that invokes an EJB can only include an EJB client JAR artifact as a dependency without dependency on the actual EJB JAR. This type of dependency inclusion will be shown later. Generating an EJB client JAR can be done by setting the *'generateClient'* plug-in configuration element to true. Note that the contents of the EJB client JAR file can be customized using *'clientIncludes'* and *'clientExcludes'* plug-in configuration elements. By default, the plug-in excludes the following from the EJB client JAR.

- **/*Bean.class
- **/*CMP.class
- **/*Session.class
- **/package.html

The plug-in doesn't do any ejb-specific processing during the generation of the EJB JAR except for validating the existence of an ejb deployment descriptor if the ejb version is 2.0+. By default the plug-in assumes the 2.1 version. In EJB3, the *'ejb-jar.xml'* deployment descriptor isn't required and in such cases, the exact version to be used can be specified using the *'ejbVersion'* plug-in configuration element. For more information on *'maven-ejb-plugin'* see the plug-in documentation at http://maven.apache.org/plugins/maven-ejb-plugin/index.html.

The *'ejb-jar.xml'* file and application server-specific EJB descriptor file should be put under the 'src/main/resources/META-INF' directory. The *'maven-ejb-plugin'* by default picks them up from that location while building the EJB archive file. We'll add the necessary EJB source files for this module as shown in Figure 8. Please download and review the source code to understand the implementation.

The *'GetEmployeeInfoRemote'* interface defines the business method *'getEmployeeInfo (String employeeId).'* The *'GetEmployeeInfoBean'* is the stateless session bean class that provides a concrete implementation for *'getEmployeeInfo'* method as shown below.

Figure 7

```
public Employee getEmployeeInfo(String employeeId) throws Exception {
   Employee employee = null;
   try {
     ConnectionFactory connectionFactory = getConnectionFactory();
     Connection connection = connectionFactory.getConnection();
     RecordFactory recordFactory = connectionFactory.getRecordFactory();
     MappedRecord input = recordFactory.createMappedRecord("EmployeeInfoInput");
     input.put(EmployeeInfoCCIRecordFactory.EMPLOYEE_ID_INPUT, employeeId);
     MappedRecord output = recordFactory.createMappedRecord("EmployeeInfoOutput");
     Interaction interaction = connection.createInteraction();
     interaction.execute(null, input, output);
     employee = (Employee) output.get(EmployeeInfoCCIRecordFactory.EMPLOYEE_RESULT);
   } catch (Exception e) {
     throw e;
   }
   return employee;
}
```

Geeth Narayanan is a senior architect at Ecommerce Technology, United Airlines, Inc. He has 10 years of experience in the IT industry, specializing in solutions using Java EE technologies. Geeth has master's degree in electrical engineering from the University of Toledo, Ohio.

geethakrishn.narayanan@united.com

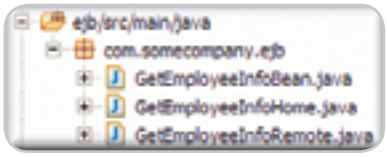


Figure 8

Figure 9

Note that the *'getEmployeeInfo'* method internally calls *'getConnectionFactory'* method to get a reference to the actual connection factory. The *'getConnectionFactory'* is a protected method, as shown below, and helps in making the bean testable.

```
protected ConnectionFactory getConnection-
    Factory() throws Exception {
   Context context = new InitialContext();
   return (ConnectionFactory) context.lookup
     (connectionFactoryJNDIName);
}
```

Running JUnit Tests Using Maven

We'll create a JUnit test case for *'GetEmployeeInfoBeanTest'* under the *'src/test/java'* directory with one *'testGetEmployeeInfo'* test method as shown below:

```
public class GetEmployeeInfoBeanTest extends
TestCase {
 public void testGetEmployeeInfo() throws Exception
{
    String employeeId = "1000";
    GetEmployeeInfoBean employeeInfoBean = new
GetEmployeeInfoBean() {

protected ConnectionFactory getConnectionFactory()
throws Exception {

EmployeeInfoSPIManagedConnectionFactory managedCon-
nectionFactory = new EmployeeInfoSPIManagedConnecti
onFactory();
      return (ConnectionFactory)managedConnectionFa
ctory.createConnectionFactory();
```

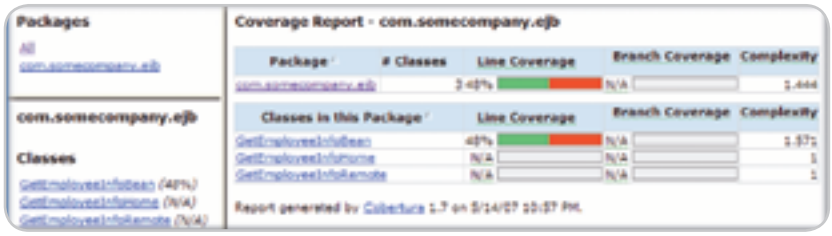


Figure 10

```
                                    }
                                };
    Employee employee = employeeInfoBean.getEmployee
Info(employeeId);
    assertNotNull( employee );
    assertTrue( employee.getId().equals(employeeId)
);
 }
}
```

To run the test cases using Maven, right-click on the module's POM file in Eclipse and then in 'Run As' options, select 'Maven2 test.' Maven will compile the Java files found in *'src/test/java'* and execute JUnit test cases. The result of test case execution can be found in the console.

Skipping Maven Test Execution

Maven will execute test cases whenever a 'test' phase or any other phase that occurs after a 'test' phase in the Maven lifecycle management is invoked. For example, Maven will execute test cases when a Maven 'package' or 'install' phase is invoked. Maven does this by triggering *'maven-surefire-plugin'* mojo at the time of the 'test' phase. This plug-in is responsible for executing the JUnit test cases found in the *'test-classes'* directory in the project target build directory.

Test case execution can be explicitly skipped for compelling reasons. There are a couple of ways to make Maven skip test execution. One way is to explicitly include *'maven-surefire-plugin'* in the POM file and set the value of the *'skip'* configuration element to 'true' as shown here:

```
<plugin>
   <groupId>org.apache.maven.plugins</groupId>
   <artifactId>maven-surefire-plugin</artifactId>
   <configuration>
    <skip>true</skip>
   </configuration>
    </plugin>
```

If Maven is executed at the command line, test case execution can be skipped by setting the *'maven.test.skip'* system property to 'true' as shown below:

mvn –Dmaven.test.skip=true install

Test cases can also be selectively skipped. This can be done by including *'maven-surefire-plugin'* in the POM and explicitly specifying test classes using wildcard patterns in the *'excludes'* configuration element as shown in the following snippet:

```
<plugin>
   <groupId>org.apache.maven.plugins</groupId>
   <artifactId>maven-surefire-plugin</artifactId>
   <configuration>
    <excludes>
     <exclude>**/*FunctionTest.java</exclude>
    </excludes>
   </configuration>
    </plugin>
```

For more information on 'maven-surefire-plugin,' see the plug-in documentation at http://maven.apache.org/plugins/maven-surefire-plugin/index.html.

Reporting Source Code Test Coverage Using Maven

Code coverage is a measure of how thoroughly test cases exercise the main application source code. This is an indirect way of measuring the quality of the tests. Code coverage is a type of white-box testing as logical test assertions are made against the internals of our classes and not against other subsystems or components the current application may interact with. Code coverage helps to isolate logical paths in the source code that aren't exercised under the test.

There are many tools to measure code coverage. Cobertura is an open source tool that can be used to measure test coverage. Cobertura instruments the code base and tracks lines of code that are being executed and lines that aren't being executed as the test suite runs. Besides identifying untested code, Cobertura can also optimize code by flagging dead and unreachable code. Cobertura has a Maven plug-in that can be configured in the POM. The plug-in has a set of goals that can be invoked separately. For more information on Cobertura Maven plug-ins, refer to plug-in documentation at http://maven-plugins.sourceforge.net/maven-cobertura-plugin/. We'll add the

Figure 11

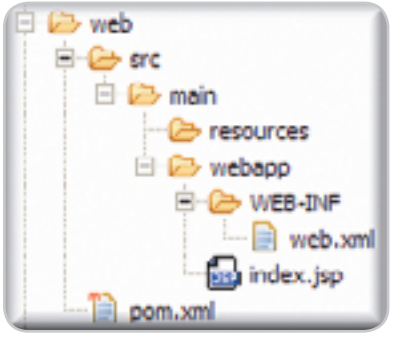


Figure 12

following plug-in configuration to the *'ejb'* module POM.

```
<plugin>
   <groupId>org.codehaus.mojo</groupId>
   <artifactId>cobertura-maven-plugin</artifac-
tId>
   <version>2.0</version>
   <executions>
    <execution>
      <goals>
        <goal>cobertura</goal>
      </goals>
    </execution>
   </executions>
    </plugin>
```

The Cobertura plug-in can be invoked at the command line in the module directory by running the command:

```
mvn cobertura:cobertura
```

When this command is executed, Maven will invoke the Cobertura plug-in so it can instrument the source code and measure test coverage. This plug-in will compile the instrumented code separately and generate the compiled instrumented classes under the *'target/generated-classes'* directory. The coverage reports will be typically generated under the *'target/site/cobertura'* directory as shown in Figure 9.

The report can be viewed by opening 'index.html' in a browser as shown in Figure 10.

The line coverage is only 48% since we have only one simple test case. However, coverage can be increased by inspecting the class in the report to check which methods and lines aren't covered and by adding sufficient cases to test different logical conditions so that almost all lines of code get executed. Typically a code base is considered to be under good test coverage if the line coverage is more than 80%, although achieving upwards of 90% isn't that hard with the appropriate test setup.

Packaging and Installing 'ejb' Artifacts

Listing 3 is the final modified POM file for *'ejb'* module and Listing 4 is the *'ejb.xml'* file that should be under the *'src/main/resources/META-INF'* directory. (Listings 4 and 5 can be downloaded from the online version of this article at http://java.sys-con.com).

To deploy an *'ejb'* artifact to a local repository, right-click on the module's POM file in Eclipse and then in the 'Run As' options select 'Maven2 install.' Maven will execute *'maven-ejb-plugin'* to install both EJB JAR and EJB client in the local repository along with POM information as shown in Figure 11.

Setting Up a 'web' Module

The creation of *'web'* modules is fairly simple using Maven. Maven provides a *'maven-archetype-webapp'* archetype to create the Web application project directory structure. We'll use *'maven-archetype-webapp'* to create a *'web'* module. To do this, go to the 'EmployeeInfo' directory on the command prompt and execute the Maven command:

```
mvn archetype:create -
DarchetypeGroupId=org.apache.maven.
archetypes -DarchetypeArtifactId=maven-
archetype-webapp -DgroupId=com.some-
company -DartifactId=web -Dversion=1.0
```

Maven creates the *'Web'* module directory structure as shown in figure 13. Note that *'maven-archetype-webapp'* creates a *'web/src/main/webapp'* directory containing a default *'index.jsp'* page and a *'WEB-INF'* directory containing a *'web.xml'* file. Even though an *'src/main/java'* directory isn't created, any source code needed by the Web application can be put in that directory. Similarly any JUnit test cases for the Web application can be put in the *'src/test/java'* directory. Note that *'maven-archetype-webapp'* creates a *'web/src/main/resources'* directory that can be used for storing any application resources such as properties files that will be needed in the runtime classpath (see Figure 12).

Refresh the *'EmployeeInfo'* project in Eclipse and update the Maven source directories. As was the case earlier, we will use Maven's project inheritance model to remove any redundant version, group, and common dependency information from the *'web'* POM file. Note that by default POM file has a packaging type set to 'war.'

The *'web'* module is a very simple Web application that consists of an index page containing a simple HTML form. The form has a text field to input Employee IDs. When the form is posted with an Employee ID, the HTTP request is received and processed by a Struts action. The Struts action retrieves the employeeId, looks up the stateless session EJB *'GetEmployeeInfoBean,'* and invokes the 'getEmployeeInfo' method to retrieve the Employee details.

The Struts action then stores the Employee details Castor object as a request attribute and forwards the request to the *'employeeInfo.jsp'* page. The *'employeeInfo.jsp'* page uses JSTL tags to render the Employee information. This description implies that the *'web'* module will have dependencies on *'ejb,' 'xmlBinding,' 'castor,' 'struts,' and 'jstl'* artifacts. However, since the *'ejb'* module depends internally on *'xmlBinding'* and since *'xmlBinding'* depends on the *'castor'* artifact, it's sufficient to include dependency on the *'ejb'* artifact without any need to explicitly include dependencies on *'xmlBinding'* and *'castor'* since Maven will transitively resolve dependencies. We'll add dependencies on the 'struts-1.2.4.jar' and 'jstl-1.1.2.jar' artifacts in the POM file. Note that this module also needs compile time servlet and EJB specification classes. We'll add dependencies on *'geronimo-spec-j2ee-1.0-M1.jar'* with a scope value of 'provided'.

Listing 5 is the modified POM.

Note the highlighted dependency declaration for the *'ejb'* artifact. We included the *'type'* element with a value of *'ejb-client'* to indicate to the Maven dependency manager to include the EJB client JAR instead of an

EJB main JAR as a dependency. As described earlier in the context of the *'ejb'* module, we explicitly forced the *'maven-ejb-plugin'* to create a client JAR along with a main EJB JAR. As a result of the above declaration, the Maven dependency manager will include *'ejb-1.0-client.jar'* in the Web application classpath.

We'll add the source code, JSPs, 'web.xml,' and 'struts-config.xml' files for the *'web'* module as shown in Figure 13.

Packaging and Installing the '*web*' Module

To deploy a *'web'* artifact to the local repository, right-click on the module's POM file in Eclipse and then in 'Run As' options, select 'Maven2 install.' Maven will compile, package, and install the *'web-1.0.war'* artifact in the local repository along with the POM information.

Setting Up an '*ear*' Module

Setting up an 'ear' module is fairly easy. We'll use the command below to create a basic Maven project in the 'EmployeeInfo' directory and edit the POM to change the packaging type to 'ear' and remove redundant group, version, and dependency information since such information is inherited from parent POM.

```
mvn archetype:create -DgroupId=com.somecompany -DartifactId=employeeInfoEAR –Dversion=1.0
```

To build an EAR file, the first step would be to add *'maven-ear-plugin'* to the POM file. The EAR plug-in replaces the Jar plug-in when the project <packaging> is set to *'ear.'* This plug-in can generate an *'application.xml'* file based on the plug-in configuration information provided in the POM. The plug-in configuration provides the ability to add different J2EE modules such as RAR, EJB, JAR, and WAR. Note that the *'maven-ear-plugin'* will include different modules in the final EAR file that are declared under the <*modules*> element in the plug-in configuration using a module-specific configuration element such as <*rarModule*>. All the modules that will be included under the plug-in configuration should be declared as dependencies in the POM file. The following are the common configuration options available for the module-specific configuration element:

- **groupId** – Sets the groupId of the current artifact you want to configure.
- **artifactId** – Sets the artifactId of the current artifact you want to configure.
- **classifier** – Sets the classifier of the current artifact you want to configure if multiple artifacts under the ear matches the groupId/artifact.
- **bundleDir** – Sets the location of current artifact inside the ear archive. If not set, the current artifact will be packaged in the root of the archive.
- **bundleFileName** – Sets the new name for the current artifact inside the ear archive. If not set, the artifact's filename in the repository is used.
- **excluded** – Set to true to exclude the current artifact from being packaged into the ear archive. Default is false.
- **uri** – Sets the uri path of the current artifact within the ear archive. Automatically determined when not set.
- **unpack** – Set to true to unpack the current artifact into the ear archive according to its uri. Default is false.

Including JEE Modules in EAR

The RAR module can be included in the EAR by configuring the <*rarModule*> element in the plug-in POM configuration. Similarly the EJB module can be included in the EAR by configuring the <*ejbModule*> element in the plug-in POM configuration and by including a dependency on the EJB module in the POM.

The inclusion of the WAR module is similar as well using <*webModule*>. However, <*webModule*> provides the additional configuration option 'contextRoot'**.** This can be used to set a Web application context root name different from the actual Web application artifact name. We'll use this option in our *'Web'* module to set the context root as '/employeeInfo'. We'll also explode this artifact in the EAR by setting the 'unpack' configuration option to true. The modified POM is shown in Listing 5. For more information on the *'maven-ear-plugin,'* go to http://maven.apache.org/plugins/maven-ear-plugin/.

Packaging & Installing an '*ear*' Module

To deploy an *'ear'* artifact to the local repository, right-click on the module's POM file in Eclipse and, in 'Run As' options, select 'Maven2 install.' Maven will compile, package, and install the *'employeeInfoEAR-1.0.ear'* artifact in the local repository along with the POM information. The same result can be achieved by running the *'mvn clean install'* command from the command prompt inside the *'employeeInfoEAR'* directory. Note that at any given point of time, the entire application can be built and installed to the local repository by executing an *'mvn install'* command from within the parent *'EmployeeInfo'* directory. This can also be achieved by right-clicking on the parent *'EmployeeInfo'* POM file in Eclipse and, in the 'Run As' options, selecting 'Maven2 install.' When this is invoked, Maven will build and install each individual child module found in the parent POM.

Figure 13

Conclusion

In this article, we attempted to explain the working nature of Maven 2 through an example. We showed how it can be used in typical J2EE application development. Maven 2 is certainly a powerful tool that significantly simplifies and standardizes build process management. By following a set of standard principles and core competencies, Maven 2 considerably increases a software developer's productivity by eliminating the grunt work typically incurred during application development. Maven 2 reuses build logic in the form of easy-to-use plug-ins and offers a cornucopia of useful features for build process management.

In this article, we only scratched the surface of Maven 2's capabilities and software developers can definitely take advantage of many other Maven 2 features like continuous integration support and project communication management. The more one uses Maven 2, the greater one appreciates its merits.

Acknowledgements

We'd like to thank our managing directors Bill Bernahl and Gail Dielman for their support and inspiration in writing this article. Our special thanks to our colleagues Todd August, Saya Alur, Huachao Li, and Prasad Nagu for spending their time reviewing the article, patiently exercising the example, and providing valuable feedback. Without their support, the article would not have been finished. Thanks to the rest of our team members who inspired us to write this article.

–Listings 1–3 are on page 24

Listing 1

```
<!DOCTYPE connector PUBLIC '-//Sun Microsystems, Inc.//DTD Connector
1.0//EN' 'http://java.sun.com/dtd/connector_1_0.dtd'>
<connector>
    <display-name>Employee Info Connector</display-name>
    <vendor-name>SomeCompany</vendor-name>
    <spec-version>1.0</spec-version>
    <eis-type>EmployeeDB</eis-type>
    <version>1.0</version>
    <resourceadapter>
        <managedconnectionfactory-class>com.somecompany.connector.Employee
InfoSPIManagedConnectionFactory</managedconnectionfactory-class>
        <connectionfactory-interface>javax.resource.cci.
ConnectionFactory</connectionfactory-interface>
        <connectionfactory-impl-class>com.somecompany.connector.EmployeeIn
foCCIConnectionFactory</connectionfactory-impl-class>
        <connection-interface>javax.resource.cci.Connection</connection-
interface>
        <connection-impl-class>com.somecompany.connector.EmployeeInfoCCICo
nnection</connection-impl-class>
        <transaction-support>NoTransaction</transaction-support>
        <config-property>
            <config-property-name>EisProductName</config-property-name>
            <config-property-type>java.lang.String</config-property-type>
            <config-property-value>Employee Info Connector</config-prop-
erty-value>
        </config-property>
        <config-property>
            <config-property-name>EisProductVersion</config-property-name>
            <config-property-type>java.lang.String</config-property-type>
            <config-property-value>1.0</config-property-value>
        </config-property>
        <config-property>
            <config-property-name>UserName</config-property-name>
            <config-property-type>java.lang.String</config-property-type>
            <config-property-value>SomeCompanyUser</config-property-value>
        </config-property>
        <config-property>
            <config-property-name>MaxConnections</config-property-name>
            <config-property-type>java.lang.Integer</config-property-type>
            <config-property-value>10</config-property-value>
        </config-property>
        <reauthentication-support>false</reauthentication-support>
    </resourceadapter>
</connector>
```

Listing 2

```
<build>
          <plugins>
                    <plugin>
                              <groupId>org.apache.maven.plugins</
groupId>
                              <artifactId>maven-jar-plugin</artifactId>
                              <executions>
                                        <execution>
                                                  <id>jarCreation</id>
                                                  <goals>

<goal>jar</goal>

                                                  </goals>
                                        </execution>
                              </executions>
                    </plugin>

                              ...

<plugin>
                              <groupId>org.apache.maven.plugins</
groupId>
                              <artifactId>maven-rar-plugin</artifactId>
                              <executions>
                                        <execution>
                                                  <id>rarCreation</id>
                                                  <goals>

<goal>rar</goal>

                                                  </goals>
                                                  <configuration>
<includeJar>false</includeJar>
                                                  </configuration>
                                        </execution>
                              </executions>
                    </plugin>
                       <plugin>
                              <groupId>org.apache.maven.plugins</
groupId>
                              <artifactId>maven-pmd-plugin</artifactId>
                    </plugin>
          </plugins>
   </build>
```

Listing 3

```
<project>
   <parent>
          <artifactId>EmployeeInfo</artifactId>
          <groupId>com.somecompany</groupId>
          <version>1.0</version>
   </parent>
   <modelVersion>4.0.0</modelVersion>
   <artifactId>ejb</artifactId>
   <packaging>ejb</packaging>
   <name>ejb</name>
   <dependencies>
          <dependency>
                    <groupId>geronimo-spec</groupId>
                    <artifactId>geronimo-spec-ejb</artifactId>
                    <version>1.0-M1</version>
                    <scope>provided</scope>
          </dependency>
          <dependency>
                    <groupId>geronimo-spec</groupId>
                    <artifactId>geronimo-spec-j2ee-connector</artifac-
tId>
                    <version>1.0-M1</version>
                    <scope>provided</scope>
          </dependency>
<dependency>
                    <groupId>com.somecompany</groupId>
                    <artifactId>connector</artifactId>
                    <version>1.0</version>
          </dependency>
   </dependencies>

                              ...

<build>
          <plugins>
                    <plugin>
                              <artifactId>maven-ejb-plugin</artifactId>
                              <configuration>
                                        <archive>
                                                  <manifest>

<addClasspath>true</addClasspath>

                                                  </manifest>
                                        </archive>
                                        <generateClient>true</gener-
ateClient>
                              </configuration>
                    </plugin>
                    <plugin>
                              <groupId>org.codehaus.mojo</groupId>
                              <artifactId>cobertura-maven-plugin</arti-
factId>
                              <version>2.0</version>
                              <executions>
                                        <execution>
                                                  <goals>

<goal>cobertura</goal>

                                                  </goals>
                                        </execution>
                              </executions>
                    </plugin>
          </plugins>
   </build>
</project>
```

CODE GEAR FROM Borland

Top 5 Reasons Why Java Developers love JBUILDER® 2007

1 Supercharged Development

Support for the latest Java technologies and AppServer™ platforms, performance tuning and advanced debugging with Optimizeit™ , simplified top-down and bottom-up Web Services and EJB development, and full Java UML 2.0 modeling with round-trip live source.

2 Fully Integrated IDE

A completely integrated IDE solution for source control, bug tracking, requirements management and project planning. In-IDE integration for live and off-line editing of all project artifacts including defects, source, project tasks and requirements. All projects are automatically synchronized when on-line.

3 Optimized Code Performance

Optimizeit™ delivers memory and CPU profiling and debugging and provides high-level performance-related data displayed in real time that allows developers to understand whether a performance issue is related to CPU, memory, or both.

4 Full UML Integration

UML 2.0 and legacy support for UML 1.4 integration that developers need and want. The full UML integration includes support for generating method-level sequence diagrams. Even code with no UML support can use LiveSource© technology, the code is the model and the model is the code. Providing advanced forward and reverse engineering at its best.

5 Increased Team Velocity

TeamInsight™ enhances collaborative development with a centralized portal that allows team members to monitor project activity for the source code repository, track recent check-ins, view quality metrics, even view live burn-down charts for project progress.

www.codegear.com

Developing Rich Internet Applications Using Swing

A solution based on OpenSwing & Spring frameworks

by Mauro Carniel

The Java development platform always provides limited support for application development based on a graphical user interface, an area where more traditional languages and integrated development environments (IDEs) such as Visual Basic or Delphi have based their success.

Today the Java SE distribution offers essentially the same features it offered in 1999 with regards to Java graphical components: the Swing toolkit. Swing provides graphical components like grids, trees, text fields, checkbox, radio button, combo-box and others. Anyway these components have changed little and not been improved in latest releases of Java. All in all Swing greatly lacks some key aspects:

- **Input fields are too simple**. They can't be compared with the advanced graphical components available in other languages. They don't support common properties like text trim, padding, text length checking, uppercase; moreover there's no numeric field, currency field, date control or calendar control. There's no relationship between the input field and internationalization settings.

  In addition, Swing components are just hard to use. For example, creating a powerful table with editing capabilities, colors, data formatting, column or row locking usually requires a lot of code, time, and skills. A more powerful and easier set of components is needed to develop rich GUIs, especially for beginner Swing programmers.
- **There's no binding mechanism between graphical components and the data model**. This means that to set or get data from or to a data model based on POJOs (Plain Object Java Object, i.e., Java Beans) requires additional coding, not provided by the Swing toolkit.
- **There's no data retrieval layer between the presentation tier and business logic/data access layers** that facilitates and normalizes data exchange with the presentation tier and other layers.

All these deficiencies don't encourage the development of enterprise applications with rich GUI content and based only on the Swing toolkit.

This gap in Java was partly covered in the past years, thanks to solutions supplied by the market in the shape of commercial and open source solutions, but without reaching any definition of a reference standard.

Commercial solutions should be discarded because they're proprietary, not directed by the market, and in plain disagreement with the "Java philosophy" of open source solutions or solutions supported and promoted by the market, led through mechanisms of sharing and the collective definition of standards such as JCP (the Java Community Process).

With regard to open source solutions, there are several Swing extensions, born to provide:

- a suite of advanced graphics components
- a binding layer between graphics controls and data model
- a framework that defines guidelines to design an application and its components and use these components correctly
- a data retrieval mechanism that (preferably) abstracts from the real location of the data (locally or remotely)

There are many free solutions available that meet some of these topics, such as JMatter, JGoodies, JDNC (Java Desktop Network Components) and its evolution, the Swing Application Framework, currently submitted for approval to JCP. There are other open source projects that fit specific issues, as those reported on the JavaDesktop portal.

However these solutions address *some issues* (like the availability of advanced graphics controls or data binding capability) but *not all the issues* that could arise in enterprise application development with rich GUI content and not always applicable on different architectures (both two- and three-tier applications, with data communications based on HTTP or SOAP or RMI or any other protocol).

Mauro Carniel is an architect at Tecnoinformatica Group. He has more than 7 years of enterprise software development experience utilizing J2EE based technologies, including JSP, JSF, Swing, EJB. He started focusing more on GUI based client/server java applications since 1998. He has a MSc in Information Technology from University of Udine, Italy.

maurocarniel@tin.it

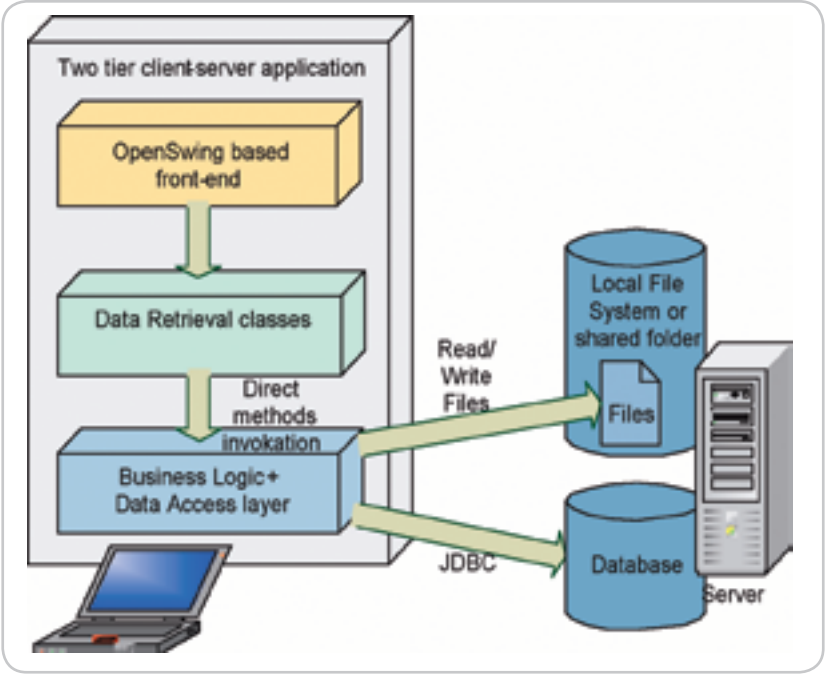


Figure 1

A good solution should include the capacity to decouple GUI development from data retrieval issues: this could facilitate application development with several architectures, such as two-layered applications (desktop applications) and three-layered applications (Rich Internet Applications). This way the same suite of graphics components and data binding mechanisms could be reused in different application architectures.

Broadly, there's a need for a *complete solution:* a framework (i.e., a set of development guidelines) *and* a set of advanced Swing components *with* data binding capabilities *and* data retrieval mechanisms not limited to a specific architecture to develop applications having rich GUI contents quickly and easily.

RIA development would become one of the possible scenarios that issues from that solution.

A complete client-side solution should integrate all these aspects and ideally fit with existing server-side layers and frameworks: in the context of server-side development, especially in the context of Web application development, valid frameworks already exist such as Spring and effective ORM (Object to Relational Mapping) layers like Hibernate, iBatis, TopLink, JDO, and JPA; hence, it's unnecessary to develop other server-side frameworks. It's better to interconnect them with the complete client-side solution.

To realize this kind of client-side solution, it's possible to fit more client-side products together, such as some of those described above, but this attempt requires skills in many products and a lot of time and isn't a suitable choice for organizations with low skills levels and limited budgets.

The OpenSwing framework addresses these issues by providing a unique and uniform client-side solution: it provides a suite of advanced graphics components that are usually powerful enough that they don't have to be extended any further. They meet the development requirements of enterprise applications with rich GUI contents. The development process becomes easier and faster by developing a GUI through IDEs' graphical designers like other non-Java RAD environments.

At the same time, this framework provides other software layers that complement the OpenSwing graphics components suite by supporting data binding, POJO-based data modelling, and remote data access by allowing the development of RIAs (three-layered client/server applications) or desktop applications (two-layered client/server applications).

OpenSwing

OpenSwing is an open source framework that can be used to develop Java applications based on Swing's front-end.

It's possible to apply this framework to develop two-tier client/server applications with an underlying database or based on other data storage devices (like files on a file system) or three-tier applications with several combinations of technologies such as RIAs (where the client and server tiers communicate through the HTTP protocol) or distributed applications (where the client and server tiers communicate through RMI – see Figures 1, 2, and 3).

The framework includes a set of class libraries that can be used to:

- Create the application front-end through a collection of advanced graphics controls that include labels, text fields, multi-line text fields, numeric fields, currency fields, calendar, grid, trees, a tree combined with a grid, lookup, gantt diagram, buttons with images, combo-box, radio buttons, checkbox, wizard panel, image panel, splash screen, dialog windows, tip of the day frame, progress bar/panel/dialog, and a licence agreement panel.

  Grid usage is especially sophisticated: it allows columns or rows locking, data pagination, columns filtering and sorting, and data exporting and the grid model is based on a list of POJOs like a tree component and a panel of graphics controls whose data model is based on a POJO.

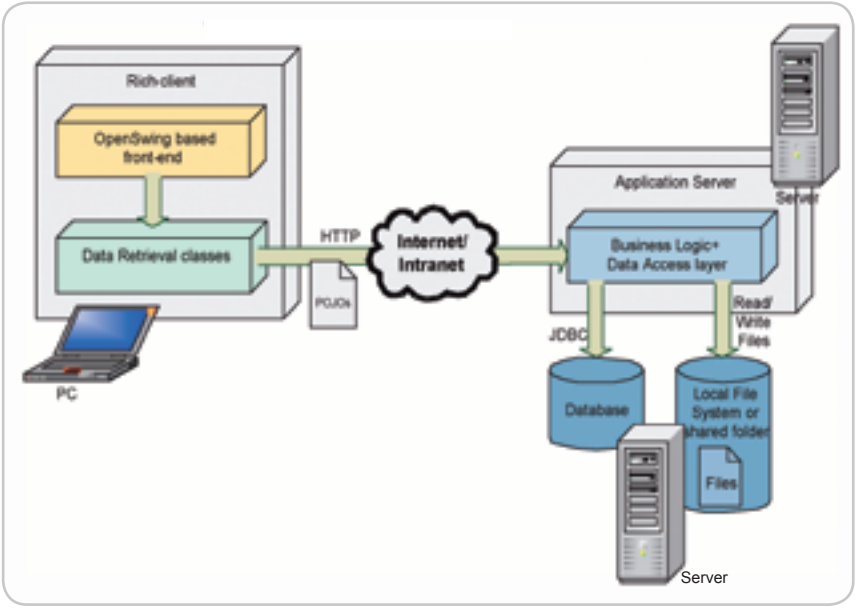


Figure 2 Three tier client server "web" application (RIA)

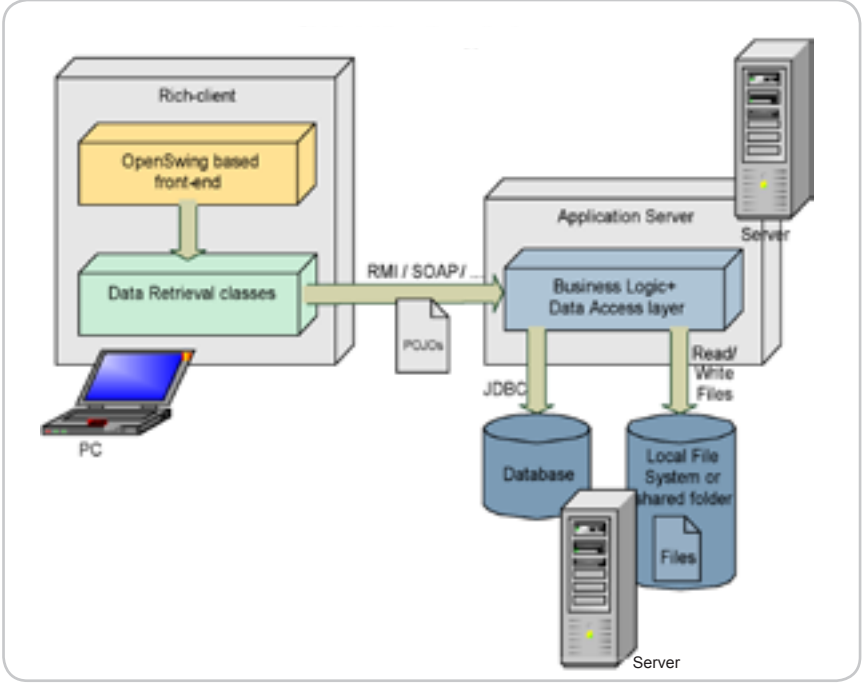


Figure 3 Distibuted three tier application

So POJO support is extended to the entire set of graphical components that compose the GUI.

The graphics controls are compliant with Java Beans specifications so they can be used in an IDE graphical designer environment like NetBeans, JBuilder, JDeveloper, or Eclipse (when combined with the Window Builder plugin) to create graphical windows by drawing them in the graphical designer as with other non-Java RAD environments.

The framework can create applications based on the SDI (Single Document Interface) or MDI (Multiple Document Interface) paradigm and internal frames including pull-down menus, tree menus, and many front-end customization levels (see Figure 4).

- Create a business logic + data-access tier through a set of utility classes that simplify the server-side development process; this utility layer can be omitted and replaced by other popular server-side frameworks, such as Spring and combined with ORM layers like Hibernate or iBatis; for these products OpenSwing provides some helper classes that makes them easy to embed.
- Create a data retrieval tier between the presentation tier (application front-end) and business logic tier. This tier can be easily extended by developing a communication layer above the standard layer offered by OpenSwing (HTTP-based) to meet specific needs (such as RMI communication with EJB, SOAP, or other communications mechanisms with server-side applications).

This framework also provides some basic features that cover many issues that usually arise in enterprise applications, such as data extraction from grids, document viewing on the most popular desktop applications (like reports on Excel, Acrobat Reader, etc.), activities logging, internationalization support (label translation, date format, decimal symbols, grouping, currency support, etc.), and authorization management according to grants owned by the connected user.

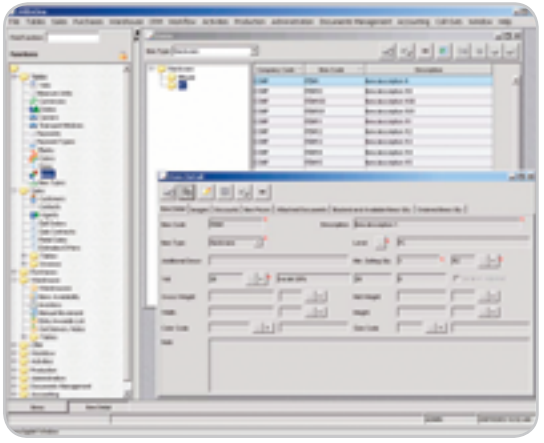


Figure 4 Example of a MDI application realized through OpenSwing graphics components

All these framework layers strongly decouple each other, so they can be used in a distinct way according to specific needs. Each layer in the framework depends on underlying layers of the framework.

The main software components of the framework are reported in the schema (see Figure 5).

Framework classes are in large part related to the graphics controls used in the presentation tier (orange color); other presentation tier classes include data retrieval, managing events fired by graphics controls, and client-side logging (green color).

The OpenSwing framework provides server-side classes too not directly connected to presentation tier classes that can be applied with three-tier applications (cian color) to simplify the realization of the Web layer and data access layer.

The OpenSwing data access layer maps value objects (POJO objects) to SQL instructions used to retrieve data from a relational database or to insert/update records. Through this layer the ORM activity (Object-to-Relational Mapping) is greatly simplified. It can be combined with the rest of the server-side layer in the case of three-tier client/server applications, or it can be combined directly with the presentation layer in two-tier client/server applications.

It's also possible to fully replace this layer with other (more efficient) ORM products such as Hibernate or iBatis.

Developing RIAs Using OpenSwing & Spring Frameworks

RIAs are three-tier applications with the same features and functionality as traditional desktop applications but typically run in a Web browser and don't require software installation.

The rise of RIAs lately has become an important topic in the Java community. Besides new technologies like AJAX and Macromedia Flex, the combination of Swing and Java Web Start has also been proposed as a RIA technology. However, HTML/AJAX or Flex technologies aren't without weakness like Swing.

A possible solution is the combination of OpenSwing and the Spring framework deployed with Java Web Start.

Spring is one of the most popular server-side frameworks for developing Java Web applications; it offers several advantages when developing a Web application, such as facilities to interconnect other technologies, like Hibernate, iBatis, JSF, and Struts.

The Spring framework has been designed to strongly decouple server-side layers that compose a Web application, like the data access layer, transaction management, and presentation layer based on Web pages.

Spring can easily be used in combination with applications having a non-HTML front-end too; in this case it's still possible to use some Spring features such as a data access layer and transaction management without using other layers such as the presentation layer based on Web pages (JSP, JSTL, Turbine, etc). Hence, Spring can be connected with GUIs based on OpenSwing.

The greater strengths of this combination are:

- Rich GUIs
- Good response time because communication between the client and server tiers is limited to data exchange
- Development time of the rich GUI is lesser than HTML/AJAX thanks to IDE graphical designer adoption and the power of adopted client-side solutions so development costs are low too
- Application installation isn't required because updating and distributing the client-side application is managed through Java Web Start that caches this layer the first time it's been downloaded
- Users can use the client-side application from any computer with an Internet connection because client-side applications communicate through the HTTP protocol with server-side applications
- Client-side applications are compatible with any operating system with a Java Virtual Machine; this isn't always true with HTML-based applications where JavaScript/HTML isn't compatible with all browsers and operating systems
- Server-side application development is based on an excellent framework: it's possible to inherit all advantages that Spring provides, such as XML-based configuration, strong decoupling between server-side layers, inversion of control, dependency injection, transactions management, object relational mapping, Aspect-oriented programming, Spring Application Context, etc.

OpenSwing provides utility classes that can be used when combining it with the Spring framework and when combining it with an ORM product, such as Hibernate or iBatis to simplify the management of client-side info passed to the ORM layer and used to drive data retrieval (filtering or sorting conditions, pagination settings, etc.).

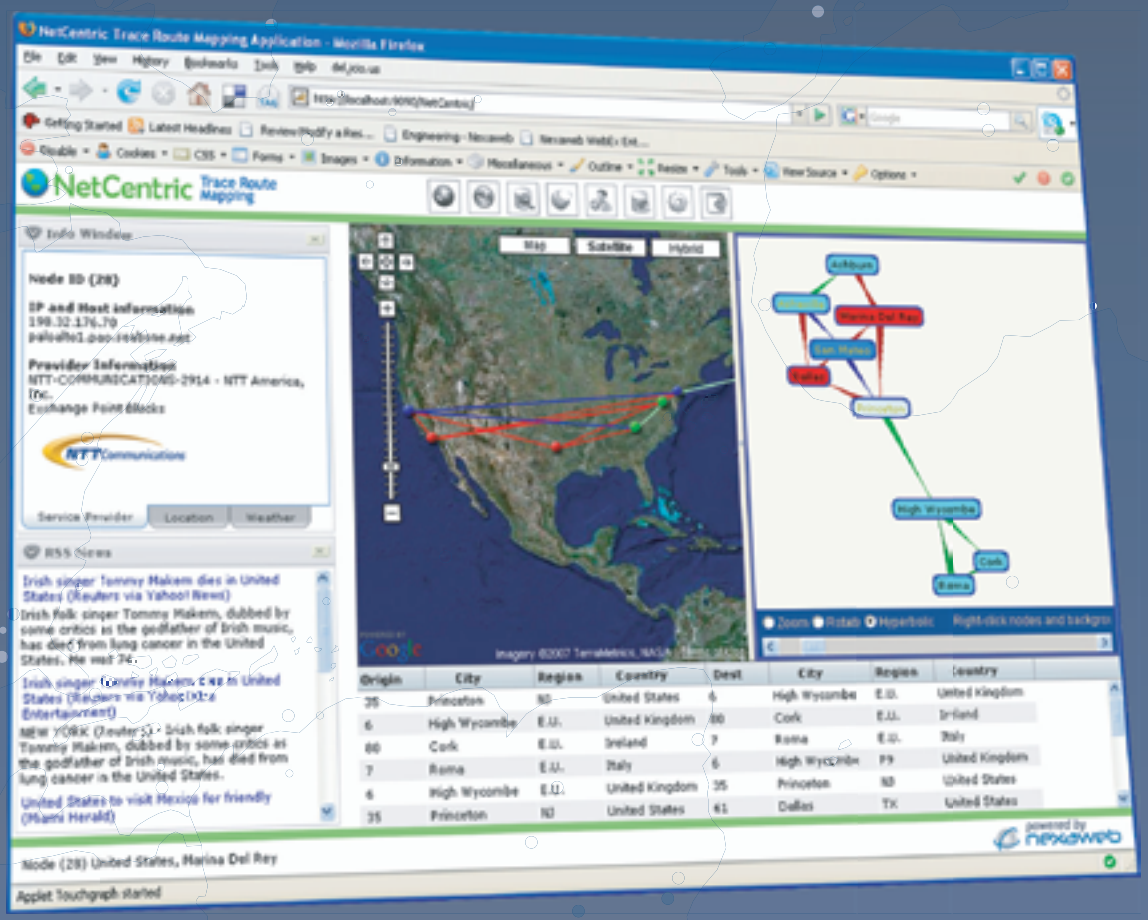
Utility classes provided by OpenSwing to embed the Spring framework consist of:

- A HandlerMapping class to support HTTP

Who's Kidding Who?

A 'Real World' enterprise mash-up doesn't mean restaurant locations and movie times overlaid with a mapping tool...

'Real World' means supporting 5,000 geographically-disbursed users who need access to mainframe applications, real-time market data, RSS feeds, and a rich charting tool that work together seamlessly, regardless of network connectivity.



Nexaweb's Enterprise Web 2.0 Platform lets you build 'Real World' enterprise mash-ups Faster, Better, and Cheaper than any other commercial or open source toolkit available today.

Don't believe it. Try us - We Dare You!

Visit us at Real World Java in New York or at dev.nexaweb.com.

nexaweb
Enterprise Web 2.0

requests originated from OpenSwing client-side classes; such HTTP requests contain serialized objects that must be managed on the server side (e.g., action class to invoke, filtering or sorting conditions, pagination settings, etc.)
- An interceptor class to support client session identification
- A ViewResolver class to support HTTP responses to send to OpenSwing client-side classes that expect POJOs as results.

With this helper classes it's possible to connect an OpenSwing client-side application having a Swing-based front-end to a server-side application based on Spring using HTTP as the communication protocol. Since only POJOs are transmitted between client and server layers, the response time is good, so this kind of application can be successfully applied in Internet-based environments such as RIAs.

Behind Spring there's usually a data access layer typically based on some ORM tool such as Hibernate or iBatis. When adopting Hibernate or iBatis there are some utility classes provided by OpenSwing that simplify the input process originated from the client side: for example, filtering or sorting conditions applied to a grid or pagination issues automatically managed through these utility classes to simplify their use inside the ORM layer (see Figure 6).

Configuring an RIA with OpenSwing and Spring

When creating an RIA with an OpenSwing-based GUI, it's possible to use a classic *DispatcherServlet* servlet provided with Spring as the unique access point for all HTTP requests; these requests come from the "ClientUtils.getData()" utility method provided by OpenSwing that can be used in a client-side application to generate requests to the server-side application: all data retrieval classes located in the client-side application will use this utility method to remotely contact the server-side via the HTTP protocol.

In any J2EE Web application there must be a defined "WEB-INF/web.xml" file; with Spring this is the first file to configure too. When combining OpenSwing with Spring, a typical web.xml content is like the one described in Listing 1.

As shown in there, a DispatcherServlet servlet class is defined; moreover, other XML files must be defined: they are the other classical files required by Spring: *applicationContext.xml* and *dataAccessContext-….xml*

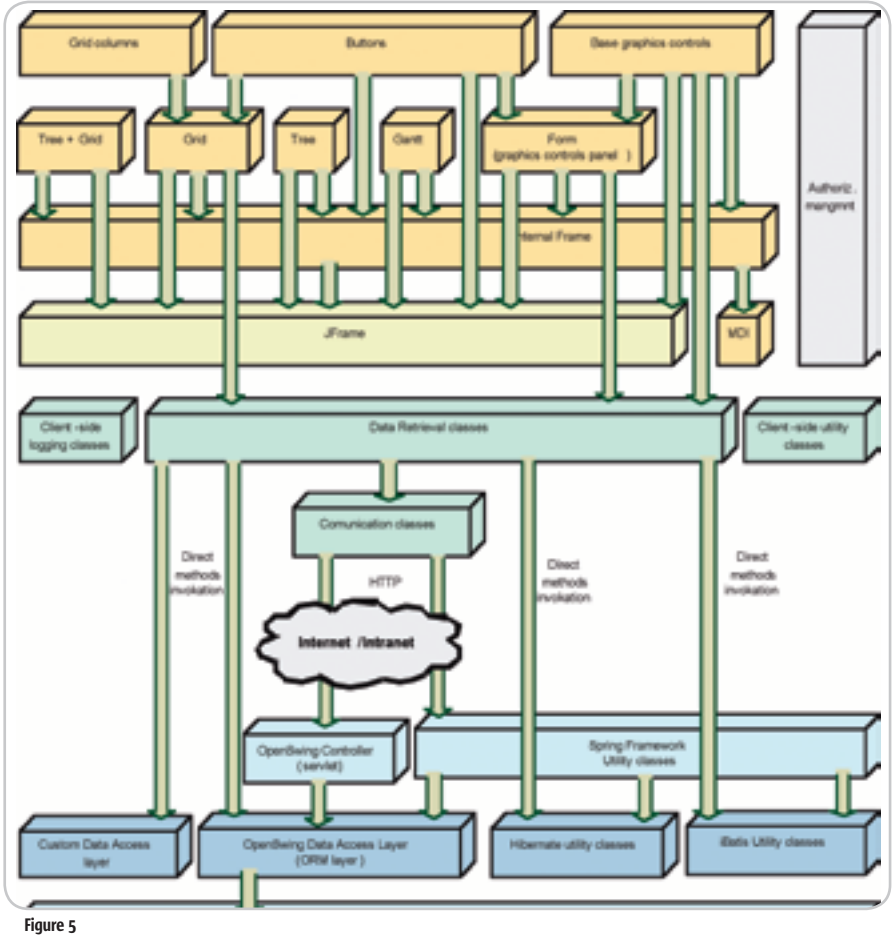


Figure 5

When defining the DispatcherServlet servlet, Spring requires another XML file be created: "xxx-servlet.xml" in which "xxx" is the name of the servlet defined in the web.xml (in our example: "controller"); Listing 2 shows the "controller-servlet.xml" file.

OpenSwing provides two main classes that must be defined in this XML file:
- *OpenSwingHandlerMapping* – This job of this class is to get all the requests coming from the DispatcherServlet: these HTTP requests always contain a serializable object of the *org.openswing.swing.message.send.java.Command* type that contains a server-side bean name to invoke; this bean is a controller type object recognized by the Spring framework; all controller type beans in a "controller-servlet.xml" file must be defined as having as an "id" the "methodName" values stored in the command object and defined in the client-side layer.

  At this point the Spring framework is the only actor: it's possible to define a facade, dao objects, transactions, and any other Spring component. Consequently, it's possible to include any technology that Spring allows to connect: ORM layers (such as Hibernate or iBatis or TopLink or JPA), EJB, etc.

  The only constraint to respect is that the value to return to the client-side must be an object that extends *org.openswing.swing.message.response.java.Response*. If the client request is generated by a grid control or a lookup then the return value must be a VOListResponse; if the client request is generated by a form then the return value must be a VOResponse type as with any other application based on OpenSwing and not on Spring (independent of the number of tiers, two or three tiers).
- *OpenSwingViewResolver* – This class returns a Response type object to the client side; this object has been generated in the server-side application and is given back through HTTP by serializing the object.

  So OpenSwingViewResolver doesn't render a Web page. It serializes objects for the client-side application.

It's possible to include any kind of Interceptor object in a Spring configuration as in any application based on Spring. The *OpenSwingHandlerMapping* class provided by OpenSwing always extracts the command serialized object from the request and stores it as the request's attribute named *OpenSwingHandlerMapping.COMMAND_ATTRIBUTE_NAME*. In this way the command object is available to all interceptor objects added to the application.

Optionally OpenSwing provides an interceptor class named *SessionCheckInterceptor* that

Web 2.0 and Rich Internet Apps

CALL FOR PAPERS NOW OPEN!

September 24-26, 2007

Santa Clara Convention Center
Hyatt Regency Silicon Valley
Santa Clara, CA

To Register
Call 201-802-3020 or
Visit www.AJAXWorld.com

Hurry! Limited Seating
This Conference Will Sell-Out!

Providing developers and IT managers alike with comprehensive information and insight into the biggest paradigm shift in website design, development, and deployment since the invention of the World Wide Web itself a decade ago.

On September 24-26, 2007 over 1,000 developers, architects, IT managers, and software professionals of every stripe will be converging in Santa Clara, CA to attend the West coast AJAXWorld Conference & Expo -- the most comprehensive meeting on the most significant technology subjects of recent times: AJAX, Rich Internet Apps & Web 2.0.


Hyatt Regency Silicon Valley
Santa Clara, CA

Join Over 3,600 Early AJAX Adopters Who Attended the #1 Web 2.0 Event in the World!

Experience AJAX, RIA, and Web 2.0 Knowledge and Best Practices at AJAXWorld Conference and Expo 2007

Delegates will hear first-hand from the creators, innovators, leaders and early adopters of AJAX, and a slew of leading vendors will provide sneak-peeks of the very latest frameworks, tools, products and applications during the conference, which will have over 100 sessions and presentations given by 125 different speakers - the largest AJAX-focused speaker faculty ever assembled in one place at one time.

AJAXWorld Conference & Expo 2007 West will be jump-started with a full one-day "AJAX University Boot Camp," followed by the Main Conference and Expo over the following 2 days.

Highlights Include:

- ***100+ conference speakers*** *selected from among the world's leading AJAX technologists and practitioners, including high-level IT executives, industry thought leaders, VCs and analysts*
- ***Topical, Interactive "Power Panels"*** *simulcast on SYS-CON.TV*
- ***Demos from 30+ vendors*** *both in General Session and in the Exhibit Hall/AJAX Showcase*
- ***Pre-Conference "AJAXWorld University Bootcamp"*** *for those wanting an all-day, hands-on encounter with AJAX*
- ***Leading-edge sessions*** *on issues like Offline AJAX, Mobile AJAX, Enterprise AJAX, AJAX Security, Desktop AJAX, Semantic Mash-Ups, Next-Generation SaaS, JavaScript & XML, Google Web Toolkit, Adobe Apollo, Adobe Flex, AJAX RIA GUIs, and Appropriate vs Inappropriate Use of AJAX.*

Early Bird Price: $1,495
(Register Before May 25, 2007)

SYS-CON EVENTS
For more great events visit www.EVENTS.SYS-CON.com

VISIT WWW.AJAXWORLD.COM FOR THE MOST COMPLETE UP-TO-DATE INFORMATION

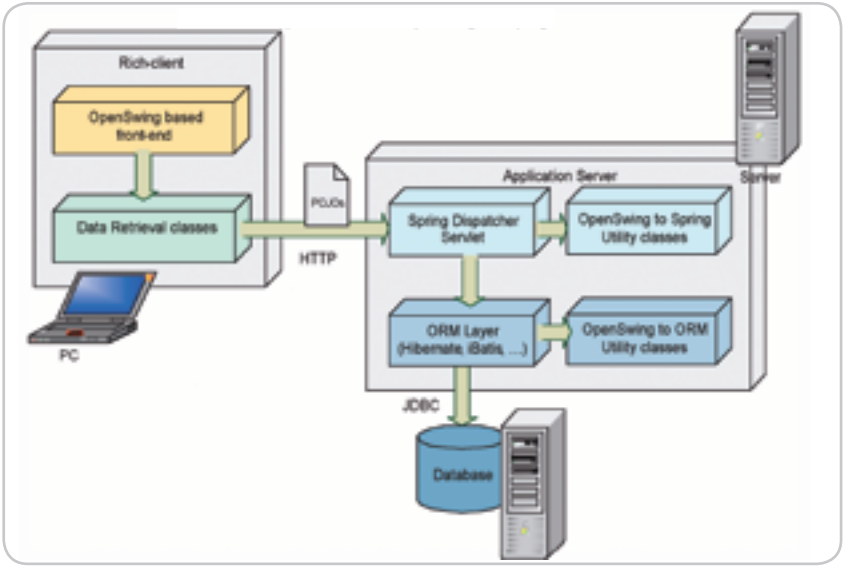


Figure 6 Rich Internet Application based on OpenSwing and Spring

could be included in "controller-servlet.xml" file: this interceptor checks each HTTP request coming from the client side; SessionCheckInterceptor dispatches requests only if the command object contains a session identifier previously stored in the servlet context (when the client was authenticated); if a command object doesn't contain a session identifier or it contains a session identifier not stored in the servlet context then the request is rejected.

The choice of storing session identifiers in the *servlet* context instead of in the *session* context bound to the client derives from the nature of three-tier client/server applications based on the Swing front-end: these applications may be started without a browser (for example, by directly using Java Web Start), so that the traditional means of client session identification (cookies or URL rewriting) can't be applied outside the browser. Consequently, session info bindable to a client must be stored in the servlet context and fetched starting from a client session identifier that must be sent from client to server in each command object.

SessionCheckInterceptor class extracts the client session identifier from the request (through the Command.getSessionId method) and checks in the ServletContext if this identifier is stored. Session identifiers are stored in a HashSet whose attribute name is *OpenSwingHandlerMapping. USERS_AUTHENTICATED*; if the identifier is stored then the interceptor returns true, otherwise it returns false and gives an *org.openswing.swing.message.response. java.ErrorResponse* object back to the client.

Resources

- JMatter - http://jmatter.org/
- JGoodies - http://www.jgoodies.com/
- JDNC – Java Desktop Network Components: https://jdnc.dev.java.net/
- JSR 296 - Swing Application Framework: http://jcp.org/en/jsr/detail?id=296
- JavaDesktop: http://community.java.net/javadesktop/
- OpenSwing Framework: http://oswing.sourceforge.net
- Spring Application Framework: www.springframework.org
- Hibernate Framework: www.hibernate.org/
- iBatis: http://ibatis.apache.org

Listing 1

```
<?xml version="1.0" encoding="UTF-8"?>
    <!DOCTYPE web-app
    PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.2//EN"
    "http://java.sun.com/j2ee/dtds/web-app_2_2.dtd">
    <web-app>
        <context-param>
 <param-name>log4jConfigLocation</param-name>
 <param-value>/WEB-INF/log4j.properties</param-value>
        </context-param>
        <context-param>
 <param-name>contextConfigLocation</param-name>
 <param-value>/WEB-INF/dataAccessContext-local.xml /WEB-INF/application-
Context.xml
        </param-value>
        </context-param>

        <listener>
    <listener-class>org.springframework.web.context.
ContextLoaderListener</listener-class>
        </listener>

        <servlet>
            <servlet-name>controller</servlet-name>
            <servlet-class>org.springframework.web.servlet.
DispatcherServlet</servlet-class>
            <load-on-startup>2</load-on-startup>
        </servlet>

        <servlet>
           <servlet-name>JnlpDownloadServlet</servlet-name>
           <servlet-class>com.sun.javaws.servlet.JnlpDownloadServlet</
servlet-class>
        </servlet>

        <servlet-mapping>
            <servlet-name>controller</servlet-name>
            <url-pattern>/controller</url-pattern>
        </servlet-mapping>

        <servlet-mapping>
            <servlet-name>JnlpDownloadServlet</servlet-name>
            <url-pattern>/demo18.jnlp</url-pattern>
        </servlet-mapping>

    </web-app>
```

Listing 2

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN 2.0//EN" "http://www.spring-
framework.org/dtd/spring-beans-2.0.dtd">
<beans>
    <bean id="handlerMapping"
          class="org.openswing.springframework.web.servlet.handler.
OpenSwingHandlerMapping">
<!--
        <property name="interceptors">
            <list>
                <ref bean="sessionCheckInterceptor"/>
            </list>
        </property>
-->
    </bean>
<!--
    <bean id="sessionCheckInterceptor"
          class="org.openswing.springframework.web.servlet.handler.
SessionCheckInterceptor">
        <property name="loginMehodName" value="login"/>
    </bean>
-->
<!-- ====== DEFINITION OF LOGIN CONTROLLER ====== -->
    <bean id="login" class="demo18.server.LoginController">
        <property name="username" value="admin"/>
        <property name="password" value="guest"/>
    </bean>

<!-- ====== VIEW DEFINITIONS ====== -->

    <bean id="swingViewResolver"
          class="org.openswing.springframework.web.servlet.view.
OpenSwingViewResolver">
    </bean>


<!-- ====== DEFINITIONS OF PUBLIC CONTROLLERS ====== -->

…

</beans>
```

JAVA DEVELOPER'S JOURNAL Advertiser Index

Advertiser	URL	Phone	Page
AJAXWorld Conference & Expo	www.ajaxworld.com	201-802-3022	31
Altova	www.altova.com	978-816-1600	Cover II
Business Objects	www.businessobjects.com/rareoccurrence	888-333-6007	11
CodeGear	www.codegear.com	888-233-2224	25
Infragistics	www.infragistics.com/jsf	800-231-8588	4
InterSystems	www.intersystems.com/ja1p		Cover IV
Java Developer's Journal	www.jdj.sys-con.com	888-303-5282	33
Nexaweb	dev.nexaweb.com	781-345-5500	29
Red Hat Middleware	www.redhat.com	866-273-3428 x45555	13
SAP TechEd '07	www.sapteched.com		7
Software FX	www.softwarefx.com	561-999-8888	Cover III

General Conditions: The Publisher reserves the right to refuse any advertising not meeting the standards that are set to protect the high editorial quality of ***Java Developer's Journal***. All advertising is subject to approval by the Publisher. The Publisher assumes no liability for any costs or damages incurred if for any reason the Publisher fails to publish an advertisement. In no event shall the Publisher be liable for any costs or damages in excess of the cost of the advertisement as a result of a mistake in the advertisement or for any other reason. The Advertiser is fully responsible for all financial liability and terms of the contract executed by the agents or agencies who are acting on behalf of the Advertiser. Conditions set in this document (except the rates) are subject to change by the Publisher without notice. No conditions other than those set forth in this "General Conditions Document" shall be binding upon the Publisher. Advertisers (and their agencies) are fully responsible for the content of their advertisements printed in ***Java Developer's Journal***. Advertisements are to be printed at the discretion of the Publisher. This discretion includes the positioning of the advertisement, except for "preferred positions" described in the rate table. Cancellations and changes to advertisements must be made in writing before the closing date. "Publisher" in this "General Conditions Document" refers to SYS-CON Publications, Inc.

This index is provided as an additional service to our readers. The publisher does not assume any liability for errors or omissions.

Estival JSRs

Changes, new chair of the JCP

Onno Kluyt

It's been busy at the JCP for Spec Leads, Expert Groups, and Executive Committees over the summer. Quite a number of new proposals were submitted and were approved to be developed as JSRs; even more moved to new development stages, drawing closer to the finish line. And, I might add, that all happened at a balanced pace, meeting both the initial JSR development commitment and satisfying the rigors of developing complete RIs and TCKs in most the cases. Here are some of them.

Portlet Specification 2.0, JSR 286, posted its Public Review Draft. This JSR sets out to develop a binary-compatible updated version of JSR 168, Portlet Specification 1.0. It plans to add functionality that was not addressed in the first version of the specification. If you want to check out the details of the draft, you have until August 21 to review it and send your comments to the Spec Lead and Expert Group. The JCP EC will vote on the Public Draft between August 21–August 27 (http://jcp.org/aboutJava/communityprocess/pr/jsr286/).

JSR 196, Java Authentication Service Provider Interface for Containers, was recently voted on by the JCP Java SE/EE EC and approved as a finalized standard with 10 votes out of 16. You can view the ballot results at http://jcp.org/en/jsr/results?id=4307. In the Spec Lead's words, Ron Monzillo, Sun Microsystems, the specification defines a standard interface by which authentication modules may be integrated with containers in such a way that these modules may establish the authentication identities used by containers.

At the beginning of its development, *JSR 297, Mobile 3D Graphics API 2.0*, from Nokia Corporation, with Tomi Aarnio at the helm, published its Early Draft Review, which can be downloaded at http://jcp.org/aboutJava/communityprocess/edr/jsr297/. The Spec Lead and Expert Group tell us on the JSR Public Page that "the proposed specification is a new revision of JSR-184, Mobile 3D Graphics API for J2ME" and promises "to extend and enhance Mobile 3D Graphics to better leverage state-of-the-art hardware."

Led by a Star Spec Lead, Jaana Majakangas of Nokia, *JSR 293, Location API 2.0*, published its Public Draft in June. You can send your comments to the JSR's leads until September 4. We learn from the JSR Public Page that JSR 293 is intended to be a successor to JSR 179 and more. Visit http://jcp.org/aboutJava/communityprocess/pr/jsr293/index.html to learn more about the approach of this JSR.

A proposal for a new release of the Java Platform, Enterprise Edition 6, was recently submitted by Spec Leads Bill Shannon and Roberto Chinnici of Sun Microsystems and was approved on July 16 by the JCP Java SE/EE ECs. The new release of Java EE 6 plans to focus on extensibility to allow third-party libraries/extensions to fit easily with the rest of the Java EE programming model. It also sets out to develop rules for profiles and define the Web profile for Web application developers. You'll find a detailed description of the JSR on the Public Page at http://jcp.org/en/jsr/detail?id=316.

Linda DeMichiel proposed a new persistence project, *JSR 317, Java Persistence API*, on behalf of Sun Microsystems. The purpose of the Java Persistence 2.0 specification as described on the JSR Public Page is to augment the Java Persistence API to include further features requested by the community. Visit the JSR Public Page for details on the aspects the Expert Group plans to consider (http://jcp.org/en/jsr/detail?id=317).

Another JSR from Sun Microsystems, *Enterprise JavaBeans 3.1, JSR 318*, is on the Java SE/EE EC approval ballot at the time of writing (http://jcp.org/en/jsr/detail?id=318). The Enterprise JavaBeans 3.1 specification sets out to accomplish two things: further simplify the EJB architecture by reducing its complexity from the developer's point of view and add new functionality in response to the needs of the community.

JSR 283, Content Repository for Java Technology API Version 2.0, from Day Software, entered Public Review recently. The specification is the successor to JSR 170, the first standard for content repositories. JSR 283 adds several enhancements to the API to make it easier for companies to simplify their repository architecture, increase efficiency, and reduce cost. To download the Public Draft, go to http://jcp.org/aboutJava/communityprocess/pr/jsr283/. You have until September 4 to send in your comments to the Spec Lead.

Nokia and Motorola, co-Spec Leads of *JSR 272, Mobile Broadcast Service API for Handheld Terminals*, published the Proposed Final Draft 2 of the specification at http://jcp.org/en/jsr/stage?listBy=proposed. The JSR is targeted at defining a common Java API to control and access digital broadcast content from mobile devices.

JSR 256, Mobile Sensor API, has an updated Final Release that you can access at http://jcp.org/en/jsr/stage?listBy=final. This is another Java ME JSR from Nokia. It fills the need for a standard way of manipulating sensors for Java ME applications. To accomplish this, it has defined a unified way to manage sensors and access sensor data.

Nakina Systems, a manufacturer of telecom network management systems, the Spec Lead of *JSR 254, OSS Discovery API*, published the Proposed Final Draft for the specification. The JSR is essentially a standardization platform for OSS components and an alternative to vendor-proprietary APIs. Check the draft at http://jcp.org/en/jsr/stage?listBy=proposed and send your feedback to the Spec Leads.

Two individual developers, Werner Keil and Jean-Marie Dautelle, partnered as co-Spec Leads to drive the development of *JSR 275, Units Specification*. They recently published the Early Draft Review of the spec, which you can get and review from http://jcp.org/aboutJava/communityprocess/edr/jsr275/index.html.

JSR 225, XQUERY API, published its Public Review Draft 2. If you are interested in XML and XML-related technologies, check it out at http://jcp.org/aboutJava/communityprocess/pr/jsr225/index.html. The JSR is spearheaded by Oracle's Jim Melton and sets out to develop a standard for querying XML data.

JSR 142, OSS Inventory API, published its Maintenance Draft Review 3 at http://jcp.org/aboutJava/communityprocess/maintenance/jsr142/index3.html. Developed by MetaSolv Software, the specification addresses the need to provide standardization conventions that allow interoperability of OSS components and reduce the cost of their integration in an end-to-end OSS solution.

This summer brings more changes to the JCP. If you checked the JCP.org Web site recentlyand my blog (http://jroller.com/page/OnnoKluyt), you know by now that I'm stepping down as Chair of the JCP and moving on to new roles and responsibilities within Sun. Patrick Curran, who is a veteran of the software industry and has a long-standing record in conformance testing, is taking over from me as Chair of the JCP. Congratulations, Patrick!

Stay tuned for news and comments from me in SYS-CON publications as I step into my new role.

Onno Kluyt is director of the Community Growth group at Sun Microsystems and the Chair of the JCP.

onno@jcp.org