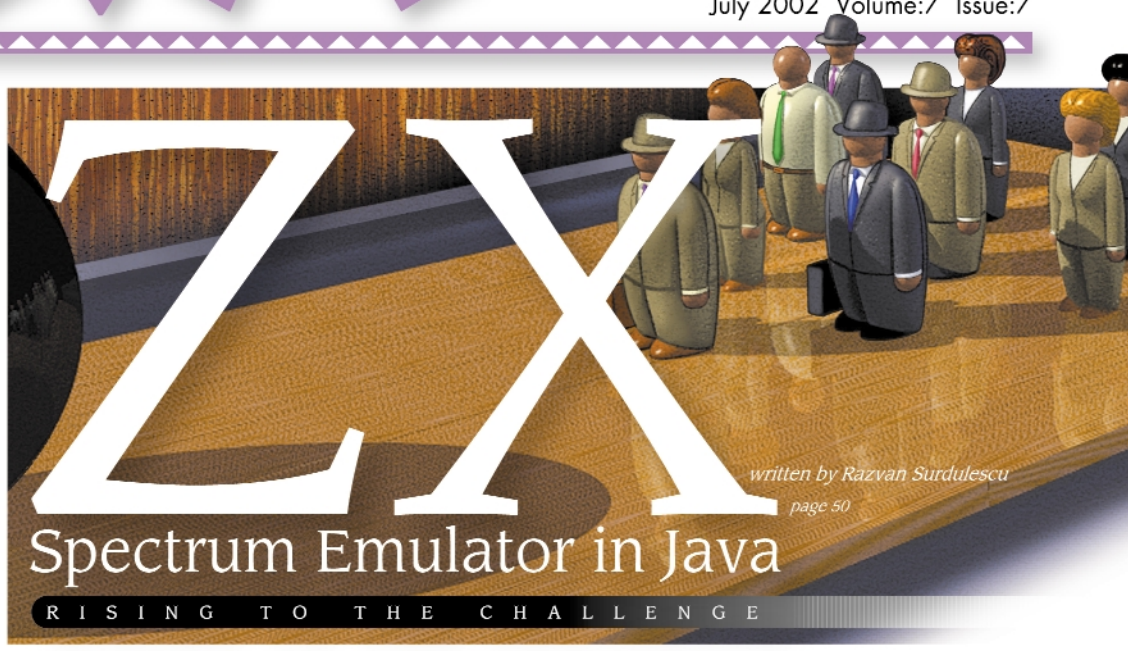
JAVA™ DEVELOPER'S JOURNAL

The World's Leading Java Resource

July 2002 Volume:7 Issue:7

JAVADEVELOPERSJOURNAL.COM

SYS-CON MEDIA



ZX Spectrum Emulator in Java

written by Razvan Surdulescu page 50

RISING TO THE CHALLENGE

JAVA DEVELOPER'S JOURNAL

INTERNATIONAL ADVISORY BOARD

• CALVIN AUSTIN *(Lead Software Engineer, J2SE Linux Project, Sun Microsystems),* • JAMES DUNCAN DAVIDSON *(JavaServlet API/XMP API, Sun Microsystems),* • JASON HUNTER *(Senior Technologist, CollabNet),* • JON S. STEVENS *(Apache Software Foundation),* • RICK ROSS *(President, JavaLobby),* • BILL ROTH *(Group Product Manager, Sun Microsystems),* • BILL WILLETT *(CEO, Programmer's Paradise)* • BLAIR WYMAN *(Chief Software Architect IBM Rochester)*

EDITORIAL

EDITOR-IN-CHIEF: ALAN WILLIAMSON
EDITORIAL DIRECTOR: JEREMY GEELAN
EXECUTIVE EDITOR: NANCY VALENTINE
J2EE EDITOR: AJIT SAGAR
J2ME EDITOR: JASON R. BRIGGS
J2SE EDITOR: JASON BELL
PRODUCT REVIEW EDITOR: JIM MILBERY
FOUNDING EDITOR: SEAN RHODY

PRODUCTION

VICE PRESIDENT, PRODUCTION AND DESIGN: JIM MORGAN
ASSOCIATE ART DIRECTOR: LOUIS F. CUFFARI
EDITOR: M'LOU PINKHAM
MANAGING EDITOR: CHERYL VAN SISE
ASSOCIATE EDITORS: JAMIE MATUSOW, GAIL SCHULTZ, JEAN CASSIDY
ASSISTANT EDITOR: JENNIFER STILLEY
ONLINE EDITOR: LIN GOETZ
TECHNICAL EDITOR: BAHADIR KARUV, PH.D.

WRITERS IN THIS ISSUE

BILL BALOGLU, CARL BARRATT, JASON BELL, JASON BRIGGS, MICHAEL DEASY, BILL DETTELBACK, KEN GREENWOOD, JOE MITCHKO, BILLY PALMIERI, NORMAN RICHARDS, AJIT SAGAR, ERIC SIMMERMAN, JOSEPH SMITH, RAZVAN SURDULESCU, LEANDER VAN ROOIJEN, ALAN WILLIAMSON, BLAIR WYMAN

SUBSCRIPTIONS:

FOR SUBSCRIPTIONS AND REQUESTS FOR BULK ORDERS, PLEASE SEND YOUR LETTERS TO SUBSCRIPTION DEPARTMENT

SUBSCRIPTION HOTLINE: SUBSCRIBE@SYS-CON.COM

COVER PRICE: $5.99/ISSUE
DOMESTIC: $49.99/YR. (12 ISSUES)
CANADA/MEXICO: $79.99/YR. OVERSEAS: $99.99/YR.
(U.S. BANKS OR MONEY ORDERS). BACK ISSUES: $10/EA., INTERNATIONAL $15/EA.

EDITORIAL OFFICES:

SYS-CON MEDIA 135 CHESTNUT RIDGE RD., MONTVALE, NJ 07645
TELEPHONE: 201 802-3000 FAX: 201 782-9600

JAVA DEVELOPER'S JOURNAL (ISSN#1087-6944) is published monthly (12 times a year) for $49.99 by SYS-CON Publications, Inc., 135 Chestnut Ridge Road, Montvale, NJ 07645. Periodicals postage rates are paid at Montvale, NJ 07645 and additional mailing offices. POSTMASTER: Send address changes to: JAVA DEVELOPER'S JOURNAL, SYS-CON Publications, Inc., 135 Chestnut Ridge Road, Montvale, NJ 07645.

Alan Williamson Editor-in-Chief

Java in a Flash!

When I wrote my last editorial I was on a plane to Toronto. What I neglected to tell you was where I was off to after Toronto. It was to Redmond, Washington, as the guest of Microsoft, where they showed me the virtues of their .NET framework. It was a very interesting visit and I learned a lot. I'm in the throes of writing up my report on the whole shebang and once I have my facts straight, I'll publish them in *JDJ*. So keep an eye out next month for that.

By the time you've read this, the World Cup will be over and another country will be heralded as the greatest football nation for the next four years, until they thrash it out again in 2006 in Germany. Why do I draw attention to this you ask? Well, it's a great test of the technology we're all building. It's at times like these that you realize just how powerful and far reaching the Internet has become. No matter how much capacity sites prepare themselves for, sometimes it's just not enough. Take the BBC Web site. On the first day of the World Cup they experienced some 8 million hits, three times more than normal traffic. Some users complained of being locked out, but on the whole it coped.

What I find wonderful about this tale is that a lot of the BBC site is run with Java technology – a great success story about the power of Java. A shot across the bow of the anti-Java brigade I say! While this sort of traffic peak is something the majority of developers will never experience, it is nice to know that should such a flood come our way, we've chosen a technology that's beautifully scalable.

• • •

This month we have a new J2SE editor taking over the reins from Keith Brown. I bid farewell to Keith and thank him for his insight over the last few months. He'll still be popping up from time to time when his schedule permits, so you haven't heard the last of him. The J2SE mantel will be taken up by Jason Bell, a *JDJ* writer of old, who will give his slant on the whole Java space. That's two Jasons we have on the editorial staff…it's an invasion!

One of the best parts of this job is talking to you. Over time I have had some wonderful conversations with people from all walks of life – people now engaged in the world of Java, with origins from the strangest of places. For example, I had a delightful exchange with a Bill Reister who took me into the world of flying military jets. Another exchange was with a Razvan Surdulescu who came to us with a great idea for a story that we all instantly jumped at. Razvan has written an emulator for the ZX Spectrum in Java. His proposal got my vote for many reasons. I owe my current life choice to that little Z80-based machine. When I was just 11-years old it whetted my appetite for the world of computers.

How many of you remember programming in those days? It's funny to look at it now, especially when we look at the power Java has laid before us. For example, for a long time I couldn't figure out how programs could ever work without the notion of line numbers! Remember choosing 10, 20, 30…giving yourself enough spacing in case you had to add in additional statements.

For those of you who are aware of the Spectrum, check out Razvan's article this month; I think you'll enjoy the trip down memory lane. For those of you who aren't familiar with that machine, read it anyway; it's a great piece on emulation.

Until next month.

Author Bio

alan@sys-con.com

Alan Williamson is editor-in-chief of Java Developer's Journal. *During the day he holds the post of chief technical officer at n-ary (consulting) Ltd, one of the first companies in the UK to specialize in Java at the server side. Rumor has it he welcomes all suggestions and comments.*

J2EE INDEX

Ajit Sagar J2EE Editor

Team Spirit

Here's a short pop quiz: Have you ever built an application in J2EE and taken it through the entire product life cycle? Or, for that matter, any distributed computing application? If the answer is "Yes," then answer this one: Have you handled all the facets of the application on your own – as a one-man team? If you answered "Yes" to both questions, my response is: I don't believe you. You can do one or the other, but not both, if we're talking about a real-world application, that is.

J2EE offers a platform for developing applications whose components or subsystems can be distributed across the different tiers of the computing network. The obvious advantage is the decoupling of the programming logic that leads to reusable and scalable solutions. The other main advantage is that development projects can structure their teams so that each member is assigned to develop the subsystem that utilizes his or her particular skillset. The operative word here is *team*. J2EE projects are team-oriented projects.

Of course, J2EE development can mean different things to different folks. There's a plethora of configuration options for application components and a variety of APIs that can be applied to their development. Sun's Java Blueprints outline the different ways to skin the *n*-tier. It all depends on the business requirements of your application, the technologies available at your company (or the ones that your corporation is willing to invest in), and the external systems you will have to integrate with.

It's true that a J2EE application can be built by a one-man team. For instance, if you're building the Pet Store application, the airline reservation application, or the typical bank account two-phase-commit example, you can do it yourself. Or if your application consists of a few business flows built on simple business logic and textbook data schemas, one or two developers with similar skillsets will suffice.

However, the colloquial interpretation of a J2EE application is one that uses the J2EE object model – EJB. Typically, EJBs apply to large-scale business applications, and an EJB-based project requires a mix of skillsets. In fact, the J2EE application server vendors have cornered the market on the J2EE frameworks and component containers. The Java IDEs offer the code development environment. The data and business modeling tools enable design and analysis. The vendors offer a mix of options for developing J2EE applications, all based on the J2EE Blueprints.

To effectively utilize the tools in the market, you need to first define an appropriate team for your application. The skillsets you'll need can be broadly classified into the following areas:

- Front-end graphics
- JSP/servlet/HTML/Web development
- EJB/middle-tier Java components
- Database and application server administration

The following is an example of an EJB-based project. If your application requires development using Web services, JMS, JCA, or other Java platform APIs, you'll have to define your team accordingly. While it's possible to "overload" developers to work across the different areas, in a complex application the pragmatic approach is to assign developers to specific areas and migrate them to other areas when needed.

Fortunately, there are resources in the market that will aid you in defining the appropriate architecture. The J2EE vendors have mature offerings that help you develop reusable components, as well as environments that enable you to migrate components across the tiers. Several online and print resources for applying good design patterns and best practices are available too.

ajit@sys-con.com

Author Bio

Ajit Sagar is the J2EE editor of JDJ and the founding editor of XML-Journal. A lead architect with a software solutions firm based in Dallas, he's well versed in Java, Web, and XML technologies.

Certificate Authorization in Your J2EE PKI

Integrate self-certified certificates with J2EE systems via JSSE

WRITTEN BY
ERIC SIMMERMAN

When a client recently requested secure communication among multiple platform boxes distributed across three continents, I decided to leverage the 100% Java-based security available via Java Secure Socket Extension.

JSSE requires trusted certificates for authentication services, but my client had no Public Key Infrastructure (PKI) in place for certificate generation and distribution. So I built a PKI implementation where my client acted as the certificate authority and then integrated this PKI with J2EE systems via JSSE to provide secure communication services. In this article, I'll demonstrate how you can do the same. (The source code is available on the ***JDJ*** Web site, www.sys-con.com/java/sourcec.cfm.)

PKI/JSSE Overview

If you already have a PKI in place or have other experience with key-based cryptography, you may want to skip this section. For those new to the subject, the following brief overview provides some of the key terms and acronyms used throughout this article. There's a lot to absorb here, but the actual application of these concepts in the following sections should make matters more clear.

One of the additions to the upcoming release of J2SDK 1.4 will be Java Secure Socket Extension 1.0.2. Currently available as an optional add-on, JSSE is a set of Java packages that enables secure Internet communications via Secure Sockets Layer (SSL) v3 and Transport Layer Security (TLS) 1.0 protocols.

Netscape developed SSL in 1994 and subsequently transferred control of the protocol to the Internet Engineering Task Force. The IETF renamed SSL to Transport Layer Security (TLS), and released their first specification in January 1999. TLS 1.0 is a modest upgrade to the most recent version of SSL, version 3.0, and the differences between the two are minor.

SSL and TLS use public key cryptography to provide authentication, secret key cryptography to provide privacy, and a message authentication code to provide data integrity. While in this article I focus on the use of a key pair for authentication purposes, all these cryptographic processes require only one.

One key in the pair is made public and the other is held strictly private. For authentication purposes, the public key in a key pair is associated with a certificate. A certificate in a PKI implementation is an electronic document used to identify a communicating entity by its association with a public key. Since certificates are used to address the problem of impersonation, their distribution must be governed by a trustworthy entity. These entities are known as certificate authorities (CAs).

When you participate in a client-authenticated TLS or SSL conversation, you receive your counterparticipant's certificate. The certificate specifies the identity of your counterparticipant and guarantees it by providing a certificate authority's signature. The CA in this case acts much like a notary. If you trust the notary, you can trust the certificate. CAs for the uncontrolled communication encouraged by the World Wide Web are typically independent third parties who charge hundreds of dollars for the issuance of a single certificate. This expense is not due to the cost of producing a certificate, but to the costs involved in ensuring that an entity requesting a certificate is in fact who it claims to be.

In an enterprise system where multiple communicating parties are controlled by a single entity, it makes little sense to incur this expense. You can save time and money while maintaining internal control of your PKI by establishing yourself as the CA in your secure communications. So let's get to it.

> "You can save time and money while maintaining internal control of your PKI by establishing yourself as the CA in your secure communications"

Becoming a CA

JSSE was not included in Java Development Kits prior to version 1.4, so you may need to install JSSE as an optional extension to your Java platform. Luckily, the JSSE distribution includes explicit installation instructions, making this step a breeze. Next, you'll need an SSL toolkit capable of issuing X.509 certificates. I recommend OpenSSL because it's open source, well documented, and free. There are several other options available including man-

aged services, but I'll be using OpenSSL for demonstration purposes.

Once you have the necessary software installed, the next step is to create your certificate authority private key and certificate. I'll create mine with the single-line command:

```
openssl req -new -x509 -newkey
  rsa:2048 -config openssl.cnf -key-
out rootCAKey
 -out rootCACert -days 3650 -rand
  "install.log:sunnyday.gif"
```

The first option "req" signifies that we're performing an X.509 certificate management operation, followed by "new" and "x509" showing that we're creating a new certificate without a certificate request. The "newkey" parameter specifies the type of private key along with its size in bits. The "days" parameter specifies that this certificate will be valid for 10 years, and the "rand" parameter points to a couple of files used to help randomize my key generation.

Once you execute this command, you'll be prompted for a pass phrase that will be used to encrypt and decrypt your new private key. When choosing a pass phrase and restricting access to your CA private key, keep in mind that your private key is a cornerstone of your PKI and that its safekeeping is vital. After confirming your pass phrase, you'll be prompted for the attributes of your CA certificate as shown in Listing 1.

OpenSSL generates base64 encoded certificates between "-----BEGIN-----" and "-----END-----" lines by default. This format is commonly but mistakenly referred to as PEM format. True Privacy Enhanced Mail format is actually used by some SSL servers like old Lotus Domino and 4D WebSTAR Server, so I'll refer to the OpenSSL format as OSSL. The JSSE integration tools expect OSSL-formatted certificates. Regardless of which SSL toolkit you used to complete this step, your final product should be an OSSL certificate and a private key. These two components, along with your SSL toolkit, provide all you need. Congratulations, you've successfully become a certificate authority.

JVM Integration

Your next step is to tell your JVM that you're a trusted CA so it will inherently trust any certificates you issue. This is accomplished by adding your CA certificate to your JVM's CA certificates keystore. The default CA certificates keystore is found in your runtime environment's security directory, found underneath the lib directory. It's intuitively named "cacerts". We'll be modifying the cacerts file, so make a backup copy before continuing.

The keytool utility distributed with J2SDK allows us to perform operations on any JVM keystores, so let's use it to examine the default cacerts.

```
keytool -list -keystore cacerts
```

When you execute this command, you'll be prompted for the keystore password. The default cacerts password is "changeit".

As you can see in Listing 2, the default cacerts contains certificates from VeriSign and Thawte, which together issue the vast majority of Web site certificates. Our JVM recognizes most Web-based e-commerce certificates already. Now let's add our CA certificate to the keystore.

```
keytool -import  -alias newCA -file
 rootCACert -keystore cacerts
```

When asked if you wish to trust this certificate, answer "yes". If you examine the cacerts keystore again, you'll find that your CA certificate is listed underneath the alias you provided. You're now registered as a CA in your JVM. Let's use our new power as a CA to facilitate an SSL conversation between two parties.

SSL Demonstration

To demonstrate that your JVM will trust certificates that you issue, we'll set up an SSL-enabled conversation between Alice and Bob. Alice and Bob represent two servers in an enterprise system that need to communicate securely. For this demonstration you can use, but do not need, two separate boxes. To keep things simple, we can just launch two JVMs on the same box and share the same local cacerts file (see Figure 1). In this figure, configuration A shows the setup for the demo running on separate boxes. Configuration B shows the setup for the demo running on one machine.

Regardless of your hardware setup, both Alice and Bob will need access to a cacerts that we inserted our CA certificate in. Each communicating entity also needs a personal keystore. Let's make two copies of our cacerts file and place them in a working directory. Name the first copy aliceStore and the second bobStore. Next, we'll generate a public and private key pair for Alice and store the pair in her personal keystore. A screenshot of this process is shown in Listing 3.

Alice now has a 1024-bit RSA key, which will be valid for 365 days. This key pair allows Alice to participate as an unauthenticated client in an SSL conversation. However, in our demonstration system, we want to guarantee the identity of all conversation participants. Since Alice needs a certificate to participate as an authenticated client, she'll need to generate a certificate request to send to her certificate authority.

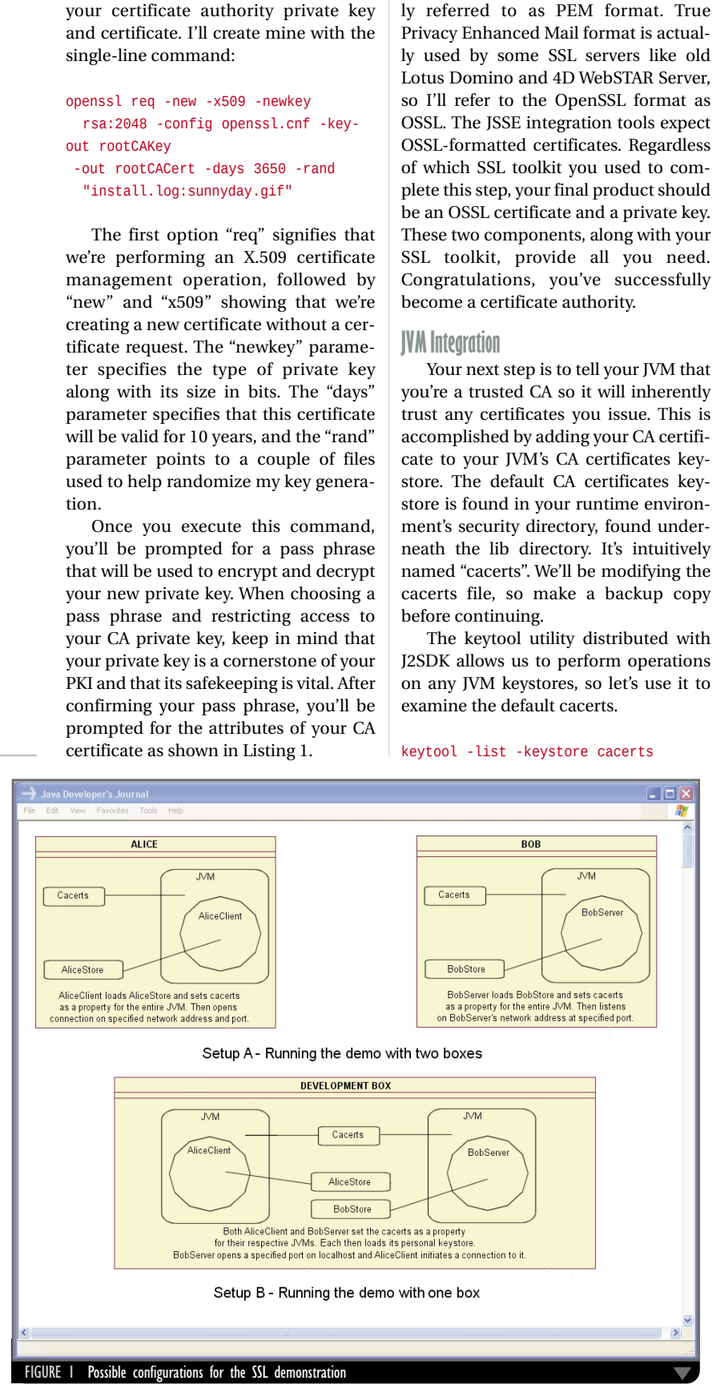


FIGURE 1 Possible configurations for the SSL demonstration

```
keytool -certreq -alias alice -sigalg
  MD5withRSA -file aliceCSR -keystore
aliceStore
Enter keystore password: changeit
```

Note that my keytool utility balks on this step if the key generated in the last step has spaces in any Distinguished Name attributes. So, I used "Tallán" for my organization name since "Tallán, Inc." threw an exception. If you complete this step successfully, you'll have a certificate request for Alice. Now we can put our CA hat back on and generate a certificate according to Alice's request.

```
openssl x509 -req -CAcreateserial -
  CAkey rootCAKey -days 365 -CA
  rootCACert -in
 aliceCSR -out aliceReply
Loading 'screen' into random state -
  done
Signature ok
subject=/C=US/ST=Connecticut/L=Glaston
  bury/O=Tallan/OU=Server/CN=Alice
Getting CA Private Key
Enter PEM pass phrase: [passphrase]
```

The resulting aliceReply is a certificate that's valid for 365 days. We return this certificate to Alice, which needs to store this certificate alongside her key pair in her personal keystore.

```
keytool -import -alias Alice -trust
  cacerts -file aliceReply -keystore
 aliceStore
Enter keystore password:  changeit
Certificate reply was installed in
  keystore
```

Great, Alice is all set. Now just repeat the same procedure for Bob. Once you have Bob's certificate stored in bobStore, it's time to work with some code.

There are four classes that we'll use in this demonstration. The first, SSLDemoException, is a simple wrapper around the Exception class. There's nothing to it. The other common class is the abstract base class SSLDemo, which contains some common features of the client and the server. The first item of interest in SSLDemo is the static initializer, where I dynamically add the Cryptographic Service Provider "SunJSSE" to my list of security providers, as described in the JSSE installation instructions. I also override the default keystore used by the TrustManager. This step designates the keystore that holds all my trusted CA certificates.

To improve performance for multiple connection conversations, my constructor preseeds a pseudorandom number generator that will be used to generate each SSLContext. I also load my personal keystore upon construction. My personal keystore holds the certificate chain needed to validate my own certificate. As an example, Alice's personal keystore must hold Alice's certificate and our CA certificate.

Since a good portion of the JSSE-specific logic is encapsulated in my base class, my client and server classes are kept relatively clean of JSSE-specific code. Each must define its personal keystore and use SSL-specific socket factories. In addition, the BobServer conditionally sets client authentication on its SSL-enabled server socket.

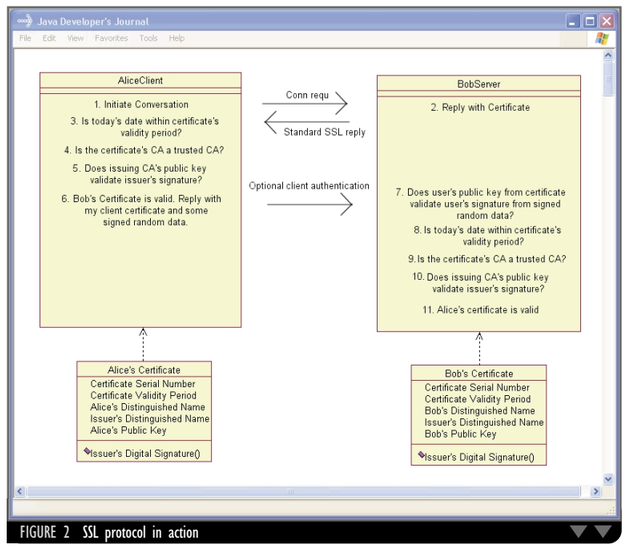


FIGURE 2 SSL protocol in action

Let's put the code into action and step through an SSL conversation. First start the BobServer with client authentication enabled.

```
java BobServer true
Bob is listening on port: 4242

Then invoke AliceClient
Java AliceClient "Hello Bob, this is
 Alice"
Sending message to Bob: Hello Bob
Bob's Reply: RE: Hello Bob - Hello
 Alice, this is Bob.
Done...
```

The result seems a bit anticlimactic from the outside, but Figure 2 details the more impressive internal workings.

As you can see, client authentication simply requires that the client in the client/server conversation also presents a valid certificate. Since Alice has a valid certificate, AliceClient can talk to BobServer with client authentication enabled or disabled. However, if Alice only had a public/private key pair in aliceStore with no certificate, then Alice would only be able to converse with Bob with client authentication disabled. That demonstration is left as an exercise for the reader. So Alice accepted Bob's certificate and Bob accepted Alice's. They both trusted certificates that we issued as the CA in our PKI.

Conclusion

We've provided you with the ability to act as a CA in a minimalist PKI implementation, and demonstrated how you can integrate your self-certified certificates with J2EE systems via JSSE. Before you rush off to develop SSL-enabled enterprise systems, be aware that I've only exposed you to some fairly complex material. There's a lot more to developing enterprise PKI than simple certificate generation and distribution. I've detailed enough to make you dangerous, but security is only as good as its weakest link. Careful consideration must be given to all areas of a system's security to avoid negating any that SSL might provide.

Resources

- *Java Secure Socket Extension:* http://java.sun.com/products/jsse/
- *The OpenSSL Project:* www.openssl.org/
- *Netscape's Secure Sockets Layer:* www.netscape.com/security/techbriefs/ssl.html

eric.simmerman@tallan.com

AUTHOR BIO

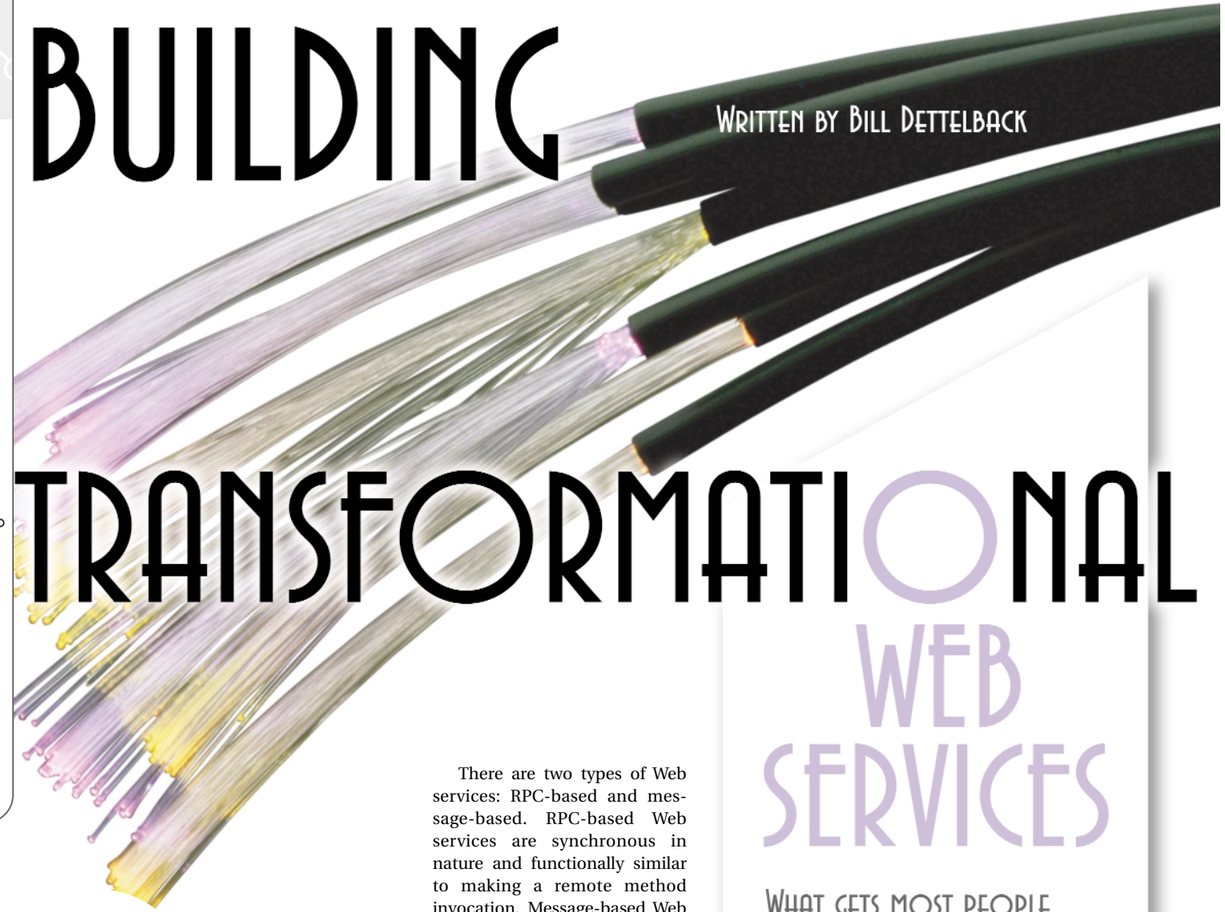
Eric Simmerman is a senior consultant in the development division of Tallán, Inc., where he designs and implements numerous enterprise systems in the U.S. and abroad. He's a Java 2 Sun Certified programmer and a Cisco Certified Network Associate and has been developing professionally for seven years.. He holds a BS in computer engineering from Virginia Tech.

Infragistics, Inc.

www.infragistics.com

BUILDING TRANSFORMATIONAL WEB SERVICES

Written by Bill Dettelback



Web services is intended to create the synergy of many applications working together. The whole is greater than the sum of its parts. Web services must be resistant to change and quickly adaptable to new types of users who want to use them on their own terms and with their own data formats.

These transformational Web services, as I'll call them, will rely heavily on XML to meet this demand. There's a good reason why current Web services protocols such as SOAP and WSDL are defined using XML. XML is an excellent format for storing and managing information in a standard, extensible manner that is also flexible. In this article, I'll look at why a native XML database can be a powerful complement to certain types of Web services.

There are two types of Web services: RPC-based and message-based. RPC-based Web services are synchronous in nature and functionally similar to making a remote method invocation. Message-based Web services are asynchronous and used primarily to pass business messages (or documents) between services. To make a J2EE analogy, an RPC-based Web service is very similar to invoking a method on a session bean, and a message-based Web service is very similar to placing a message on a JMS queue.

Web services are described using an XML dialect called Web Services Description Language (WSDL). WSDL provides a vocabulary for defining exactly how to access a Web service, whether it's RPC or message-based, what parameters to provide, the URL to use, and so forth.

Middleware providers have quickly adopted Web services and included tools with their products for managing SOAP messages with EJBs and servlets. Many of these tools focus on shielding the J2EE application from the XML as much as possible, minimizing the amount of

> What gets most people excited about Web services is that it provides a vision of a future where disparate applications are hooked together in innovative yet undiscovered ways to solve the next generation of IT problems.

the service in detail, message-style Web services have the flexibility to accept any structured information in the SOAP message body. This makes them very attractive for implementing Web services that deal with more complex data whose structure might change over time or depending upon the caller.

Unlike RPC-based, message-based Web services have direct access to the different parts of the SOAP message that invoked them. This provides a great deal of flexibility in how the Web service chooses to process the message. It also means that the service might be doing a lot of work with raw XML (e.g., DOM programming) if the document is complex or large.

> "XML IS AN EXCELLENT FORMAT FOR STORING AND MANAGING INFORMATION IN A STANDARD, EXTENSIBLE MANNER THAT IS ALSO FLEXIBLE"

SOAP-specific information that the developer must deal with. In particular, RPC-based Web services can be implemented with EJBs and servlets that have no idea they're being invoked by SOAP. The tools provide facilities where an XML-to-Java mapping can take place and the Web services implementation deals exclusively with Java data types.

This is a worthy goal, as a lot of SOAP programming is low level and tedious. However, it's important to remember that in some cases there's real value to preserving the XML associated with a SOAP message. By discarding the XML that composes the SOAP body, a Web service is throwing away the ability to manage extensible information. This extensibility (an inherent trait of XML) is key to building a Web service whose interface is not tightly bound to a particular document structure.

Message-based Web services provide convenient gateways for applications that need to submit requests and then get notified of the response at some point. There's loose coupling between these Web services and the caller. Unlike RPC-based Web services, where the caller must know each parameter of

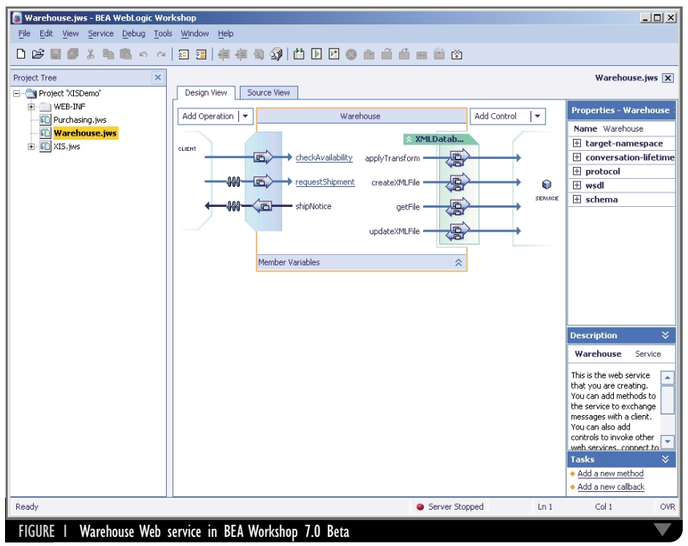


FIGURE 1 Warehouse Web service in BEA Workshop 7.0 Beta

If many different partners use the Web service, it's cumbersome for the developer to change the business logic of the Web service every time a new partner is added. In addition, it's a challenge to store the data found inside the XML document if every caller is going to provide a document adhering to a different XML Schema. To make life simpler, many applications simply discard the XML after parsing out the relevant bits of information and storing them in a relational database. But sometimes it makes sense to keep not only the relevant data, but the XML document itself, as parsing and storing destroy the structure of the document (and add considerable overhead to the process).

This is where native XML databases fit in. A native XML database operates very much like a familiar relational database. However, instead of managing data as rows and columns, it manages data as XML. A native XML database is built from the ground up to store and manage XML in a parsed format. This means there's no conversion taking place from the XML to some external storage format (either rows and columns or "BLOBS" of text). Native XML databases typically outperform relational databases when it comes to storing and retrieving XML because they store the elements of a document in a preparsed format.

An XML database is a perfect fit for the needs of the message-based Web service outlined earlier. It provides a transactional, secure place to hold incoming messages that won't bog down the application's database of record. But there's another key reason for using an XML database – native XSLT support.

XSLT is a language that describes stylesheets for transforming XML into other formats. Most applications that process XML in some fashion use XSLT primarily to format the XML before presenting it to a user. XSLT can do far more than simply convert XML into HTML or the like. XSLT is ideally suited to transform XML from one dialect to another. When we consider the needs of a Web service that must handle multiple incoming messages that may be in different formats, XSLT is the natural choice for solving this problem. A native XML database that has a built-in XSLT processor has a major advantage over in-memory transformations, namely the ability to handle large document sizes without excessive memory consumption.

Let's focus on a concrete example. Imagine a warehouse that serves a variety of retail suppliers. It contains numerous inventory types and is constantly expanding the types of inventory it contains. At different times during the year the

warehouse will focus on certain types of inventory more regularly than others, which means the population of users is constantly shifting. The warehouse exposes the ability to request an order via a message-based Web service. I've chosen to implement this example using the beta version of BEA WebLogic Workshop.

As you can see in Figure 1, the Web service has a few methods: checkAvailability(), which finds out whether an order can be filled with current inventory, and requestShipment(), which starts an order process. For now we can ignore shipNotice(), which is a callback when the order is ready to be shipped.

Suppliers will invoke the requestShipment() method via SOAP invocations. The signature of the requestShipment() as seen in Workshop is shown in Listing 1. (Listings 1–5 can be downloaded from www.sys-con.com/java/sourcec.cfm.)

This service lets suppliers provide their own identifier (<supplier>), an identifier for their purchase order (<poNumber>), and, finally, the purchase order itself (<doc>). The service is intentionally vague about what the purchase order should look like so it can handle different kinds.

Currently, the warehouse provides inventory for two suppliers, Acme and BigRetail. The Acme purchase order in its entirety can be found in Listing 2, the BigRetail purchase order in Listing 3. Notice that while they're structurally different, both documents express basically the same information. What is important is that both suppliers expect the Web service to behave the same way for their purchase order.

I implement the method in Workshop by accepting a DOM node for the document, a string for the poNumber, and an integer for the supplier code:

> "NATIVE XML DATABASES TYPICALLY OUTPERFORM RELATIONAL DATABASES WHEN IT COMES TO STORING AND RETRIEVING XML"

```
public void requestShipment( Node doc, String poNumber,
int supplier ) throws Exception
```

This method needs to store the purchase order somewhere and then begin the process of filling the order (checking inventory levels, etc…). Since these processes will need to refer to the particulars of the purchase order, it would be unwieldy to write parsing code to handle so many different types of documents. A cleaner approach is to convert each incoming purchase order into a "canonical" format and operate only on those documents (see Figure 2).

Let's examine the requestShipment method in detail (see Listing 4). First the code takes the incoming purchase order and serializes it into an XML string using Apache's org.apache.xml.serialize.XMLSerializer class. This string is then passed to the XML database's createXMLFile method. I've chosen to use eXcelon's eXtensible Information Server (XIS) as the XML database. I've "wrapped" a subset of the XIS's API using a Web service control named XMLDatabase. This is a technique specific to Workshop and lets us keep the database-specific API out of our Web service code. The XIS presents a filesystem metaphor for storing XML in folders and documents. The createXMLFile() method takes an XML database name as the first parameter ("Warehouse"), a path to the filename indicating where to store the XML (using the PO number as the filename in the "incoming" directory), the XML string itself, and finally an argument indicating whether or not to trim white space.

Once the original purchase order is stored in the database, the method can perform a transformation to a canonical format. The XSLT stylesheet to use for this transformation is chosen based upon which supplier number was received. The transformed purchase order document is returned by the applyTransform() method as a string that's also stored in the database, this time in the "toBeShipped" directory.

The code then invokes the processOrder() method (the details of which I have omitted for brevity) that starts the business process of filling this order. Once the system has started processing the order, it can make the callback to issue the ship notice for the client.

One detail I have not discussed is how to write the XSLT that transforms the Acme and BigRetail purchase orders into our canonical format. These stylesheets are not terribly complicated to write, however, they can be rather tedious to construct.

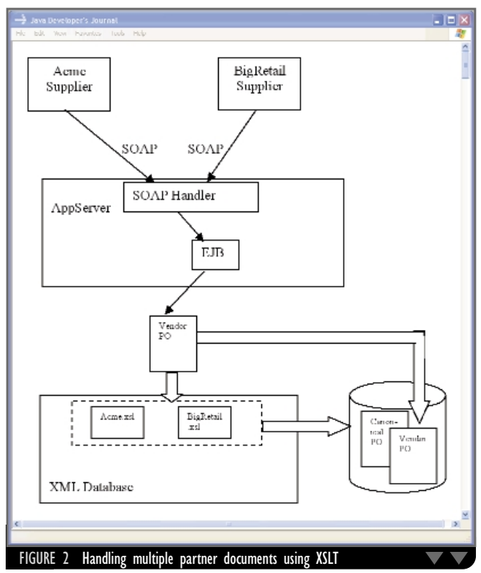


FIGURE 2 Handling multiple partner documents using XSLT

cessing to an external entity, this Web service can be used by many suppliers simultaneously without concern for memory consumption issues. Unlike an in-memory XSLT processor, the database server will process large documents, not the application server hosting the Web service. For extremely large (multimegabyte) documents, this can have a profound effect on scalability.

4. ***Extensibility is preserved:*** For example, if BigRetail decided to change the <IDENTIFIER> element to include children (for instance <SUB_ID>), this could cause major code changes at the warehouse. To make matters worse, if BigRetail expected that the warehouse would respond with an XML invoice using the same element-child structure, the impact to the code could be widespread. By localizing the XML transformations into XSLT, such extensions are quickly handled in a single spot without massive code changes.
5. ***Knowledge can be gleaned from a supplier's behavior:*** Related to the first point on persistence, because everything is retained by the system, reports and analysis can be performed on the purchase orders to gain knowledge about the business. For example, if a particular supplier is repeatedly requesting the same item during the same week each month, you might want to target that supplier with discounts on related products the week prior to help increase sales. Without retaining this information, such opportunities would be lost.

> "Transformational Web services can be easily constructed by leveraging the strengths of a native XML database and XSLT"

The XIS comes with a tool called Stylus Studio that helps build stylesheets visually. Figure 3 shows the XML mapper in Stylus for the Acme purchase order; Listing 5 is the resulting XSL stylesheet code. These tools can be helpful for large documents with many processing rules; however, for simple documents, hand-coded XSLT can be more than adequate.

There are several benefits to using an XML database as the intermediary for these types of Web services:

1. ***Everything is persistent:*** Every business document is preserved in the database in the event of a system failure. This includes the supplier's original purchase order that can be referred to as needed if there is a dispute or question after an order has been shipped.
2. ***The Web service does not get involved in transformations:*** The role of the Web service is straightforward and does not change much when a new supplier is brought on board. The way I have coded, the requestShipment() method would require a new case statement to be added for a new supplier. However, even this could be parameterized by storing the stylesheet names in the database keyed by supplier identifier.
3. ***Scalability is greatly enhanced:*** By offloading the XSLT pro-

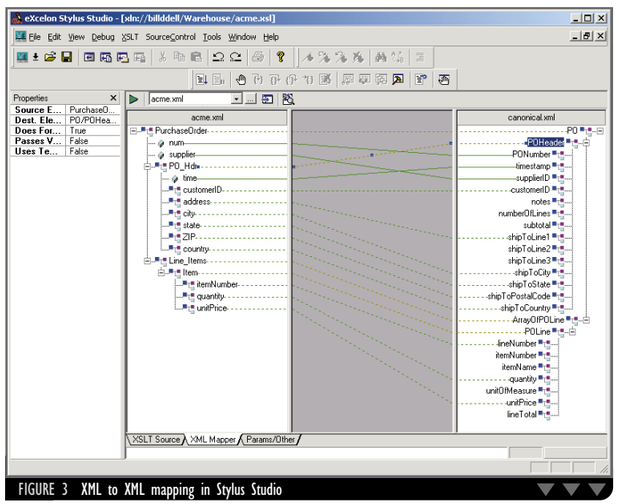


FIGURE 3 XML to XML mapping in Stylus Studio

Summary

Transformational Web services can be easily constructed by leveraging the strengths of a native XML database and XSLT. This architecture lends itself to building Web services that adapt to new types of incoming messages and are tolerant of change.

While the previously discussed example involving processing purchase orders is relatively simple, you can imagine more complex Web services that rely on the transformational power of XSLT and the persistence of an XML database. For example, imagine a Web service that provides a unified view of a loan application, taking financial information from a variety of banks and institutions for an individual and providing most of the application prepopulated.

Another example might be a Web service that acts as a middleman and clearinghouse for exchanging documents pertaining to complex legal transactions. It would be nearly impossible to provide an infrastructure for such services without the flexible storage and transformation facilities provided by XSLT and a native XML database. ●

Author Bio

Bill Dettelback is a systems architect with eXcelon Corporation. Prior to joining eXcelon, he worked as a senior member of the technical staff at AT&T building AI-based customer care applications. Bill holds a BS and an MS in computer science from the New Jersey Institute of Technology.

billd@exln.com

Programming Restrictions in EJB Development

Building scalable and robust enterprise applications

WRITTEN BY
LEANDER VAN ROOIJEN

In 1998 Sun introduced their distributed server-side component architecture under the name of Enterprise JavaBeans (EJB). Since then, the EJB technology has seen a widespread acceptance throughout the industry. The "write once, run anywhere" philosophy embraced by the EJB specification is undoubtedly a major factor in its success. An EJB component can be built once and then deployed on different platforms without recompiling or altering the source code.

The concept behind the EJB API is simple: let the application programmer concentrate on writing business logic and shield him or her from complex system-level services. To achieve this the EJB components must operate within a controlled runtime environment called the *container*. This container is responsible for all the system-level services, such as instance pooling, transaction handling, security, exception handling, and data caching. To provide the EJB component with these low-level services, it's mandatory that the EJB interact only with the container and the resources provided by it.

Programming Restrictions

Not having to write system-level services comes at a cost. To ensure portability across containers from multiple vendors, the EJB developer is expected to operate within the framework provided by the EJB specification. This means there are restrictions on certain features that are normally available to the Java developer. The specifications of the runtime environment (EJB 1.1 ch18, EJB 2.0 ch24) list these restrictions.

Some restricted features such as changing the socket factory of ServerSocket are rarely used, and this article won't elaborate on these types of restrictions due to the fact that few developers will be exposed to this level of programming. However, there are restrictions that can have a significant impact on your usual coding techniques.

In this article I'll state some of the programming restrictions as defined in the EJB specifications 2.0 (almost the same as the 1.1 spec) and try to explain the reason for the restriction. Where possible I'll also include an alternative approach to reaching your goal without violating the restriction.

Before diving into the details of the restrictions, it's important to clear up one of the greatest misconceptions in EJB programming. Many believe that these restrictions apply only to the code they write within the EJB's classes and not the helper classes used by the EJB component. This is wrong. The EJB components run within the container, hence all classes that run within the container are subject to the same restrictions.

An enterprise bean must not use read/write static fields. Using read-only static fields is allowed. Therefore, it is recommended that all static fields in the enterprise bean class be declared as final.

This restriction is simple but it has the greatest impact on a programmer's normal coding routines. When using a static field (class variable) you expect all your instances to access the same field. If you're operating within one JVM, this is a correct assumption; however, the EJB server can run multiple JVMs for performance or load-balancing reasons. Now each JVM contains one or more instances of the EJB and these instances no longer reference the same field. In addition, there's the problem of concurrent access to the static field, because in the EJB specification the use of the threading API is also restricted, so no synchronization mechanisms are available (more on this later).

To use a static field you'll have to declare it to be read-only. A final static field cannot be changed after it has been initialized, and instances across all JVMs will now be consistent.

One problem facing many programmers is how to implement the Singleton pattern (one instance of a particular class in the whole application) in an EJB when the use of static fields is restricted. Some argue that using a JNDI to store a reference to your instance can solve this problem, but I haven't seen a foolproof implementation that works. You might be dealing with the same instance but you still have the concurrency issue to contend with. Furthermore, a Singleton EJB would by definition run in a single container, and that means introducing a single point of failure. Since the EJB architecture is all about load balancing and fault tolerance, I believe the Singleton pattern should not to be used in distributed EJB environments.

If you want to use a Singleton for a nonblocking and read-only service and

> According to the specifications the file system is not an appropriate mechanism to access data

it's not a problem having one instance in each JVM or for each deployed ejb-jar's classloader, then the use is warranted. An example of these services is the caching of data retrieved through expensive JNDI calls. You just have to realize that it's not a Singleton in the classic form and use it accordingly.

An enterprise bean must not use thread synchronization primitives to synchronize execution of multiple instances.

As mentioned before, the same EJB can be run in different JVMs. It's up to the container to take care of the object's life cycle and any concurrency issues. Therefore, declaring an EJB component's method to be synchronized is a violation of the specification. The utility classes used within the EJB, such as the Collection classes, are allowed to use the synchronized keyword on its methods or blocks of code because these are all dealing with single instances.

An enterprise bean must not use the AWT functionality to attempt to output information to a display, or to input information from a keyboard.

This one is obvious. The whole concept of separating the GUI layer from the business logic layer is compromised if direct interaction is permitted. Most EJB servers will allow you to print to the screen from within the container, which can be useful at times during the development process (debugging, etc.); however, it should not be required in a production system.

An enterprise bean must not use the java.io package to attempt to access files and directories in the file system.

According to the specifications the file system is not an appropriate mechanism to access data. Since file systems are not transactional by nature, it's a valid argument. Another problem is that there's no resource manager involved in java.io operations so the container has no control. Now if an EJB uses files on a system with heavy client loads, there's the chance of running out of file descriptors and bringing down the system.

Java developers are accustomed to using property files in their applications because they're helpful in specifying configuration settings. A simple mechanism within EJBs called *environment entries* provides these parameters. These environment entries are placed in the deployment descriptor and consist of a name, type, value, and description. The type can be any of the seven primitive wrapper classes (Integer, Boolean, etc.) or a String object. The entries are retrieved through the bean's environment, naming context. The following code provides the deployment descriptor of an EJB:

```
<enterprise-beans>
  <session>
…
    <ejb-name>YourBeansName<ejb-name>
    …
    <env-entry>
     <description>the example environ
      ment entry< description>
     <env-entry-name>entryname</env-
      entry-name>
     <env-entry-
      type>java.lang.String</env-
      entry-type>
     <env-entry-value>myValue</env-
      entry-value>
    </env-entry>
   </session>
 </enterprise-beans>
```

To retrieve the entry in your code use:

```
Context ctx = new InitialContext();
Context env = (Context)ctx.look-
  up("java:comp/env");
String myValueString =
  (String)env.lookup("entryname");
```

A legal method to read information from files within an EJB component is to use the classloader. The container has granted the classloader permission to use the java.io package, therefore it's possible to use the getResource() or getResourceAsStream() method defined in java.lang.Class to load in a file.

When using a file system it's best to access it through a resource manager the same way you use the javax.sql.DataSource connection factory to obtain a connection. To gain access to a resource (any entity that can be accessed through a URL string) use a JNDI lookup to get a handle to the java.net.URL object and use it to obtain a URLConnection. By using this mechanism to access a file, the resource is decoupled from the application code using the resource. For the application that's using the resource, there's no difference between the local file system and one located on the opposite side of the world.

To achieve the flexibility and scalability of using a resource manager for URL entities, a resource manager must be set up in our application server. The method to declare a resource manager is vendor-dependent, so consult the container's documentation. Once this is done, all that remains is adding an entry in the deployment descriptor of the bean and the application can use the resource. The following code provides the deployment descriptor of an EJB using a URL resource:

```
<enterprise-beans>
 <session>
   …
   <ejb-name>YourBeansName<ejb-name>
   …
   <resource-ref>
    <description>the URL of the
     resource< description>
    <res-ref-name> url/TheResourceURL
         </ res-ref-name >
    <res-type> java.net.URL</ res-type >
    <env-auth>Container</ env-auth >
  </ resource-ref >
 </session>
</enterprise-beans>
```

To retrieve the resource in your code use:

```
Context ctx = new InitialContext();
  java.net.URL url = (java.net.
   URL)ctx.lookup
   ("java:comp/env/url/
  TheResourceURL");
java.net.URLConnection conn =
  url.openConnection;
//Now create an input stream for
  reading
java.io.InputStream is =
```

```
  Conn.getInputStream();
// or an output stream for writing to
   the resource
java.io.OutputStream is =
   Conn.getOutputStream();
```

An enterprise bean must not attempt to listen on a socket, accept connections on a socket, or use a socket for multicast.

In other words, an EJB is not allowed to act as a network server. This makes sense since clients are supposed to connect to the EJB using a specified remote protocol that's in-line with the implemented security permissions. When a client uses a socket to connect to the EJB, it's a potential security hole. This does not limit the use of a network socket client from within your EJB. An example of such a socket client is a stateful session bean that queries an existing inventory system that can be accessed only through a TCP connection.

The enterprise bean must not attempt to query a class to obtain information about the declared members that are not otherwise accessible to the enterprise bean because of the security rules of the Java language. The enterprise bean must not attempt to use the Reflection API to access information that the security rules of the Java programming language make unavailable.

One feature provided by the EJB architecture is a security model. It's possible to declare programmatic or declarative security roles on methods that can be called by clients. These clients must have a required role to be granted permission to invoke the method. By using the reflection API, it's possible to bypass these security restrictions and invoke methods that you should not have access to. Any other use of the Reflection API is unrestricted.

The enterprise bean must not attempt to create a classloader; obtain the current classloader; set the context classloader; set security manager; create a new security manager; stop the JVM; or change the input, output, and error streams.

This basically states that you are not allowed to alter the runtime environment of the container. In enterprise applications, security is one of the most important issues, and the security managers and classloaders are at the root of protecting applications from unauthorized access of the runtime entities. Again, the EJB architecture has dealt with security by having the container be responsible for managing all access to the EJBs. By intercepting all method calls on the entities, the container can use the Java 2 platform security policy mechanism to prohibit certain functionality. To be able to perform this the container must have a stable execution environment that it creates by using a set of classloaders and security policies. If we were to make runtime changes to them, the container would lose control over the execution environment and security could not be guaranteed.

The enterprise bean must not attempt to manage threads. The enterprise bean must not attempt to start, stop, suspend, or resume a thread; or to change a thread's priority or name. The enterprise bean must not attempt to manage thread groups.

The reason for not being able to suspend, resume, start, stop, or create new threads again has to do with the container's ability to control the runtime environment. When dealing with a high-transaction environment, performance is an important issue. If there's no strict management of resources, like the use of threads, there's a good chance that the application will use too many resources and slow the system down to unacceptable levels. Or worse, the server is no longer able to serve client requests.

Thread-specific storage is another problem with using the threading API. For an EJB container to do its work properly, it needs to track information for the current request. In a system with thousands of simultaneous requests being processed, it's important to have an efficient and manageable way of getting at the required information.

Since the Java 2 platform, there's a class called java.lang.ThreadLocal for storing information associated with a specific thread. Most container vendors use a single thread dedicated to servicing a single request, and hereby they can use the ThreadLocal class to store the current user, transaction, security context, or any other required parameter from the request. If an EJB were to create new threads, the required information might not be propagated correctly to the new thread, potentially causing serious errors.

Not having to manage threads is a relief for most developers, but not being able to use custom threads has its drawbacks. For example, if you have a bean that needs to query different multiple external systems (ERP, legacy, etc.), it will have to be done in a series. Since these systems will take time to process your request, the time spent waiting can add up.

If you could use threads, you would be able call these requests in parallel, significantly reducing the overall time needed to process your request. There's talk of introducing a limited threading model into the EJB architecture, and it will most likely be in the form of an EJB retrieving a thread from a thread pool managed by the container. No threading mechanism has been mentioned in the EJB 2.0 specifications, so it could still be a long way off.

The enterprise bean must not attempt to pass this as an argument or method result. The enterprise bean must pass the result of SessionContext.getEJBObject(), SessionContext.getEJBLocalObject(), EntityContext.getEJBObject(), or EntityContext.getEJBLocalObject() instead.

The life cycle of all bean instances is managed by the container. That's why we always access the bean through the EJBObject, never directly. The container can activate or passivate a bean at any time, and the client is never aware that this is taking place. While a client has a reference to the EJBObject, the container can passivate the actual bean instance. When the client invokes a method through the EJBObject, the container intercepts the call and performs tasks such as activating the bean and reassociating it with the EJBObject.

This programming restriction is questionable when you're dealing with helper classes. Since the helper classes execute

> Not having to manage threads is a relief for most developers, but not being able to use custom threads has its drawbacks

only while the bean is active, theoretically, the use of the "this" reference should not cause any problems when used in this manner, but Sun Microsystems might want to clarify the issue.

In Practice

Most programming restrictions are clear and concise, but a few restrictions are open to interpretation. This is probably why not all containers enforce these restrictions rigorously. The container from the J2EE reference implementation throws SecurityExceptions when the restrictions are violated. Many commercial EJB servers, such as BEA's WebLogic and IBM's WebSphere, don't seem to enforce a lot of these restrictions. These servers are still valid implementations because according to the EJB specifications, the container may allow more or fewer permissions to the enterprise bean instance, which can prevent portability across different containers.

When developing EJB components it's best to test them against the reference implementation to verify that they conform to the specifications. Unfortunately, most development projects are governed by strict deadlines and portability is not always at the top of the list. However, some developers forget that you should stick to the specification not just because of the portability issue, but because reliability can also be affected. Unfortunately, reliability problems always become apparent near the end of the development phase when there's little time left. If it turns out that some components need to be reengineered because they violate the programming restrictions, it will be a lot more time-consuming to fix them than to verify them against the reference implementation from the start.

One more reason to adhere to the programming restriction of the EJB specification is reuse. When you're in the business of building solutions for your clients, you'll undoubtedly deal with EJB servers from multiple vendors. The ability to reuse generic components that have been tested and proven to work across multiple EJB servers can save you significant amounts of time.

> The EJB technology has put the building of scalable and robust enterprise applications within reach of a great number of Java programmers

The EJB technology has put the building of scalable and robust enterprise applications within reach of a great number of Java programmers. Unfortunately, a number of EJB developers have never read the specifications and are unaware of all the programming restrictions. Hopefully, this article has cleared up some issues that Java developers face during their road to proficient EJB development. Just remember, EJBs are only portable if you write them that way.

lrooijen@cgey.nl

Author Bio

Leander van Rooijen is a senior consultant for Cap Gemini Ernst & Young in the Netherlands. A Sun Certified Java developer, he's a specialist in server-side technology. Leander holds a degree in mechanical engineering.

IBM

www.ibm.com/websphere/winning

IBM

www.ibm.com/db2/rocks

Are You Ready to Rumble?

.NET vs J2EE Smackdown

WRITTEN BY
MICHAEL DEASY

There is a raging debate in the developer community: Microsoft's new platform .NET versus the Sun standard J2EE. J2EE has been around for a few years, while Microsoft's attempt is decidedly the new kid on the block.

While the comparisons may be defined by some as purely technical, some have deemed it a battle of corporate cultures. Sun has certainly chosen to consider that the battle.

During a lunch discussion with my fellow officers in the Tulsa Java User Group, we explored the issues. We also decided to see if we could get some vendors to come in and discuss the issues from both perspectives. This expanded into a formal debate, and thus the .NET vs J2EE Smackdown was born. Our group started to plan this event; we contacted Microsoft and Sun and both sides were more than willing to engage.

On April 25, 2002, in the Williams Theater in downtown Tulsa, Microsoft and Sun came together on one stage (together again, for the first time anywhere) – and the battle was joined. The officers of the JUG decided to consult with the officers of the PBUG, and we held the meeting as a joint venture – sponsored by both user groups as well as Oaktree, Inc., a consulting company that sponsors our groups extensively. We had two podiums, many microphones, LAN connections, and team theme songs. It was exciting to watch as over 200 people filled the theater to see the Tulsa Smackdown, to find out what this debate was really about.

What If They Held a Raging Debate and Nobody Showed Up?

Microsoft and Sun showed up in force. Microsoft sent three of their technical evangelists: Mark "The Doctor" McLoughlin, Dave "Brick Wall" White, and Dino "Hannibal" Ciesa, while Sun sent a team of four of their faithful (and faith spreading): Bill "The Liberator" Day, Max "The Preacher" Goff, Rima "The Instigator" Patel, and Tom "Secret Weapon" Daly. They were all seasoned veterans of the software Ring Wars, and brought with them their colorful styles and insights. Microsoft chose "Eye of the Tiger" (from *Rocky III*) as their theme song, and as it poured out of the speaker systems they came proudly down the aisle of the theater, their Microsoft logo'd button-down finery on display. Sun chose an old Sly and the Family Stone song "Freedom" as their mantra and came boldly to the stage during their song.

The rules for the Smackdown were as follows: eight questions that each team had to answer; they were required to give software examples, state their case clearly, and show their stuff. Both teams had the questions ahead of time and had veto rights over anything they deemed unworthy. Each team had five minutes to present their case, then the other team had a five-minute rebuttal. They each had two yellow cards to raise as penalties on the other side if they deemed a question worthy of an immediate, two-minute rebuttal. It goes without saying that these two teams pulled their cards and used the maximum on each and every question.

The crowd was eager for the words, lines of code, and interfaces to fly, and so we began with introductions, prizes, and a lot of good cheer, then the debate began.

Two Companies Separated by a Common Coast

Sun and Microsoft brought some of their strongest speakers and came with a sense of anticipation that reportedly went far up the chain in their respective corporate worlds. And both came with their own specific agenda. Microsoft came intent on showing a room full of programmers what their product can do and how truly RAD it is. Sun came with a lot of invective and vitriol, and the intention of making this debate about the stark differences in their corporate cultures and goals.

The debate began with five-minute introductions. Microsoft came on first, providing a somewhat detailed and very colorful blanket overview of .NET. They gave a strong summary of what their systems could do, how they connected together, and what we had to gain from their use.

Sun came out with a copy of "the lawsuit" and began talking about the differences between the corporate cultures at Microsoft and Sun. They said, "Point of fact…" and then quoted the lawsuit regarding Microsoft's predisposition for disposing of the competition and using unfair trade practices. There was not much in the way of information about J2EE.

Top Questions from Our Home Office in Enid, Oklahoma . . .

The following are the questions posed to each side and the approximate responses.

1. *What sets your platform apart from that of your competitor?*

Microsoft talked about the speed with which they could deploy their product, the multiple language and platform independence of .NET, and how easily integrated it could be.

Sun talked about customer service, feeling a warm commitment from your vendor, and avoiding the evil demons that live in the Seattle area. They also talked about unfair trading practices and monopolies.

2. *CMP TechWeb defines the term* Web services *as "Web-based applications that dynamically interact with other Web applications using open standards that*

include XML, UDDI, and SOAP. Microsoft's .NET and Sun's J2EE are the major development platforms that natively support these standards." What is your platform's level of support for Web services? Can you show us using your platform and tools?

Microsoft created a Web service, deployed it, and brought data in from a database with a few simple keystrokes. They had to uncomment some code in predeveloped areas, and it took a few seconds for the data to return their request across the Web, but it was an extremely impressive demonstration.

Sun did show some slides on this one, and brought up lists of powerful clients they had serviced or who had used Sun's products.

3. *Designing and constructing a multi-tier transactional Web-based application to support hundreds (if not thousands) of simultaneous users is a very daunting task. How does your platform address these complexities and empower organizations to build such applications quickly and with minimal expense? Can you show us using your platform and tools?*

Sun approached this with some very strong examples of Web sites they built or support involving hundreds and/or thousands of users. The most impressive was probably the ESPN Web site, a site that definitely has a huge fan base. They did give an example of how to throw connectivity together very quickly; unfortunately, it never did load.

Microsoft again showed some PowerPoint and explained how .NET can be made more flexible. There was no time for either contender to get into a whole lot of detail but they both showed good examples.

4. *Many of our audience members are software developers who are constantly looking to hone their skills and learn the latest technologies. Ultimately, each one of us will try and decide which platform will produce the most marketable skill set. Do you think your platform offers the most marketable skills and if so, why?*

This question brought about surprising and unexpected responses. Microsoft gave a two-minute talk on how to leverage your current skill set and make use of your C++ and Visual Basic knowledge. They addressed their training program and the wide availability of classes and skill-building tools, from online to college classes.

Sun took the stance that if you decided to become a .NET developer, you were just in it for the quick buck, the easy money. The speaker was critical of anyone who didn't choose what he considered the slower but stronger path to growth and money by learning Java. Addressing a room full of analysts and developers with regard to learning skills quickly and becoming more profitable, he could have chosen his words differently. Also, the path to quick money does not typically lie in the use of tools.

Java does have a huge market share advantage; perhaps they don't have a ready-made platform for their tools, but no one in the room doubted that they had the upperhand in this phase of the discussion. They simply chose, again, not to present the strong things that they could do. They continued railing about too much filthy lucre in the Microsoft camp.

5. *Security is becoming increasingly more important to organizations. Explain your platform's security infrastructure and how it differs from your competitor's platform.*

This question was really Sun's chance to shine. Sun began with a lot of facts and figures about Microsoft's security issues. This has been pretty widely disseminated in the press and Microsoft quickly confessed that they had indeed been working very hard at bringing themselves up to speed. This also led to an argument over who had raped the Pet Shop application most thoroughly. Both sides did a lot of finger pointing, and they both pointed to a number of bench tests.

Microsoft and their failing Passport system did have a decided disadvantage on this question. While it may be true that you don't have to use Passport, it's very difficult to get around in a lot of Web areas without it. It does seem like another area where Microsoft was attempting a coup. Big points to Sun on this question.

6. *Gartner estimates that there will be 901 million mobile phone users worldwide by 2003, while Palm, Inc., continues to report record sales of their Palm Pilot handheld. In short, handheld devices are here to stay and the market is expected to experience continued growth. Does your platform support the development of mobile clients? If so, provide the details.*

This question saw great presentations from both sides. Sun had programs they had written, then downloaded into their cell phone during the presentation. The ease with which the tech displayed movie sites for Tulsa, letting us all know what was showing at the AMC 20, was very cool. It was not really clear how this was directly J2EE-related. But he did show a lot of slides regarding their flexibility with many phones.

Microsoft did a similar slide presentation and was frustrated by the lack of a mini camera to display their phone screen. They did a decent presentation, but again, because they chose to demonstrate their technological capabilities, point and set to Sun on this question.

7. *The cost of development, deployment, and maintenance significantly impacts software development projects. How does your platform stack up in terms of these costs when compared to your competitor's platform?*

The question of cost caused a rousing debate, and as time was running out we made question seven the last of the Smackdown.

Microsoft had some very strong points about the cost of their server software and .NET platform in general – bringing their totals in at under $10,000 for a well-equipped enterprise tool. They were very critical of Sun and BEA for the costs of servers and O/S software, quoting numbers in the hundreds of thousands and tearing into their opponent a bit.

Sun retorted with a strong presentation touting both Linux and Apache as free and J2EE-compliant. They attempted to demonstrate this with slides, which was interesting. However, the Sun sales reps are not pushing Apache or Linux – they're driving the market toward their O/S, their flavor of UNIX. Microsoft held the advantage in this piece of the debate and so our competition came to a close.

In the final analysis, both teams made a strong showing. The general consensus was that Microsoft did a better job of showing off their technological capabilities. They used more samples, and their actual program examples were breathtaking. There was a bit of disappointment in Sun's approach – attacking Microsoft's character – and any time they used their technology to advantage they looked very strong, probably stronger and older in their product lifetime.

The various camps gathered around the myriad reps out front and collected trinkets into their bags: pens, books, and T-shirts. The Microsoft crowd was very pleased with their presentations, and the Sun crowd felt very vindicated and righteous. It was a strong event on the whole, educating at times, frustrating at others. No matter whose side you might fall on though, you got to see some sparks and hear a couple of major players tell their tale.

Author Bio

Michael Deasy works in the Seattle area as a project manager, Web designer, and freelance writer. He has been working with PowerBuilder since version 3. Mike holds an MBA from Southern Nazarene University.

deaser26@hotmail.com

JASON BELL J2SE EDITOR

Learning from History

Evolution: *A gradual process in which something changes into a different and usually more complex or better form.*

There's no escaping that the evolution of programming languages has its advantages and disadvantages. The addition of the java.util.regex package to the JDK1.4 API is a perfect example of Java's development since 1995. However, there's a group of programmers who know only Java and no other language, so it's difficult for them to see why things like regular expressions are included. It all boils down to how your own career evolved.

My own personal evolution is a strangled route through a number of languages: Perl, Unix shell scripting, C, and PHP. I added Java to my skillset halfway through, and I was constantly trying to adapt my thought patterns from Perl to Java code. After many late nights trying to get things to work in Java, it all paid off in the end. However, I always wondered why the Java API didn't have any regular expressions. Until I found third-party packages, such as gnu.regexp and OROMatcher classes, I always went back to Perl and completed the job that way.

I believe that it's time for us to ask ourselves honestly how our own evolution is progressing. Are we constantly learning or are we stuck in a loop?

Musicians spend hours perfecting their skills by practicing scales and arpeggios and familiarizing themselves with their instruments. Before any musician joins an orchestra (or a garage band for that matter), he or she needs to achieve a certain skill level.

Musicians also have access to history, and as seasons come and go you'll notice that certain artists are being influenced by other artists. You'll always have pioneers, however. The Beatles were musical pioneers and a multitude of bands have since been influenced by them.

There are parallels we can draw on as Java programmers. We study the APIs and try numerous examples and routines to satisfy ourselves that we have grasped the concept so when the time arises, we can transfer our knowledge to the situation at hand. The success of any musical performance is based on the ability of the performers to interpret a piece of music; the success of any Java project is based the ability of the programmers to interpret the requirements of a project plan.

Where does our programming history come from? Well, there's plenty of information available on the Internet, mailing lists, books, CDs, as well as from your colleagues.

Have you ever searched Google, for example, in an attempt to solve that illusive problem, and then been presented with seven different ways of dealing with it? We need to easily pick out the diamonds from the dust, the documents that will encourage and educate people who need it the most. What are we doing to lay this foundation down for future programmers?

Site References

- *Dictionary.com:* www.dictionary.com
- *JDK1.4:* http://java.sun.com/j2se/1.4/
- *OROMatcher classes:* http://jakarta.apache.org/oro/index.html
- *gnu.regexp package:* www.cacas.org/java/gnu/regexp/

jasonbell@sys-con.com

AUTHOR BIO

Jason Bell is a programmer based in York, England. He has been involved in numerous Web projects over the past five years, the last two of which have been servlet-based.

J2SE INDEX

New Atlanta Communications

www.newatlanta.com

Hello World! in 70 Bytes

Take the challenge

WRITTEN BY
NORMAN RICHARDS

The Austin Java User Group recently sponsored a contest to create the smallest Java Hello World! program. The rules were simple: create the smallest Java class that when executed will display the text "Hello World!" (and only that text) to the console.

The restrictions were that the class must execute under Sun's 1.3 JRE. It may make use of any class or file distributed with the JRE, but any additional files (excluding arguments on the command line) count against the byte total of the Java class file.

In this article, I explain how I arrived at my 70-byte solution. I hope that in the process you learn a bit about Java class files and the Java Virtual Machine. I also urge you to take the challenge before viewing my solution.

Getting Started

Let's first look at the canonical Java Hello World! program.

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

Compiling this class with javac produces a 416-byte Java class file. That's quite a few bytes just to print "Hello World". Javac generates debugging information by default. Debugging can be disabled with the "-g:none" option to reduce the byte count to 336 bytes, but to go much further we have to look at exactly where those bytes are going.

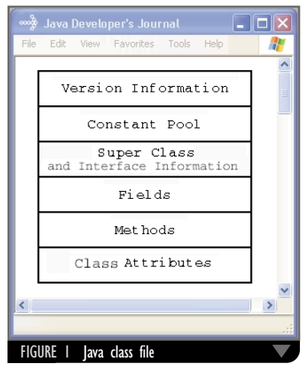


FIGURE 1 Java class file

Figure 1 shows the basic components of a Java class file. In the initial Hello World program, as with most Java classes, the bulk of the bytes come from the constant pool. The method declarations are the next largest chunk, but most of the information used in a method declaration is stored in the constant pool. Table 1 shows the constant pool from the first class.

Note that the constant pool contains most of the details of our code. The class name and method names are all there as are the names and types of all the external classes, methods, and variables we touch. Constant values (such as our "Hello World!" text) are included too. Also, note that constant pool entries link to other constant pool entries. For example, a method reference entry links to a class reference entry (which in turn references a UTF8 text entry holding the name) and a name and type entry (which links to the UTF8 text entries holding the name and the method signature).

To realize how this is used, consider the Hello class expressed as bytecode with constant pool references indicated by the number sign ("#") and the constant pool index number. Don't be scared off by the bytecode. It's not as complicated as it looks. If you've ever programmed an assembler of any sort, this should look quite natural. If not, don't worry. The important thing to note is that in terms of size, the actual bytecode is very small (the constructor is 5 bytes and the main method is 9) because almost everything we do references a field or class or constant defined in the constant pool, which is quite large (see Listing 1).

First Steps

To have maximum control over what goes in the class file I decided to generate the class file directly. Normally a tool such as Jakarta Bytecode Engineering Library (BCEL) would be the best choice to generate the class, but in this case I wanted maximum control (and understanding) of each byte that goes into the class file.

My first attempt was to simply generate a basic Hello World program, minimizing constant pool references and removing any unnecessary portions of the class. There are three main steps.

First, I wanted to make sure I generated only the main method. When compiling a class, the Java compiler will insert a default constructor if you don't specify one. However, this is only necessary if you need to create an instance of the class. If you just need to use the main method, you don't need to be able to create an instance and can remove the constructor.

Next, I wanted to inherit from an already referenced class. Every class referenced in the class file requires entries in the constant pool. I can remove the java.lang.Object constant pool reference I would normally get (remember, even if you don't specify it in your Java source code, your class extends java.lang.Object) by specifying some other class already referenced in the class file. The choices were java.io.PrintStream and java.lang.System. (java.lang.String is used as a parameter only so we don't have class information for it in the constant pool.) Since java.lang.System is final and cannot be extended, the choice was java.io.PrintStream.

Finally, since the name of the class is also stored inside the class file, instead of naming the class "ReallySmallHelloWorldClass", I wanted to choose a name whose text is already in use in the constant pool. Some choices were

#	TYPE	VALUE
1	METHOD REFERENCE	class=#6 signature=#12
2	FIELD REFERENCE	class=#13 signature=#14
3	STRING CONSTANT	value=#15
4	METHOD REFERENCE	class=#16 signature=#17
5	CLASS REFERENCE	name=#18
6	CLASS REFERENCE	name=#19
7	UTF8 TEXT	"<init>"
8	UTF8 TEXT	"()V"
9	UTF8 TEXT	"Code"
10	UTF8 TEXT	"main"
11	UTF8 TEXT	"([Ljava/lang/string;)V"
12	NAME/TYPE	name=#7 type=#8
13	CLASS REFERENCE	name=#20
14	NAME/TYPE	name=#21 type=#22
15	UTF8 TEXT	"Hello, World!"
16	CLASS REFERENCE	name=#23
17	NAME/TYPE	name=#24 type=#25
18	UTF8 TEXT	"Hello"
19	UTF8 TEXT	"java/lang/Object"
20	UTF8 TEXT	"java/lang/System"
21	UTF8 TEXT	"out"
22	UTF8 TEXT	"Ljava/io/PrintStream;"
23	UTF8 TEXT	"java/io/PrintStream"
24	UTF8 TEXT	"println"
25	UTF8 TEXT	"(Ljava/lang/String;)V"

TABLE 1: Constant pool for Hello World

#	TYPE	VALUE
1	CLASS REFERENCE	name=#3
2	CLASS REFERENCE	name=#10
3	UTF8 TEXT	"Code"
4	UTF8 TEXT	"main"
5	UTF8 TEXT	"([Ljava/lang/String;)V"
6	UTF8 TEXT	"Hello World!"
7	UTF8 TEXT	"java/lang/System"
8	UTF8 TEXT	"out"
9	UTF8 TEXT	"Ljava/io/PrintStream;"
10	UTF8 TEXT	"java/io/PrintStream"
11	UTF8 TEXT	"print"
12	UTF8 TEXT	"(Ljava/lang/String;)V"
13	FIELD REFERENCE	class=#14 signature=#15
14	CLASS REFERENCE	name=#7
15	NAME/TYPE	name=#8 signature=#9
16	METHOD REFERENCE	class=#2 signature=#17
17	NAME/TYPE	name=#11 type=#12
18	STRING CONSTANT	value=#6

TABLE 2: Constant pool GenClass 1

"print", "out", and "main", the names of methods and fields referenced. I chose "Code", which is the constant pool tag associated with the Code attribute in the method. The Code attribute is where the bytecode that's associated with the method is stored.

Code to generate this first class is in GenClass1.java. (The source code and Listing 1 can be downloaded from the *JDJ* Web site, www.sys-con.com/java/sourcec.cfm.) The resulting class file is 248 bytes. The following code snippet shows the bytecode, and Table 2 shows the constant pool.

```
public class Code
   extends java.io.PrintStream
{
   public static void
     main(String[] args)
   {
     ;; System.out.print
     ;; ("Hello World")

     ;; get System.out
     0: getstatic #12

     ;; args variable
     3: aload_0

     ;; constant int value 0
     4: iconst_0

     ;; get args[0]
     5: aaload

     ;; invoke print
     6: invokevirtual #15

     9: return
   }
}
```

Hello Command Line

Next, remove the "Hello World!" from the constant pool and pass it in as an argument on the command line. It takes 3 bytes (aload_0, iconst_0, aaload) to reference args[0] as opposed to 2 bytes to load a constant (ldc) string from the constant pool; however, not needing to store the text in the constant pool frees up two constant pool slots and brings the total size down to 231 bytes. Code to generate the class is in GenClass2.java.

The constant pool is similar enough to the first example that we can skip it, but keep in mind that some of the positions in the constant pool have changed. The following is the new bytecode.

```
public class Code
   extends java.io.PrintStream
{
   public static void
     main(String[] args)
   {
     ;; System.out.print("Hello World")

     ;; get System.out
     0: getstatic #13

     ;; get the String
     ;; "Hello World!"
     3: ldc #18

     ;; invoke print method
     5: invokevirtual #16

     8: return
   }
}
```

sun.misc.MessageUtils

Even at 231 bytes the class file is still quite large. Most of the bloat is associated with retrieving the static field out on java.lang.System and invoking the print method on java.io.PrintStream. With that in mind, I scoured the JRE-provided classes for code that would either get System.out for me or print some given text to stdout. Fortunately, there is such a class, sun.misc.MessageUtils, that provides a static method "toStdout" that will print a string to System.out. Using this, I can replace the static field reference (System.out) and the method invocation (java.io.PrintStream.print) with one single static method invocation (sun.misc.MessageUtils.toStdout). Of course, since java.io.PrintStream is no longer in the constant pool, a new superclass is needed. Fortunately, the MessageUtils class is now available to take on this job. Code to generate this class is in GenClass3.java. The resulting class file is 171 bytes. The following code snippet shows the bytecode, and Table 3 shows the constant pool.

```
public class Code
   extends sun.misc.MessageUtils
{
   public static void
     main(String[] args)
   {
     ;; toStdout(args[0])

     ;; args
     0: aload_0

     ;; constant 0
     1: iconst_0

     ;; get args[0]
     2: aaload

     ;; invoke toStdout
     3: invokestatic #9

     6: return
   }
}
```

#	TYPE	VALUE
1	CLASS REFERENCE	name=#3
2	CLASS REFERENCE	name=#4
3	UTF8 TEXT	"Code"
4	UTF8 TEXT	"sun/misc/MessageUtils"
5	UTF8 TEXT	"toStdout"
6	UTF8 TEXT	"(Ljava/lang/String;)V"
7	UTF8 TEXT	"main"
8	UTF8 TEXT	"([Ljava/lang/String;)V"
9	METHOD REFERENCE	class=#2 signature=#10
10	NAME/TYPE	name=#5 type=#6

TABLE 3: Constant pool for GenClass 3

#	TYPE	VALUE
1	CLASS REFERENCE	name=#3
2	CLASS REFERENCE	name=#4
3	UTF8 TEXT	"Code"
4	UTF8 TEXT	"sun/misc/MessageUtils"
5	UTF8 TEXT	"toStdout"
6	UTF8 TEXT	"(Ljava/lang/String;)V"
7	UTF8 TEXT	"<clinit>"
8	UTF8 TEXT	"()V"
9	STRING CONSTANT	value=#12
10	METHOD REFERENCE	class=#2 signature=#11
11	NAME/TYPE	name=#5 type=#6
12	UTF8 TEXT	"Hello World!"
13	UTF8 TEXT	"sun/applet/Main"
14	CLASS REFERENCE	name=#13

TABLE 4: Constant pool for GenClass 4

#	TYPE	VALUE
1	CLASS REFERENCE	name=#3
2	CLASS REFERENCE	name=#4
3	UTF8 TEXT	""
4	UTF8 TEXT	"sun/security/util/ PropertyExpander"

TABLE 5: Constant pool for GenClass 5

Goodbye Main

At this point, I began to lament the size of the signature of the main method – "([Ljava/lang/String;)V". I decided to try removing the main method entirely and echoing the text in a static initializer block. "<clinit>", the internal name for the static initializer, is a few bytes longer than "main", but the size of the static initializer's method signature "()V" is much shorter than main's. For this to work, I needed to find a class with a main method that doesn't echo any text to the console to extend, so we still have an accessible main method.

The shortest named one I found among the various JRE classes is sun.Applet.Main, the main program for the Java applet viewer application. Applet viewer requires a command input argument, but we can pass in the class file name "Code.class" as a command-line argument. Applet viewer will silently ignore the input since it contains no applet tags. The only drawback to this is that "Hello World!" had to go back into the constant pool, bringing the solution up to 194 bytes (see Table 4). Code to generate this class is in GenClass4.java.

```
public class Code
   extends sun.applet.Main
{
   static {
      ;; sun.misc.MessageUtils
      ;; .toSdout("Hello
World!")

      ;; get String "Hello
World!"
      0: ldc #9

      ;; invoke toStdout on
      ;; sun.misc.MessageUtils
      2: invokestatic #10

      5: return
   }
}
```

OPC – Other People's Code

Despite making the class file larger, this was still an important step. What I needed was to find a way to further leverage hidden classes in the JRE. I'd already found a class with a main method to use that did nothing, allowing the static initializer to do its magic, but what I really needed was a class with a main method that would just print out "Hello World!"

Apparently, that's not such a far-fetched idea. Nestled deep within Sun's 1.3 JRE is sun.security.util.PropertyExpander with the following method:

```
public static void
   main(String args[])
      throws Exception
{
   System.out.println(
       expand(args[0]));
}
```

The expand() method doesn't alter the text "Hello World!", so we're effectively just printing args[0] by itself. As long as we pass in the text "Hello World!" as the first argument to the Java program, as we were doing earlier, we're all set. Since there are longer methods with bytecode associated with the class, the text "Code" is no longer available as a class name. Unfortunately, there are no other text fields to use in the constant pool. Since every class must have a superclass, and a class cannot be its own superclass, we have to add an entry to the constant pool for the name. However, it turns out that the Sun JVM allows for a class to have a zero length name, allowing us to keep the new constant pool entry as small as possible. If this weren't the case, we would have to choose a name like "a".

The code to generate this class file is GenClass5.java. The resulting bytecode is 70 bytes, consisting of four constant pool entries (two for the class spec and two for the superclass spec) and no fields, methods, or attributes (see Table 5).

```
    extends
sun.security.util.PropertyExpander
{
}
```

Of course, we could throw out the generated class file completely and simply invoke Java with the class directly.

```
java
sun.security.util.PropertyExpander
'Hello World!'
```

However, this wouldn't be a legal submission, so the 70-byte solution was the best one I could come up with.

Conclusion

Although hacking class files doesn't have much practical relevance, I found the challenge to be quite a lot of fun. And, even if you have never touched a class file or Java bytecode, the Hello World problem is small enough that you should be able to gain a better understanding of how the internals of Java work.

Acknowledgments

I'd like to thank Jeff Schneider and Momentum Software for devising the Hello World problem and the Austin Java Users Group for sponsoring the contest.

Resources

- *Java Virtual Machine Specification:* http://java.sun.com/docs/books/vmspec/
- *Jakarta Byte Code Engineering Library (BCEL):* http://jakarta.apache.org/bcel/
- Engel, J. (1999). *Programming for the Java Virtual Machine.* Addison-Wesley.

AUTHOR BIO

Norman Richards works at Commerce One Labs, the research division of Commerce One.

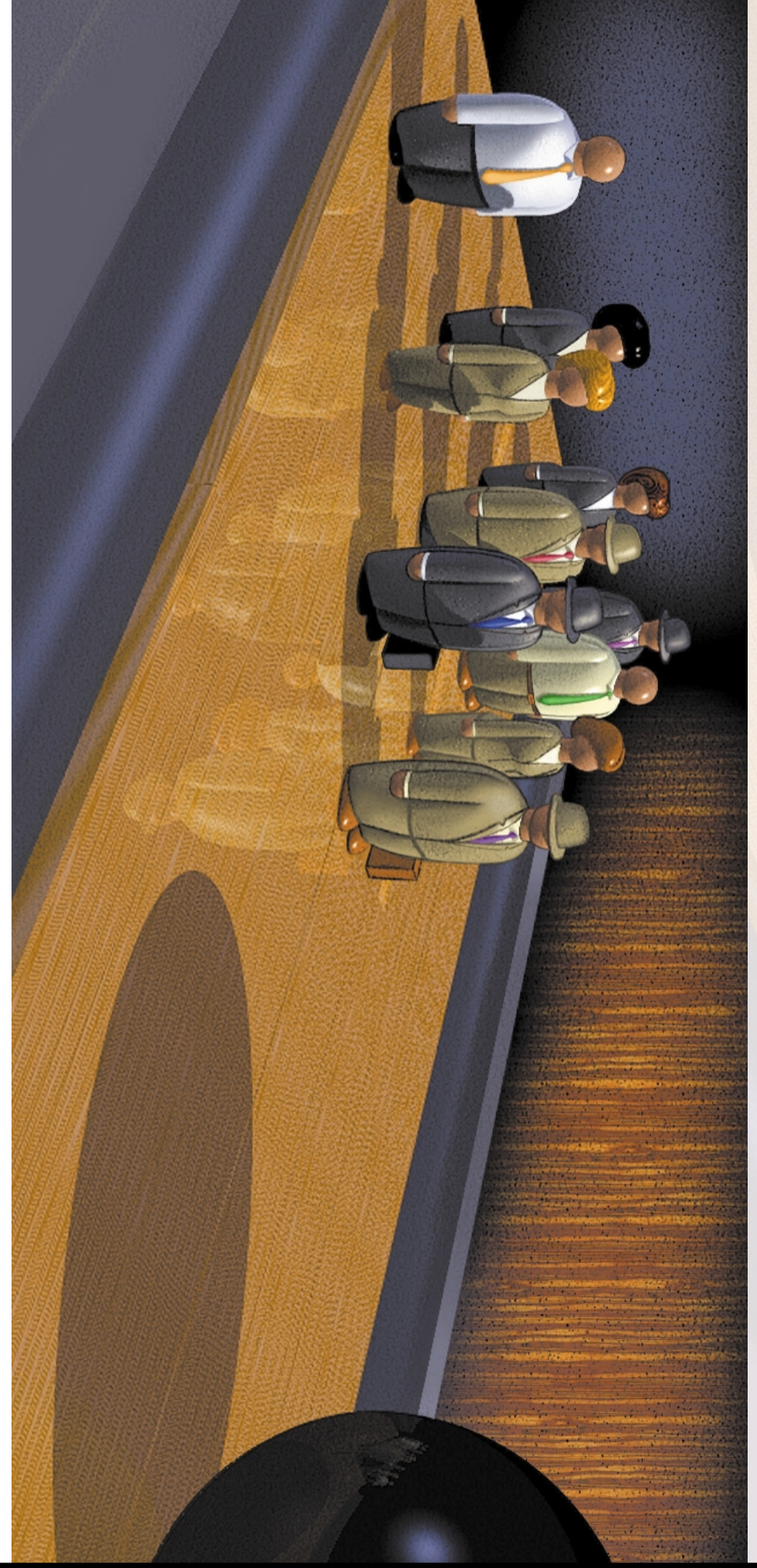
norman.richards@commerceone.com

ZX Spectrum Emulator in Java

RISING TO THE CHALLENGE

written by Razvan Surdulescu

Sir Clive Sinclair had a dream: everyone should own a computer. In the early '80s, this was quite an ambitious, almost foolhardy thing to say, given that the cost of computing machinery was well beyond the grasp of individuals. Despite the hurdles, Sinclair Research Ltd. produced one of the most popular personal computers in Great Britain and, later on, in Europe: the Sinclair ZX Spectrum.



As a child I owned one of these machines, and because of it I am, happily, a computer scientist. Although the Spectrum is no longer produced, my nostalgia for this machine never quite disappeared.

Recently, I embarked on a project to write an "emulator" (in Java) that could run some favorites from the prolific library of Spectrum software. This article is about the challenges encountered and my accomplishments during this adventure. The source code can be downloaded from www.sys-con/java/sourcec.cfm.

What Is an Emulator?

According to Merriam-Webster an emulator is "hardware or software that permits programs written for one computer to be run on another usually newer computer." Emulators are in relatively common use today, although they are invisible – a powerful testimonial to the fact that they do their job well. The Java programming language is popular in large part because it can run on many different computers. This is achieved via an emulator (a "virtual machine") that allows Java programs to be executed on different platforms.

Emulators are particularly satisfying to write. Building an emulator is challenging, since it requires an intimate understanding of both the emulated machine and the host machine in order to bridge them together. Emulators are hard to get right since they require extensive attention to detail: every single facility of the emulated environment, whether it's a CPU instruction or interactions between two components, must work as per spec, or the program running on the emulator will likely fail. Once an emulator is completed, it's possible to revive old programs in such a way that they're completely oblivious to their new surroundings. It's almost like traveling back in time.

Emulation in Java

Java has matured tremendously in the years since I first came into contact with it. Since writing emulators has been an on-and-off hobby of mine, I thought it would be an interesting experiment to see what it would take to implement one in Java.

The first two challenges I had to think about before starting down this path were *performance* and *timing*. Performance is critical to the success of an emulator since no matter how accurate, if an emulator runs programs significantly slower than the original hardware, no one would want to use it. The Java HotSpot engine has recently advanced the performance of Java programs by leaps and bounds, yet the risk still remains: for every instruction of an original program, the emulator may have to execute tens or hundreds of instructions. Timing is also critical since Java does not (yet) have facilities for running programs in real time. Most notably, garbage collection may well interfere with the proper execution of the emulated program and cause significant pauses, making the programs appear jerky onscreen.

On the flip side, Java provides an excellent environment for writing code since it has a powerful and expressive API. The emulator I'll be describing is loosely based on an open-source emulator (implemented in C) that I worked on with a few other people a number of years ago. The Java source is an order of magnitude more compact and more elegant than the original C source. The Java AWT API significantly reduced the burden of implementing the screen emulation (one of the harder aspects of the original emulator). Last, but not least, the emulator can run on any Java Virtual Machine.

The Sinclair ZX Spectrum

The machine I wrote the emulator for is the Sinclair ZX Spectrum: one of the first personal computers. For a bit of history on the machines and the man behind them, visit www.nvg.ntnu.no/sinclair/.

The Java emulator, called "JZX", is loosely based on a Linux native emulator called "XZX" that I worked on a number of years ago (www.zx-spectrum.net/xzx/).

The Spectrum is a remarkably simple machine by today's standards.

- CPU (Z80 @ 3.5MHz), I/O controller (ULA)
- 16K ROM (BASIC), 48K RAM
- Integrated keyboard, TV decoder, loudspeaker
- IN/OUT ports (microphone and headphone), expansion slot

Despite its simplicity, the Spectrum was fully capable of running sophisticated software such as games, Pascal and C compilers, databases, and word processors.

Architecture Overview

The Spectrum is assembled as in Figure 1 and operates as follows:

- The CPU fetches instructions and executes them.
  - –It passes I/O instructions to the ULA.
  - –It handles interrupts from the ULA via interrupt routines.
- The ULA handles the interaction with the outside.
  - –It scans the video RAM to produce the TV frame.
  - –It interrupts the CPU at the end of a TV frame.
  - –It decodes the I/O ports to read/write the peripherals.

The software architecture resembles Figure 1, but is different in a few key aspects (see Figure 2).

The emulator emulates two distinct machines: the original 48K Spectrum model and the subsequent 128K one. Since the machines are similar, 95% of the emulator code base is shared. Not all peripherals from the original Spectrum are emulated: the I/O ports and the speaker are missing. The I/O ports are not present since the original hardware needed them for loading and saving software onto magnetic tape, while the emulator uses files instead. The speaker is not present since it would be almost impossible to emulate it correctly in Java: the sound in the original Spectrum was produced by turning the speaker on and off rapidly to create the appropriate frequency.

Every Java class that represents a Spectrum component is derived from BaseComponent (see Listing 1). BaseComponent and BaseSpectrum form a (extended) Composite pattern, where BaseSpectrum is the Composite and every BaseComponent has a reference to its parent. Any BaseComponent can access its parent and from there, any other sibling. This simulates the "bus" of the original machine. For example, whenever a byte is written into the video RAM, the BaseMemory object can retrieve its BaseSpectrum parent from which it can retrieve the appropriate BaseScreen object and subsequently update the current screen frame.

The BaseComponent class also imposes the contract for the major "lifetime events" of the emulator: startup, reset, shutdown, and load.

- ***init():*** Initializes the component with the parent and the logger object (used for logging error and debug messages)
- ***terminate():*** Terminates the component and indicates that all its state should be discarded
- ***reset():*** Notifies the component to reset all of its state and be ready to start fresh
- ***load():*** Participates with BaseLoader in a simplified Visitor pattern used for loading relevant state into the object (the Accept() and Operation() methods are merged into load() since the Visitor is merely a passive data container)

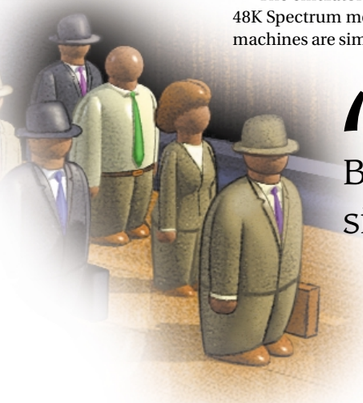
When the top-level BaseSpectrum object is created, it calls init() on itself and all its children, followed by reset(). When the emulator is shut down, it calls terminate() in the same way.

Emulation Challenges

Main Loop

The main loop of the emulator resides in the BaseSpectrum class (see Listing 2).

Every instruction executed by the Z80 CPU takes a certain amount of time, which is a multiple of one "state" (also known as "T-state"). The ULA renders one line of the screen every 224 CPU states (STATES-PER-LINE), so it's essential to keep track of how many states pass every time the CPU decodes and executes one instruction. For efficiency purposes, the screen is not updated every time a new line becomes available, but rather every 50 (TV-LINES) lines, which means one frame every 20 milliseconds.



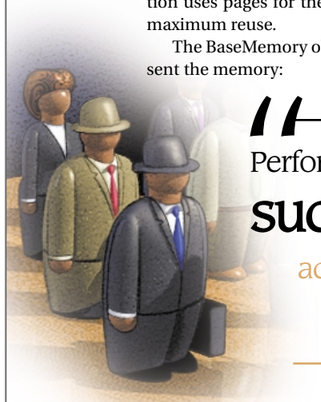
> "Building an **emulator is challenging**, since it requires an intimate understanding of both the emulated machine and the host machine in order to bridge them together"

The CPU interrupts are simulated by means of wait()ing on an external Thread object, which simply sets a public field to "true" and calls notifyAll() every 20ms, at which point the main emulator loop notifies the CPU of the interrupt. The reason for wait()ing on the interrupt is that the CPU emulation runs at the speed of the host machine; no attempt is made to slow it down in order to run at the original Spectrum speed, except whenever a screen frame is rendered. This turns out to be entirely appropriate and makes the emulation both fast and believable. Note that it's possible for the main emulation loop to "skip" one interrupt (if, for example, refreshing the screen takes too long). This is a measurably low risk that would take place only on slower machines and would not be readily visible.

Memory Emulation

The Z80 is a 16-bit CPU, meaning it can index up to 64K of memory. This works naturally for the 48K Spectrum (16K ROM and 48K RAM.) The 128K Spectrum, on the other hand, has 12x16K pages (4xROM + 8xRAM); the software can select any four to be "seen" by the CPU. The BaseMemory implementation uses pages for the emulation in both models to achieve maximum reuse.

The BaseMemory object keeps track of two arrays to represent the memory:



> "Performance is critical to the **success of an emulator** since no matter how accurate, if an emulator runs programs significantly slower than the original hardware, no one would want to use it"

```
byte[][] m_page;
byte[][] m_frame;
```

- The "m_page" array represents the full set of memory pages. There are 12 pages, each 16K long.
- The "m_frame" array represents the four pages currently being "seen" by the CPU. For example, the following code makes the CPU "see" pages 0, 4, 6, and 9:

```
m_frame[0] = m_page[0];
m_frame[1] = m_page[4];
m_frame[2] = m_page[6];
m_frame[3] = m_page[9];
```

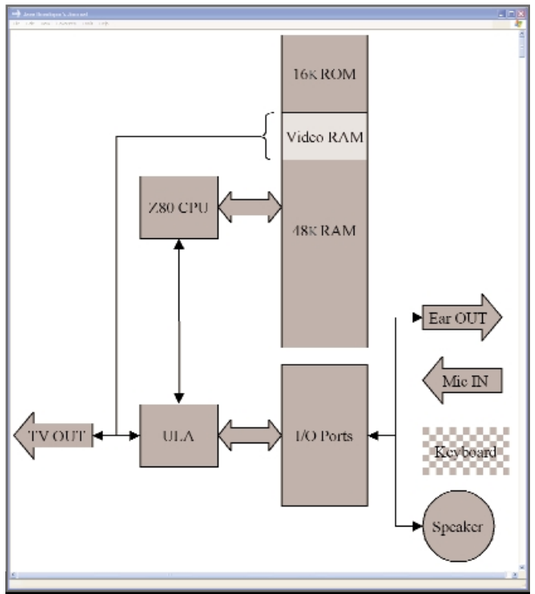


FIGURE 1 The Spectrum

The emulated CPU reads and writes data by indexing into the "m_frame" array.

The BaseMemory object allows the CPU to select the "visible" pages via the method "public void pageIn(int frame, int page)". It also allows direct access to the page data via "public byte[] getBytes(int page)" (useful for the screen emulation, for instance).

All memory operations must convert "virtual" addresses to physical ones. The frame number is simply the first 2 bits of the "virtual" address; the frame offset is the remaining 14 bits (see Listing 3).

Signed and Unsigned Data Types

You may be wondering about the return types of the methodes in Listing 3; they both return an integer (32 bits) despite the fact that they should perhaps return a "byte" (8 bits) and, respectively, a "char" (16 bits).

The Z80 CPU can operate on 8-bit or 16-bit values, either directly or via its registers. Although Java natively supports data types that are 8- and 16-bits wide, the emulator is implemented almost exclusively in terms of integer types. The reason for this is that the Java byte is a signed type, while the Java char is an unsigned type.

Consider the following (fictitious) Z80 instruction that adds the contents of the A register (8 bits) to the contents of the HL register (16 bits) and stores the results in the HL register:

```
Input:
A = 10000000 (=0x80), HL = 00000000 00000001 (=0x0001)
Result HL = HL + A:
HL = 00000000 10000001 (=0x0081)
```

The same thing in Java would look like this:

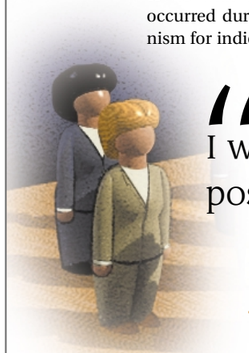
```
Input:
byte a = (byte) 0x80; char b = (char) 0x0001;
Result b = b + a:
b = (char) (b + a); (=0xFF81)
```

Surprised? The Java language specification (paragraph 5.6.2) mandates that all binary operations where the operands are of type integer (or smaller) should be promoted to integer first. In our case, the (byte) value 0x80 and the (char) value 0x0001 are first promoted to integer before they're added. Since the byte is a signed type, the integer promotion yields the value 0xFFFFFF80. Since the char is an unsigned type, the integer promotion yields the value 0x00000001. When the two integer values are added, the end result is 0xFFFFFF81, which is then truncated to a char, yielding the value 0xFF81. The only way to avoid this behavior is to explicitly prevent the sign extension in the widening conversion. The new code would look something like this:

```
b = (char) (b + (a & 0xFF));
```

The "&" will mask all bits but the last 8, yielding (the integer) 0x80, which is then added to "b", producing the correct result.

Although this solution solves the problem, it's only a partial solution for the CPU emulation as a whole. The reason has to do with the CPU flags that indicate whether overflow occurred during a particular operation. Java has no mechanism for indicating overflow, so I must always use a Java data type that's larger than the resulting value. The emulator would explicitly test for overflow and truncate appropriately. In the end, the only feasible solution is to use integer types everywhere, and explicitly deal with issues of truncation and overflow.



"I was pleasantly surprised to see that it was not only possible to implement a **Java emulator** for the Spectrum, but that it also ran fast"

To keep the code readable, I adopted a naming convention that exposes the size of the data types involved. The size in bits of the return value and each argument of a function is appended to its name. For example, "int read8(int val16)" means that the function returns 8 bits of data, and receives as an argument 16 bits of data, all embedded in an integer as the less significant bits. Furthermore, the convention is that all input arguments are correct and need no further modifications, while all return values need to be correctly truncated before being returned.

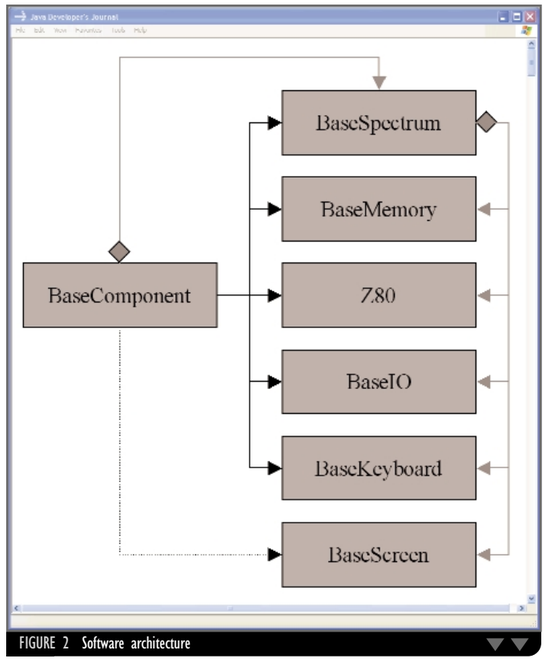


FIGURE 2 Software architecture

CPU Emulation

In addition to the signed/unsigned challenges described earlier, the CPU poses additional problems in the area of instruction decoding. The decoder is implemented as a large "switch()" statement, which switches on the first byte of the current instruction. Naturally, the code is very large and rather unwieldy to read and modify. One possible solution for dealing with such a large piece of contiguous code would be to have an array of IRunnable objects (that can decode a particular instruction in the "run()" method) indexed on the instruction code.

This approach would allow the code to be structured more elegantly, but it would proliferate the number of classes and impose significant runtime overhead. The "switch()" approach, while difficult to write and maintain, is extremely fast since the JVM implements it internally as a jump table, thereby exhibiting the same architectural approach as the object array, without the performance penalty of invoking an interface method for every instruction.

Screen Emulation

Each pixel on the Spectrum screen can be either on or off. This is represented by the appropriate bit value in a byte (the state of 8 adjacent pixels is governed by the byte value at a particular memory address in video RAM). Color information is represented by another byte.

To draw pixels on the screen, the BaseScreen object extends java.awt.Canvas and implements the drawing logic in the paint() method. A nice side effect of this is that the emulator can be "embedded" into any AWT or Swing container that can render Canvas objects. This allows the emulator to run seamlessly as a standard application or an applet. The BaseScreen object uses an offscreen image to render the screen contents, after which the image is drawn directly onto the screen via java.awt.Graphics.drawImage() (this common technique prevents flickering).

Every time the CPU writes into the screen memory area, the BaseScreen object is notified. For maximum efficiency, the only action taken at this time is to toggle a Boolean value in an array that indicates that the particular screen byte has changed. When paint() is called, a for loop iterates through the Boolean array and, for every "true" value, it draws the corresponding byte into the offscreen image.

The mechanism for fast updates to the offscreen image is the challenging part. The naïve technique is simply to use java.awt.Graphics.fillRect() to render every pixel into the

image. While this works, it's very slow due to the overhead of calling the fillRect() method and running it many times for 1x1 rectangles.

A better technique is to create the offscreen image as a decorator for a java.awt.image.MemoryImageSource object. The MemoryImageSource is created, in turn, containing a byte array in RGB format with the pixel data. The rendering code updates the byte array and then calls MemoryImageSource.newPixels() to notify the object that the data has been updated (see Listing 4).

Table 1 provides the timing results, in milliseconds, for rendering 200 consecutive frames of the (same) Spectrum game (the hardware/software configuration is Windows 2000 Professional, Pentium III 650, 192MB RAM).

These timings are barely adequate: as you recall, each Spectrum frame is refreshed every 20ms. If rendering the frame takes longer than 20ms, the emulation will look choppy (it will skip frames and slow down the machine overall).

To improve performance, I made a key observation about the way color is encoded in the Spectrum. As described earlier, every byte in video RAM is paired with another byte that describes its color: the first byte simply shows whether the pixels are "on" or "off" (the "pixel" byte) and the second byte shows what color the pixels are (the "color" byte). This means there are a total of 256 * 256 different ways that a location in video RAM could appear on the emulated screen (256 values for the "pixel" byte and 256 values for the "color" byte). I can prerender some (fixed) number of these "pixel/color" byte combinations as Java image objects and then simply use java.awt.Graphics .drawImage() to render that piece of the video RAM on the screen (see Figure 3.)

The new performance numbers are in Table 2.

As you can see, the performance improvements are dramatic for Java1 (>90%); for Java2, however, the performance is far worse (a slowdown or more than 1,000%!). The reason for the bad Java2 performance lies in the performance of java.awt.Graphics.drawImage(). This discussion is beyond the scope of this article, but you can read more about it on the Java Developer Connection Web site (http://developer.java.sun.com/developer/) in the BugParade section (http://developer.java.sun.com/developer/bugParade/bugs/4276423.html).

To resolve the performance problems in Java2, I use a different (and more conventional) technique that's similar to the MemoryImageSource technique described earlier. In Java2, the offscreen image object is a superclass of java.awt.Image, namely a java.awt.image.BufferedImage. This class has a method called setRGB() that allows you to set an RGB pixel array directly into the Image object, without the performance penalties of MemoryImageSource. newPixels().

The final performance numbers are in Table 3.

Note that prerendering all possible 256 * 256 (= 65536) Java image objects will take a toll on the memory footprint of the emulator. If I want a fixed-size cache of these images, I run the risk of "thrashing" in the cache. Discarding entries when the cache is full means the garbage collection will have more work, slowing down the emulation. It's possible to reuse entries in the cache (instead of discarding them), but this will bring us back to the original performance problems with

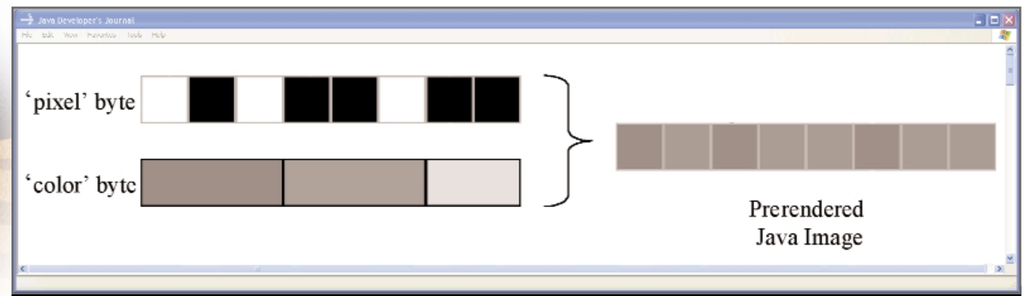


FIGURE 3 Prerendered Java image

TECHNIQUE	SUN JAVA 1.1.8_03	MICROSOFT JVIEW 5.00.3802	SUN JAVA 1.3.1
MemoryImageSource	23.93526824	24.0485756	20.20600913

TABLE 1: Timing results

TECHNIQUE	SUN JAVA 1.1.8_03	MICROSOFT JVIEW 5.00.3802	SUN JAVA 1.3.1
MemoryImageSource	23.93526824	24.0485756	20.20600913
Prerendered Java Images	0.989330465	0.939373495	N/A

TABLE 2: New performance numbers

TECHNIQUE	SUN JAVA 1.1.8_03	MICROSOFT JVIEW 5.00.3802	SUN JAVA 1.3.1
MemoryImageSource	23.93526824	24.0485756	20.20600913
Prerendered Java Images	0.989330465	0.939373495	N/A
BufferedImage	N/A	N/A	5.995489495

TABLE 3: Final performance numbers

Listing 1

```
public abstract class BaseComponent
{
 protected BaseSpectrum m_spectrum;

 public BaseSpectrum getSpectrum() {
  return m_spectrum;
 }

 public void init(BaseSpectrum spectrum) {
  m_spectrum = spectrum;
 }

 public void terminate() {
  m_spectrum = null;
 }

 public abstract void reset();

 public abstract void load(BaseLoader loader);
}
```

Listing 2

```
while (true) {
 states = current CPU state count
 if (states >= STATES-PER-LINE) {
  states = (states - STATES-PER-LINE)

  increment line count
  if (lines == TV-LINES) {
   lines = 0
   refresh screen

   wait for the next clock interrupt
   interrupt the CPU
  }
 }

 execute next CPU instruction
}
```

Listing 3

```
public int read8(int addr16) {
  int data = m_frame[(addr16 >> 14)][(addr16 & 0x3FFF)];
  return (data & 0xff);
 }

 public int read16(int addr16) {
  int high = (read8((addr16 + 1) & 0xFFFF) << 8);
  int low = read8(addr16);
  return (high | low);
 }
```

Listing 4

```
private int[] m_data;
private MemoryImageSource m_memoryImageSource;

// RGB data
m_data = new int[SCREEN_WIDTH * SCREEN_HEIGHT];
m_memoryImageSource = new MemoryImageSource(SCREEN_WIDTH,
SCREEN_HEIGHT, m_data, 0, SCREEN_WIDTH);
m_memoryImageSource.setAnimated(true);

public void paint(Graphics g) {
 for every 'address' in Video RAM {
  let 'pixels' be the byte at that address
  for every 'bit' in 'pixels' {
  let '(x, y)' be the Cartesian coordinates of 'bit'
  let 'color' be the RGB color of 'bit'
   m_data['(x, y)'] = 'color'
  }
 }

 m_memoryImageSource.newPixels();
 g.drawImage(m_offscreenImage, 0, 0, this);
}
```

Download the Code! www.JavaDevelopersJournal.com

Graphics.drawRect() or MemoryImageSource.newPixels(). A better idea is to use only half the "pixel" byte (a nibble) to prerender Java images. This means that any "pixel" byte will be drawn by concatenating two prerendered Java images. The total number of prerendered nibbles is far more manageable: 16 * 256 (=4096.) The tradeoff is that I now need to make twice as many calls to java.awt.Graphics.drawImage(), but that turns out to be inconsequential.

Debugging Techniques

Debugging the emulator is very challenging. The hardest part to debug is the CPU emulation, primarily due to its sheer size and complexity. The CPU emulation code is bigger and more complicated than the rest of the emulator. Although the Z80 CPU is simple by today's standards, it has a great many flags, registers, and instructions that manipulate these flags. For example, any addition will modify the sign flag, parity flag, carry flag, half-carry flag, and add-subtract flag. The Spectrum software uses all these flags, and any mistake most often translates into a "hard reset" of the emulated Spectrum.

Furthermore, determining exactly where the emulated software crashed is not as easy as watching for the equivalent of an illegal memory access or page fault. (There's no such thing on the Spectrum.) An incorrectly decoded instruction will most often translate to a Spectrum "hard reset" or "hang" thousands of instructions down the stream from it.

The easiest way to debug the emulator is to use another emulator (that's known to be correct) and compare the CPU traces for executing the same program. In this case, I modified the original Linux native emulator to output CPU traces (a CPU trace is the state of all registers and flags after executing every instruction). This has the advantage of pinpointing the precise spot where the Java emulator diverges from the native emulator and thus dramatically reduces the time required for debugging.

Performance Considerations

I was pleasantly surprised to see that it was not only possible to implement a Java emulator for the Spectrum, but that it also ran fast. The CPU emulation is on par with the native emulation; the screen emulation, while slightly slower and a bit more awkward, is well within the limits of realistic emulation for what I would consider "average" hardware. Surprisingly, and most probably due to screen emulation, Java2 didn't fare dramatically better than Java1, which puts Java1 on the map as a reasonable contender for this type of work.

Conclusion

As a platform for emulation, Java is a very strong player. Although the Sinclair Spectrum is not a terribly complex machine by today's standards, it poses significant challenges in its implementation, and requires strong support both in terms of language features and overall performance. Java's elegant and expressive language rises to the challenge and overcomes it easily with code that's more readable, more modular, and far more concise. Java's performance, the wild card in the equation, also meets expectations.

Author Bio

Razvan Surdulescu is a software developer at Trilogy in Austin, TX, where he writes e-business software in Java.

razvan.surdulescu@post.harvard.edu

JASON R. BRIGGS J2ME EDITOR

Books and Chewing Gum

A recent press release from Palm got me thinking about their PDAs, as well as why Palm (in the UK) never returned my e-mails...but that's another matter (and half a world away now). In any case, according to the release, 5,000 Palms are to be purchased as part of a three-year grant program for several New York State school associations. Apart from wishing I was the salesperson who got the commission on that particular contract, I wonder whether Java is in the equation for any application suites being developed (if they are) as part of that system roll out. And if not, why not?

If Java is not being considered, a colleague of mine thought of a reason. Polish. No, not people from Poland – the other meaning/pronunciation (i.e., to remove flaws from; to perfect or complete). To paraphrase his words – the JVM for the Palm suffers from a certain lack of polish compared to other applications available for the OS.

Personally, I think the Palm JVM has really improved since I first looked at it, and it's definitely better than VMs I've seen running on Windows CE, in terms of integration with the device, for example.

Java has a history of this "lack of polish" on the client side. Compare the hiccup that is an applet starting up in your browser with the smoothness of Macromedia's Flash and you'll know what I mean. However, in comparison with desktop Java, I don't think the Palm VM is really that bad, so I hope that whoever is advising the associations is giving the idea some serious thought.

One of the killer applications for the Palm seems to be electronic books. Considering the price of a number of eBook "readers" I've seen, PDAs, like the Palm, are a cost-effective alternative – so it's hardly surprising that eBooks are doing relatively well. However, the major problem I see with the whole concept of downloading books is that I quite like wandering down to a bookstore, picking a title off the shelf, and reading a few pages to see if I like it. Not that I don't also like browsing online, but the traditional retail book-buying experience definitely still has its allure. I'm not sure if I'm exactly representative of society as a whole, but I guess there's a good percentage of people out there who feel the same.

Stick with me here – I'm going off on a tangent.

I recently read on CNET that Sony is endeavoring to make their memory stick a de facto standard. I'm sure you've come across these odd little storage cards before, but in case you haven't – imagine a stick of Wrigley's chewing gum and you're halfway there. Whether or not they succeed in becoming the market standard, I have to say I quite like the memory stick. They seem a lot more convenient to lug around in larger numbers (and how is that pickpocket supposed to know that the pack of chewing gum in my pocket isn't actually chewing gum...unless of course they like chewing gum...?).

So here's the crazy idea for the month. The old "walk-in-and-browse" bookshops remain the way they are, but with a slight difference. Rather than stocking books, they stock something like a pamphlet – a standard book jacket (with cover and summary on the back) plus a few excerpted pages inside. It would work in a similar fashion to the way music stores do now – you take the cover up to the counter and they pull a CD out of a cupboard – except in this case, they pull a memory stick out of the cupboard with the book loaded on it. Those who want to browse in a shop can still do so – those who want to browse online can still do their purchasing on the Net. People like me, who enjoy doing both, are happy as well. And the humble PDA (Palms, Handsprings, Cliés, and the like) will make an admirable reader until the various companies inventing electronic paper (or whatever they're calling it) bring the technology to market.

Now to make sure that the infrastructure for all this is implemented in Java… ✎

jasonbriggs@sys-con.com

AUTHOR BIO

Jason R. Briggs is a Java analyst programmer and – sometimes – architect. He's been officially developing in Java for almost four years, "unofficially for five."

J2ME INDEX

A Smart Card's Place in a Security Architecture

WRITTEN BY KEN GREENWOOD

At the recent JavaOne conference in San Francisco, SchlumbergerSema demonstrated the benefits of a Java-based smart card. "A Java what?" was a common response by visitors. A brief explanation showed that anyone with a GSM mobile phone, and many with a credit card, carry one of these around all the time. In an age when processor speeds are measured in gigahertz, it's often difficult to think on a slightly smaller scale.

A smart card is, in essence, a computer that can be carried around in your pocket. It has input/output and power – via a set of metallic contacts – a microprocessor (32 bits will soon be common), ROM, EEPROM (64K in recent models), and an operating system (OS). For a Java-based card the OS, as defined by Java Card 2.1 standards, is a slimmed-down version of the OS found on a larger computer. The flexibility, security, ease-of-use, and rapid development cycle of Java technology has made it the leading open standard for the smart card industry.

"But what does it do?" you might ask. As with most computers, the answer to this question is, "What do you want it to do?" Smart cards have been in use for over 20 years, although only fairly recently have they been able to run an actual operating system. Because smart cards are embedded with a microprocessor, they can store large amounts of data and carry out their own card functions, such as encryption of digital signatures.

A smart card communicates with a host computer through a card reader, which can generally be connected to a USB, RS232, or PCMCIA port.

Although widely used in GSM mobile phones, the Java Card is a relative newcomer to the network security field. Despite the many advantages of smart card technology, the cost of the reader has been a restraining factor. There are significant advantages, however, and with the cost of readers going down and the introduction of direct-to-USB port technology, the strong value of smart cards as an easy-to-use, portable, and very secure means of logical identification is beginning to be better understood by those outside the immediate industry. With the need for security – both physical and network – becoming ever more critical, it's clear that a portable device with a secure memory is a good investment.

Recent laws in many countries have made electronic signatures a reality and Public Key Infrastructures (PKIs) are becoming more common. They rely on a pair of secure keys that make up a person's digital identity. One of these keys is the "public key," which can be seen and used by anyone to check the authenticity of a document signed using the corresponding "private key." As the name suggests, the private key must be kept secret at all times. A smart card offers the ability to securely create the two keys onboard the card itself, ensuring that the private key is never visible to the outside world. The use of this key to sign or decrypt a message is, again, always done on the card.

While it's possible to do these operations using a computer's hard drive, there are too many worms and viruses to make this a secure alternative. In addition, the common practice of writing down passwords or making them easy to remember is a serious flaw in any security architecture. By using a well-defined structure and communication within the card, it's possible to make certain that there's no access to the secret memory without the correct authorization. A multifactor authentication ensures that, to use the card (and therefore the private key), it's necessary to have the card with you and know the password. A third factor can even allow authentication using a biometric application, such as a fingerprint or face recognition.

Futurists once predicted we would carry computers in our pockets – with smart cards, we already do. Security has become paramount in the global consciousness, and Java-based smart cards offer a secure, mobile, practical, and affordable means of providing physical and information security.

More information about smart cards can be found at www.smartcards.net.

References

- *Smart Card Developer's Kit:* www.scdk.com.
- Guthery, S. and Cronin, M. (2001). *Mobile Application Development with SMS and the SIM Toolkit.* McGraw-Hill.
- Anderson, R.J. (2001). *Security Engineering: A Guide to Building Dependable Distributed Systems.* John Wiley & Sons.

Author Bio

Ken Greenwood is the business marketing manager for SchlumbergerSema Cards and Transactions.

kgreenwood@slb.com

SYS-CON MEDIA

PUBLISHER, PRESIDENT, AND CEO
FUAT A. KIRCAALI fuat@sys-con.com
CHIEF OPERATING OFFICER
MARK HARABEDIAN mark@sys-con.com
VICE PRESIDENT, BUSINESS DEVELOPMENT
GRISHA DAVIDA grisha@sys-con.com

ADVERTISING
SENIOR VICE PRESIDENT, SALES AND MARKETING
CARMEN GONZALEZ carmen@sys-con.com
VICE PRESIDENT, SALES AND MARKETING
MILES SILVERMAN miles@sys-con.com
ADVERTISING SALES DIRECTOR
ROBYN FORMA roybn@sys-con.com
ADVERTISING ACCOUNT MANAGER
MEGAN RING megan@sys-con.com
ASSOCIATE SALES MANAGERS
CARRIE GEBERT carrieg@sys-con.com
KRISTIN KUHNLE kristen@sys-con.com
ALISA CATALANO alisa@sys-con.com
LEAH HITTMAN leah@sys-con.com

EDITORIAL
EXECUTIVE EDITOR
NANCY VALENTINE nancy@sys-con.com
EDITOR
M'LOU PINKHAM mpinkham@sys-con.com
MANAGING EDITOR
CHERYL VAN SISE cheryl@sys-con.com
ASSOCIATE EDITORS
JAMIE MATUSOW jamie@sys-con.com
GAIL SCHULTZ gail@sys-con.com
JEAN CASSIDY jean@sys-con.com
ASSISTANT EDITOR
JENNIFER STILLEY jennifier@sys-con.com
ONLINE EDITOR
LIN GOETZ lin@sys-con.com

PRODUCTION
VICE PRESIDENT, PRODUCTION AND DESIGN
JIM MORGAN jim@sys-con.com
LEAD DESIGNER
LOUIS F. CUFFARI louis@sys-con.com
ART DIRECTOR
ALEX BOTERO alex@sys-con.com
ASSOCIATE ART DIRECTORS
CATHRYN BURAK cathyb@sys-con.com
RICHARD SILVERBERG richards@sys-con.com
AARATHI VENKATARAMAN aarathi@sys-con.com
ASSISTANT ART DIRECTOR
TAMI BEATTY tami@sys-con.com

WEB SERVICES
WEBMASTER
ROBERT DIAMOND robert@sys-con.com
WEB DESIGNERS
STEPHEN KILMURRAY stephen@sys-con.com
CHRISTOPHER CROCE chris@sys-con.com
CATALIN STANCESCU catalin@sys-con.com

ACCOUNTING
CHIEF FINANCIAL OFFICER
BRUCE KANNER bruce@sys-con.com
ASSISTANT CONTROLLER
JUDITH CALNAN judith@sys-con.com
ACCOUNTS RECEIVABLE
JAN BRAIDECH jan@sys-con.com
ACCOUNTS PAYABLE
JOAN LAROSE joan@sys-con.com
ACCOUNTING CLERK
BETTY WHITE betty@sys-con.com

SYS-CON EVENTS
VICE PRESIDENT, SYS-CON EVENTS
CATHY WALTERS cathyw@sys-con.com
CONFERENCE MANAGER
MICHAEL LYNCH mike@sys-con.com
SALES EXECUTIVES, EXHIBITS
MICHAEL PESICK michael@sys-con.com
RICHARD ANDERSON richard@sys-con.com

CUSTOMER RELATIONS/JDJ STORE
MANAGER, CUSTOMER RELATIONS/JDJ STORE
ANTHONY D. SPITZER tony@sys-con.com
CUSTOMER SERVICE REPRESENTATIVE
MARGIE DOWNS margie@sys-con.com

Java Card 2.2 Specifications Overview

Java Card engineering with RMI specifications

WRITTEN BY
JOSEPH SMITH

Ever since the Java Card 1.0 was introduced in 1996 it has been gradually maturing, and recently celebrated its fifth anniversary. American Express, Visa, and now the Department of Defense have all deployed solutions that utilize the Java Card specifications.

This rather large movement proves that Java Card technology is a great system for data security and data mobility. At last count, American Express claimed over 4 million deployed cards and Visa estimated over 7 million at the end of last year.

Since September 11, 2001, there has been a lot of interest regarding Java-based smart cards combined with biometrics (verifying the identity of a person through some physical characteristic). This has driven the Java Card specifications to the forefront, and more and more engineers (and managers) are scrambling to implement them.

Currently, Java Card implementers use Java Card 2.1.x. Soon you'll be seeing Java Card 2.2 deployed into the marketplace. Those who have started to use the Java Card 2.1.x API are realizing that it's quite a hassle to do some basic things. For example, you can't utilize strings or other common primitive types found in Java because the Java Card processor has extremely limited resources. Most cards offer a little less than 32K of memory for applets. (*Note:* Don't confuse applets with J2SE applets. Applets refer to applications that reside on the Java Card.) Another deterrent is that you have to deal with bytes! When was the last time you had to think in bytes and byte arrays?

Java vs Java Card

The Java Card contains a virtual machine that's tailored for it, the Java Card VM. Because of the limitations, the JCVM has some unsupported features such as:

- Dynamic class loading
- Garbage collection
- Threads
- Cloning
- Default visibility override
- Package visible interface can't be extended to public visibility
- Int, double, float, and long are not supported
- Static instantiated classes aren't allowed

The JCVM differs from standard JVMs in several ways:

- There's only one system class, the javacard.framework.JCSystem.
- Persistence and transience behavior are different from Java.
- Only single-dimensional arrays are allowed.
- Security management policy is implied in the JCVM and not an explicit class.
- The JCVM is a 16-bit system.

With that said, there are some useful supported features such as:

- Packages
- Virtual methods
- Interfaces, abstract classes, and exception handling
- Polymorphism
- The root class is of class Object

Other limitations that should be mentioned include:

- A maximum of 256 instance fields per class
- Array size is limited to 32K
- Memory space is limited to 32K per package

To communicate with a card, use a Card Acceptance Device (CAD). This is usually a card reader but can also refer to an ATM or point-of-sale terminal. Your communication isn't really to the card itself – it's actually to the CAD, which then sends commands to your applet.

To make engineering more complex, the structure of the applet code appears a bit backward: you use a large switch case block to process the commands you want. However, Java Card 2.2 makes all this a bit easier.

> "With the new Java Card specification, existing features become more powerful"

Java Card Remote Method Invocation

One thing that will make engineering easier is the use of Remote Method Invocation. An engineer will no longer have to spend a lot of time learning various APDU commands.

Currently, you have to issue commands in the following format:

CLA INS P1 P2 LC CData

- ***CLA:*** A class byte that can be an ISO 7816-4 such as 80, 84, or user-defined
- ***INS:*** An instruction byte that states which instruction to perform
- ***P1/P2:*** Parameter 1/2 that subdivides a command
- ***LC:*** Length of data to be transmitted
- ***CDATA:*** The data to be transmitted
- ***LE:*** Length of data to be returned from the applet

Depending on the applet, it would have to switch case on each individual CLA, INS, and P1/P2 to process the necessary command. This method of transmitting data can be a tedious job and a maintenance nightmare. The Java Card RMI makes it easier to perform a command.

Author Bio

Joseph Smith is a senior software engineer with over 13 years of experience. In the last two years, his area of expertise has been in Java Card and biometric-based solutions. He can frequently be found answering questions on Sun's Java Card Developer Forum.

More Algorithms

For the implementation of security algorithms we must rely on the individual vendors. Some vendors implement cards that contain just DES and Triple DES, while others implement DES, Triple DES, and RSA. For the implementation of RSA it's common to also include a crypto-coprocessor in the smart card chip to help speed up the calculations.

In Java Card 2.2, AES and elliptic curves are added. New key lengths are also available with these new algorithms. Of course, this can be a headache involving export restrictions, so check with your vendor to see what's actually implemented and what key lengths they support.

You can still perform digital signature and random number generation on the card as you did with Java Card 2.1.x.

Applet/Package Deletion

Applet/package deletion has been a hot topic since the first release of the Java Card. This feature of 2.2 will be up to individual vendors to support, utilizing their own implementation. But nonetheless, it's sure to be available since engineers and others will be requesting the functionality. Applets created using the Applet.register() methods exist until deleted by the applet deletion manager. This manager can be an applet provided by the vendor and is selectable by providing an application identifier, AID. Sun wanted to leave the implementation of the applet deletion manager up to the vendors and, because of that, the AID won't be standardized but it must be SELECTable. As long as the vendors provide a SELECTable behavior to the outside world, they can implement the applet deletion manager any way they see fit for their Java Cards.

Memory Resource Management

The JCSystem class offers two new methods: isGarbageCollectionSupported and requestGarbageCollection. These offer limited garbage collection support to query if a vendor has implemented garbage collection functionality and to request that the action be carried out. Garbage collection isn't a requirement, and once again it's up to the implementers. However, you can now query the amount of memory available to the applet, which wasn't available in 2.1.x.

JCSystem.getAvailableMemory is a nicer way to query if memory is available before you allocate it. In 2.1.x, you cross your fingers and hope that the memory is available when you allocate it, otherwise you'll receive an exception.

Logical Channels

In Java Card 2.1.x your commands are processed by the currently selected applet. Java Card 2.2 allows up to four sessions to be open, one session per logical channel. Now applet instances can be selected on different logical channels. The applet can take advantage of multisession functionality and can be concurrently selected with another applet on a different logical channel. The applet can even be selected multiple times. There is a basic logical channel, logical channel 0, that is activated upon a card reset. A card reset can occur when the card has been powered down via card removal or electrical failure.

Applets that are capable of being selected on multiple logical channels at the same time are called multiselectable applets. These will have the same security constraints as in Java Card 2.1.x with regard to active contexts. To forward APDU commands to a logical channel, the command will contain encoding in the header CLA byte to denote which channel. For example, CLA byte 0x02 could denote logical channel 2, and 0x00 could denote logical channel 0. The logical channel is of key importance for wireless support involving 3GPP, WAP, and ETSI.

Conclusion

Java Card 2.2 includes support for contactless smart cards. These cards don't have the visible chip on them, but can be read if the card is simply waved within a certain proximity of a contactless CAD. There has been movement to include some form of biometry in the Java Card API, and this technology will no doubt be appearing on the market soon.

With the new Java Card specification, existing features, such as transaction processing (much like commit/rollback in a database), become more powerful.

Personally, I'm excited about the RMI specification and memory management features that will make complex Java Card engineering a lot friendlier.

java_card@hotmail.com

Optimizing Java Performance in Heritage Designs

The advantages and disadvantages

by Carl Barratt

Java, in its J2ME guise, has all the attributes of a first-rate platform for embedded system design. More specifically, its platform independence, code portability, and robust operation render it particularly suited to such applications. The extensive use of embedded Java-based devices in the future is secure due to the proliferation of standards based on it, and, moreover, the endorsement of major OEMs committed to its use in their designs.

It's become clear that the potential marketplace for embedded Java devices is vast, but that some of these markets are not yet mature. Successful manufacturers in the immediate market for embedded devices, such as wireless handsets and set-top boxes, possess a huge investment in legacy code that they, not unreasonably, wish to retain. Along with the problem of generating acceptable performance in resource-constrained environments, the migration to Java-enabled devices in markets that are already established and based on other technologies is the most significant barrier to the widespread adoption of Java as the de facto standard in the embedded space.

Of the emerging solutions, both hardware- and software-based, none can claim to be a panacea. This article discusses the introduction of Java into multilanguage heritage designs, focusing on the advantages and disadvantages of deploying each solution.

Java Bytecode Execution in an Embedded Environment

Obviously, some platforms will be more proficient than others at executing Java code. The issue is clouded by hype, but, fundamentally, Java bytecode can be executed in one of three ways: software translation, hardware translation, or direct execution.

Translation in Software: The Java Virtual Machine

Bytecode can be executed using a software Java Virtual Machine (JVM) or, more specifically, a KVM designed particularly for embedded devices. Java code can be executed on any such virtual machine. A JVM takes the precompiled Java source code (bytecode) and translates it into the native machine code of the processing platform in question preceding its execution. Indeed, this process of interpretation is central to the Java concept of platform independence.

Translation in Hardware: Bytecode Accelerators

A bytecode accelerator is a hardware solution that uses the resources of an existing host processor. Accelerator solutions don't execute Java bytecode directly; instead, they convert the bytecode (in hardware) into the native instructions of the host processor prior to execution. Invariably, such solutions also utilize a software-based JVM, modified by the replacement of the main interpreter loop and execution unit with the bytecode accelerator.

Native Java Processors

Native Java processors are microprocessors designed to execute bytecode directly as their native instruction set. They

can be deployed as a coprocessor to a host processor in a multilingual, multiprocessor system, or as a standalone solution in a dedicated embedded Java design.

Embedded Multiprocessor Java Solutions

While there's a clear desire for Java capabilities to be introduced into many embedded applications, it's a prerequisite that Java bytecode is executable in parallel with existing heritage code, rather than in place of it. Primary examples of such applications would be a mobile phone running a C-coded communications stack, or a set-top box currently evolving to support interactive or Internet-based content. Understanding the fundamental design issues is vital when designing high-quality embedded devices for Java-based applications. Characteristics that influence the selection of components for any embedded system include:

> "Understanding the fundamental design issues is vital when designing high-quality embedded devices for Java-based applications"

- Resources
- Performance
- Ease of integration
- Cost

First, JVMs are inherently resource hungry. This is a corollary of the software interpretation layer, which abstracts the code, and the processor upon which that code is executed. A JVM will typically map a single Java bytecode into several native processor instructions prior to execution; therefore, to sustain acceptable Java performance, a very fast processor is required. Relatively speaking, the rise in silicon cost and power consumption intrinsic in the use of such powerful processors is huge. Additional memory resources, occupied by the JVM itself, present a further burden for embedded applications.

Bytecode accelerators also use a JVM and so require the same additional memory resources as software-only JVM solutions. Typical bytecode accelerators are efficient in terms of silicon cost when added to an existing host processor; however, if a second dedicated processor is used, the gate count of this additional processor must also be taken into account. Native Java processors vary drastically in size. Those that are stack-based, and thus accurately match the Java execution model, have a very low silicon cost, whereas those based on a standard RISC processor are less than optimal.

Since there are still no dependable metrics available to evaluate the performance of embedded Java solutions, code execution speed remains an emotive issue. When applied prudently, benchmarks are an invaluable asset. However, they're not the sole criteria for evaluation and must be regarded with caution since ultimately the crucial point is how fast the platform can execute the end application code. CaffeineMark figures are widely quoted but are not representative of real applications. It's hoped that the imminent arrival of EEMBC industry-standard benchmarks will clarify the issue as discussed in my previous article "J2ME Benchmarking: A Review" (*JDJ*, Vol. 7, issue 1).

Generally speaking, solutions that rely on the translation of the Java bytecode into one or more native instructions, by either a hardware or software interpretation process, will execute code much more slowly than solutions that are able to execute the bytecode directly. Native Java processors can execute bytecode directly for the vast majority of bytecode. More complex instruction types can be microcoded (i.e., they follow a number of internally coded steps), or else, when this is not practical, a jump to a predefined software routine is invoked (see Figure 1).

Register-rich hardware solutions (e.g., bytecode accelerators or, similarly, those native processors based on RISC cores) will suffer a further performance impact resulting from the need to preserve the state of the registers during the frequent context switches that are a feature of a threaded language like Java.

JVMs are available for most processors and are the most expedient way to enable Java capability on an existing platform. However, this approach is wholly inefficient and not in any way aligned with the J2ME paradigm, as the performance versus resources trade-off in this case is difficult to justify for embedded devices. Bytecode accelerators are specifically designed to operate juxtaposed with a host processor and are relatively easy to integrate. Furthermore, they're able to execute Java bytecode more rapidly than the pure software JVM solutions they replace. However, this is still at the expense of a reduction in the available bandwidth of the host processor for other functions (e.g., communications for an interactive application) as a result of the extra processing burden placed on it.

Native Java processors can execute Java bytecode at optimal speeds and do not place any extra burden on the host processor if deployed as a coprocessor, since they can operate concurrently. Taking everything into consideration, there's a clear migration path (probably time-line dependent) from "easy-to-integrate" JVM solutions through bytecode accelerators to the ultimate performance offered by native Java processors.

How simple is it to integrate a native Java processor with an existing host core? The answer, of course, depends on the design of the processor. The final part of this article explains such a design in more detail.

Finally, though licensing costs are somewhat tangential to this discussion, they're worth a mention since it's an important concern for devices that are produced in high volume. While cost is very much a vendor-specific issue, it's worth pointing out that solutions that utilize both a JVM and hardware intellectual property will incur license fees for both resources.

Integrating a Native Java Processor into a Multiprocessor System

The integration of a Java processor as a loosely coupled coprocessor can be simplified by the addition of a few extra features, including:

- An industry-standard bus interface
- Relocation support for the core memory map
- Host processor communication support

Externally, the Java processor must present an industry-standard bus interface (e.g., AMBA, AHB, MLB) to simplify integration of the processor with the host CPU (see Figure 2). In a coprocessor scenario, both processors are declared bus masters. Since they're able to process data concurrently and are completely independent of each other, conflicts may occur when both processors request bus access simultaneously. Ultimately, in such circumstances, the decision of which processor takes priority lies with the bus arbiter and is defined by the systems integrator at design-time. Code caches are an important feature of any coprocessor implementation. Their importance lies in the fact that not only do they reduce code access times, but they also limit system bus access and so reduce bus contention.

By default, and upon reset, a standalone processor would sensibly execute code from the first location in memory. However, in a multiprocessor system, it must be possible to relocate the program counter to allow the host to redirect the vectors for external instructions to an appropriate location in the physical address map. This could be achieved, for example, by reconfiguration of an index register.

Low-level support must also be provided for interprocessor communications. In the example described here, this is achieved using two mailbox registers: one for communication from the Java processor to the host, the other for communication in the reverse direction. A command packet passed from the sending processor to its mailbox then generates an interrupt to inform the recipient processor that a new value has been written. Subsequently, a further interrupt would be generated to inform the sending processor that the recipient has read the value. It follows that the recipient processor is then able to extract the format of the request by inspecting the mailbox, which could be a method call, data transfer, or reference to a multimedia object. Java coprocessor solutions that are currently market-ready use one of two approaches to implement data transfer. This depends on whether the processor requires dedicated memory resources or is able to support shared access to system memory. Ideally, system memory can double up as a communications area using an independent memory location that's accessible to both processors to transfer data. Otherwise, where the processor does not support shared memory, a FIFO buffer can be used to provide a data transfer path, though this increases the complexity of the design.

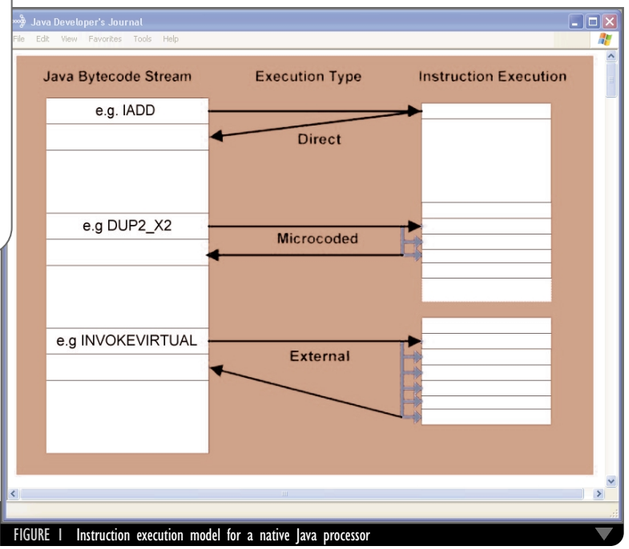


FIGURE 1 Instruction execution model for a native Java processor

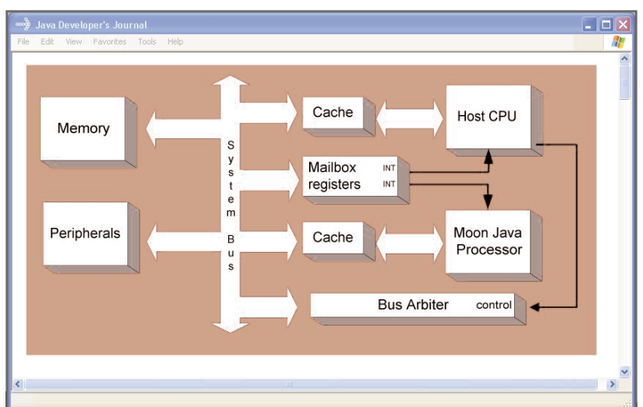


FIGURE 2 Coprocessor deployment

Ultimately, a J2ME application programmer shouldn't need to care about the hardware resources and, indeed, from an abstract point of view, there will be little or no difference between Java code developed for single or multiprocessor solutions. As an example, let's assume that the Java code wishes to make use of a set-top box resource supported by the host processor, such as the tuner. This resource would be accessible only via a Tuning API, such as the one specified in the DVB Multimedia Home Platform standards. In this scenario, a standard Java method could trigger a request (passed via mailbox registers) to the host, passing arguments to indicate which channel is required. Once the operation had been carried out, the host would signal to the Java processor, again via a mailbox register, that the request had been successfully completed (or otherwise), and that the selected channel was available.

Similarly, the process of debugging Java application code is as simple on a multiprocessor platform as it is on a single processor. This can be accomplished using standard protocols, such as the KVM Debug Wire Protocol (KDWP) to interface the Java processor directly to a development and debug environment such as Forte. In this instance, a JTAG port would be used to enable arbitrary locations in memory to be written to (i.e., to send command packets) or read from (i.e., to receive reply packets). Alternatively, debug can be accomplished via the host processor, using mailbox registers to enable communication between the two processors, as described earlier.

Summary

This article discussed issues that pertain to the selection of a Java solution for devices with a significant investment in heritage code. Moreover, following a clear migration path from virtual machines to embedded hardware solutions, the article also discussed the practical implementation of a dedicated hardware coprocessor solution. It's probable that all the solutions described, from the easiest to integrate to those offering the ultimate performance, will be deployed in multilingual, multiprocessor systems long before single-language devices are upon us.

Author Bio

Dr. Carl Barratt works in the applications department of Vulcan Machines Ltd. He has over eight years of experience in various hardware and software design roles. Carl holds a degree in electronic engineering and a doctorate from the University of Nottingham, UK.

carl@vulcanmachines.com

BEA WebLogic Workshop

by BEA Systems, Inc.

REVIEWED BY JOSEPH A. MITCHKO jmitchko@rcn.com

BEA Systems, Inc.
2315 North First Street
San Jose, CA 95131
Phone: 800 817-4232
Web: www.bea.com
E-mail: sales@bea.com

Testing Environment
OS: Windows-XP
Hardware: Dell Inspiron 8000

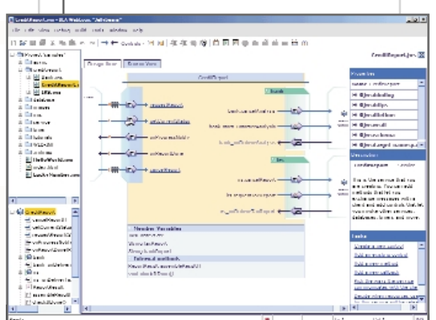


FIGURE 1 WebLogic Workshop visual development environment

To fully appreciate the power behind Workshop, you need to know a bit about Java Web Services (JWS), an up-and-coming standard in the J2EE world. Just as you can embed Java code in a JSP file and have it compile on the application server, Java code in a JWS file is compiled automatically into a Web service.

JWS allows you to take standard method calls in a Java class and, by adding one or more Javadoc-based annotations, instruct the Web application server to expose the method as a SOAP-based Web service. Workshop allows you to map an XML element in the SOAP message to a specific method parameter. This allows the service to maintain its public contract (the underlying SOAP interface) while changing the implementation.

Features

The Design View, an integrated development environment, contains a visual representation of a Web service and a runtime environment where you can code, compile, deploy, and test a Web service under development. The beta version I used came bundled with a prerelease version of WebLogic Application Server 7.0 and WebLogic Builder.

Design View

The Design View is made up of several window panes showing various aspects of the Web service under development (see Figure 1). The left side contains both a project and a structure pane, the right a property panel where you can modify the characteristics of the Web service. A visual representation of the service is provided in the center pane. When you click the source view tab, it switches to an editor containing Java source code.

Building a Web Service

Workshop provides developers with the ability to create and deploy Web services just by creating and configuring objects in a "painter"-like interface. Within the Design View, you can set up one or more public interfaces for the Web service and attach the control interfaces of the EJB components, database objects, etc., to the service. The underlying Java code that you write for the service integrates the various control interfaces into a functioning Web service.

Building and deploying your Web service is extremely easy and seamless. Initiate the build, and you're only a few seconds away from testing the interface in the test harness. If all goes well compiling the JWS file, the process of building and deploying (no syntax errors) works each and every time.

Test Harness

Workshop's browser-based test harness facility contains everything you need to verify and diagnose your Web service. The overview tab in the harness contains various links to the WSDL for the service, client source code, a description of the service, and some other useful links. The console tab takes you to the WebLogic console, where you can monitor the various components that make up your service. The Test form and Test XML tabs provide you with the ability to run the Web service and monitor the request and response messages. The harness automatically comes up when you build your Web service, and is essentially testing a fully deployed Web service.

First Impressions

The GUI design is clean and visually appealing; the service controls and adapters are well laid out and easy to read. The complete development cycle is quick and seamless; you can easily run through a complete test cycle in under a minute.

One particular aspect of the Design View I found unique is the instant code checking feature. If there's a problem in your code, it'll immediately be underlined with a wavy red line. Move the mouse focus on top of the error condition and a description appears. The autofill feature in the editor worked equally well, and listed the various methods available on a particular instantiation of a class.

Web Service Debugging

The IDE provides several of the standard debugging features you would expect in a Java development tool. I ran one of the Web service examples in debug mode and set a breakpoint or two. It worked as you would expect it to when invoking the service through the test harness. This is definitely an added bonus.

Limitations

BEA WebLogic Workshop is not a Java IDE in the traditional sense. It simplifies J2EE development for developers who don't know J2EE APIs. While you don't build EJBs directly in WebLogic Workshop, the runtime creates EJBs to implement the Web service. Also, although the builder can easily set up a JMS queue for an operation, Workshop does not assist in setting up JMS publish and subscribe message interfaces. You'll need to set them up manually or use a utility that comes with the messaging server. Also, the product does not contain an embedded workflow engine, so all workflow activity needs to be managed within the Java code.

Conclusion

Workshop has the potential to be a powerful tool in the development and deployment of large and complex Web services, where you can literally see how it all fits together and works. Combined with the latest version of BEA WebLogic Server, it becomes a very impressive platform.

Exclusive: Excerpts from JavaDevelopersJournal.com

THE INTERNATIONAL JAVA COMMUNITY AGAIN ASKS THE MULTIBILLION DOLLAR QUESTION: 'IS IT TIME SUN RELINQUISHED CONTROL OF JAVA?'

by JDJ News Desk

WHEN MAINSTREAM NEWSPAPERS far from Silicon Valley, such as *The New York Times*, start referring to you as "troubled," even a giant like Sun Microsystems has to pause and take stock.

While on the subject of stock, Sun's has now dropped 90% from a split-adjusted high just 20 months ago of $64.32 (on 9/1/00)…so what's going on in Santa Clara, and what effect might it have on Java?

This latter question is one that ***Java Developer's Journal*** editor-in-chief Alan Williamson was asked on behalf of the worldwide Java community.

"I absolutely do not wish Sun to go down," Williamson told *JDJ* News Desk, as he left the UK for a developer conference in Toronto, "but I do want them to get out of the courtroom and back into the boardroom. Like many other Java developers, I want Sun to start talking strategy as opposed to lawsuits."

"Scott McNealy," Williamson said, "has taken his eye off the ball and for some reason is dabbling in the world of law far more than he should. If he were to channel the energy he is putting into court battles and lawsuits with Microsoft into making Sun back into the mighty corporate giant it once was, then I think Javaland wouldn't even be discussing – as it is at present – the possibility that IBM, for example, might somehow try and buy Java from Sun."

IBM Favors Open Sourcing

Or buy Sun from Sun. This last suggestion was one that *JDJ* News Desk put to IBM's director of e-business standards strategy, Robert S. Sutor, in an exclusive interview.

"Well, that's pretty hypothetical," Sutor replied, laughing, adding: "I'm certainly not someone they would ask. I have no idea. I deal with standards; I'm not anywhere near that."

Of course, IBM wouldn't necessarily have to own Sun Microsystems to have greater influence over Java. The same would apply if Java were open sourced. On this point, Bob Sutor was much more forthcoming.

"IBM is very much in favor of open sourcing these types of technologies," Sutor explains. "It's the basis of Linux; it's the basis of the Apache Web Server, for example. We think it's a model that works. I moderated a panel with my old colleague Simon Phipps [Sun's chief technology evangelist] well over a year ago, where he basically said something along the lines that Sun was moving toward open sourcing Java."

What did Phipps, in turn, say? As reported in an exclusive interview with *Web Services Journal* News Desk (www.sys-con.com/webservices/article.cfm?id=230), he said, "The majority of players in the marketplace are using Java today," and added: "The Java platform is a technology platform that is used throughout the computing industry, with 500 members of the Java Community Process helping to standardize that Java platform."

Hardly, in other words, a platform in "trouble." Unless you think of Microsoft Corp's .NET strategy, which is meant to undermine Sun's dominance of the enterprise-computing space by tempting developers into using C#, not Java, in order to more easily jump aboard the "XML-Web services" bandwagon.

'The Java Platform Has Everything You Need…'

Phipps doesn't see such a jump as being at all necessary. "And as of today," he maintained firmly, "the Java platform has everything you need to manipulate XML and to process Web services. That's why in the Giga survey at the beginning of this year 75% of developers said they would choose J2EE as their platform for Web services."

IBM's Sutor, who is used to sparring with Phipps, nonetheless reminded *JDJ* News Desk that the open-source model is "a model that gets a tremendous amount of input from the community and produces high-quality code." In other words, Sun might want to seriously consider relinquishing control of Java.

"Sun owns Java," Sutor underlined, "they can decide if they want to bring it to an open-standards group or if they want to open source it. [But] open source is a model that works, and I think it would work for Java as well."

Alan Williamson wondered to what extent Sun's current predicament, with its COO Ed Zander leaving on July 1 and several other executive departures all coming at the same time, stems from an ambivalence at the very heart of Sun Microsystems…as to whether Sun is in reality a hardware company or a software company, or both.

"To be honest, it doesn't come as any great surprise that Sun is in trouble," Williamson said, "de facto Sun is a hardware/software company. But some of the diehard engineers would maybe argue that Sun is a hardware company at heart and their sally into the world of software has been mismanaged. I guess history will judge this."

What does Williamson think is triggering Sun's problems? "Well, it would be easy to point the finger at Microsoft," replied Williamson. "Right or wrong, that's the simplest thing to do, and certainly doing so will get the most passionate response from the Java development community. However, strictly speaking, is that the case?

"Well, in a way it is," he continued, referring again to what he sees as the drain of focus and resources that goes with Sun's excursions into the judicial system with MS as its target. "Let's look at the amount of money Sun has spent on lawyers over the last few years," Williamson noted, adding: "I suspect it would keep a lot of Sun marketing folks and Java developers in work for a long time."

The Ultimate Outcome

What if the worst happened? Williamson has no way of second-guessing IBM, which may or may not have designs on the Santa Clara company or its wonder language/platform. But on one point he is clear: "I do not wish Sun to go down."

But what if? Williamson remains defiant that Java is now bigger than Sun and would continue its success story. "If such a thing were to happen, I don't think Java will suffer. There are plenty of people in the wings who can more than cope with taking the Java mantle."

What about you? Do you think Java would be safe in any other hands than Sun's? Is Bob Sutor right that Java should now be open sourced? Is Alan Williamson right that Java will survive and thrive, come what may?

To respond to this article, go to www.sys-con.com/java/article.cfm?id=1443.

Thank You Alan!

I'd like to thank Alan Williamson for remembering our issues and concerns at JavaOne. I just read the April issue (Vol. 7, issue 4) and saw that my questions were posed to the JDK 1.4 product manager. One thing: scary response regarding the double and float performance question, something to the effect, "we don't specifically isolate a test for this." You would think numerical calc performance would be reviewed upon each new release.

prosys@bellatlantic.net

Alan Should Stop Being a Pessimist!

Get on with it, Alan! In March (Vol. 7, issue 3) .NET could kill Java. In May (Vol. 7, issue 5) J2ME may die. I'm getting sick of all this nonsense from the editor-in-chief.

Frankie
toboy@bigfoot.com

Frankie, my editorials are written from a positive not a pessimistic angle.

I sing the praises of Java at every opportunity, but I'm not afraid to learn some of its weaknesses. Java isn't the answer to everything, and if as a community we can learn to accept this, surely this will allow us to devote our energies to pushing Java into the areas in which it can really rock – where it can make a huge difference.

Java is a beautiful language and will be around for a long time. I want to make sure that we build and sing its strengths and accept its weaknesses.

We have a duty to care for our language; to that end, I hear Java success stories all the time and this brings a lot of pleasure to me. But in **JDJ** *we don't want to focus always on the good stuff, we have to shed some light on some of the things that maybe aren't so great.*

If Java is to survive and not become just another legacy language, we mustn't let our confidence be our downfall.

We have to learn from SmallTalk and C++ and not add Java to that list.

Alan Williamson
alan@sys-con.com

Incorrect Impression

In Scot Silverman's review of RequisitePro (Vol. 7, issue 4) the conclusion gives the impression that a third-party database engine is mandatory. This is incorrect; RequisitePro installs out of the box with MS Access DB drivers so it doesn't even require MS Access software to be installed (not mentioned on the Rational site, www.rational.com/products/reqpro/prodinfo.jsp#more).

Tom Servaes
dsdmtom@netscape.com

A Means to an End

Dan Pilone wrote a very good article ("Pervasive Computing," Vol. 7, issue 4), but he said one thing that I'd like to contradict. "Now the question is: How can a developer leverage the available tools to make an application that's useful for the consumer?"

The question should be (always): "What kind of application does the consumer want?" Technology and tools don't matter except as a means to an end: satisfying real requirements. A lot of very good developers found that out to their dismay during the Internet bubble.

CLR vs JVM

CLR, Microsoft's virtual machine, has to be lightweight ["There May Be Trouble Ahead" (Vol. 7, issue 4)]. The JVM has gone through and continues to go through many changes; the one language it supports is Java. Now imagine the CLR supporting multiple languages. How laughable to think that it can be all things to all languages.

Mark Allred
mark.allred@concert.com

JAVANEWS

4 *Macromedia Releases JRun 4*

(San Francisco) – Macromedia, Inc., has announced the immediate availability of JRun 4, the latest release of its Java application server. The new version includes the latest standards compatibility, simplified deployment, innovative clustering technology, and Web services integration. Download from www.macromedia.com/store/.

4 *Borland Expands Development Solution for Java*

(Scotts Valley, CA) – Borland Software Corporation has announced its expanded development solution for Java with the introduction of JBuilder 7, Borland Enterprise Studio 4 for Java, and Optimizeit Suite 4.2.

Borland's new products support the latest standards for Java, Web services, and wireless development.
www.borland.com

4 *Sitraka JProbe 4.0 Now Shipping*

(Toronto) – Sitraka is shipping version 4 of its comprehensive performance tuning toolkit with new investigative features such as heap snapshot differencing, new application server and IDE integration tools, and enhanced platform and environment support.
www.sitraka.com

4 *Compuware Releases OptimalJ 2.1*

(*Farmington Hills, MI*) – Compuware Corporation has announced the commercial availability of OptimalJ 2.1, a Java development environment that's at the forefront of pattern editing functionality.

OptimalJ 2.1 includes three new diagrams at the application model level – the DBMS Relational, the EJB Component, and the Web Component. These diagrams help developers understand the application generated by OptimalJ, enabling them to quickly navigate through its components.
www.compuware.com

4 *PointBase Server 4.3 and Embedded 4.3 Available with JDBC 3.0 Compliancy*

(Mountain View, CA) – PointBase, Inc., has announced the availability of PointBase Server 4.3 and PointBase Embedded 4.3. New features include JDBC 3.0 compliancy, greater transactional performance, enriched functionality, improved database console, and enhanced documentation.
www.pointbase.com

4 *New Release of CocoBase from THOUGHT Inc.*

(San Francisco) – THOUGHT Inc. has announced the release of CocoBase Enterprise O/R, version 4.0, service release 2.0.

This release includes new distributed caching features with examples, a new shared source transparent persistence facade interface with open APIs, updated inheritance and cartesian management, and updates to the Sun ONE Development Environment (Forte for Java) integration including new user documentation.

For a free 30-day copy of CocoBase Enterprise O/R go to www.thoughtinc.com/cber_info.html.

4 *eNGENUITY Technologies Announces JLOOXTelecom 2.0*

(Montreal) – eNGENUITY Technologies Inc. has announced the release of JLOOXTelecom 2.0, a ready-to-use development framework for creating distributed Java-based network/element management systems and network planning applications.

Among the new features is support of several new protocols for client/server communication. JLOOXTelecom 2.0 also uses Lightweight Directory Access Protocol (LDAP) to facilitate user authentication and control access to specialized server- and client-side business logic.
www.engenuitytech.com

4 *The Middleware Company Announces J2EE Patterns Training*

(Austin, TX) – The Middleware Company's new J2EE Patterns training course is available to developers worldwide.

J2EE Patterns is a one-week training course for hard-core J2EE developers who already have experience with J2EE technologies and want to take their knowledge to the next level. Students will learn best practices, design patterns, antipatterns, and idioms. The course is vendor-neutral, and the concepts learned can be applied to any J2EE programming environment with any modern J2EE application server.
www.middleware-company.com

4 *Parasoft Signs Global Software Agreement with IBM*

(Monrovia, CA) – Parasoft has announced an agreement with IBM to provide Java application development testing, support, and service for IBM software products. Under the terms of the licensing agreement, Parasoft will provide global use of its Jtest product as well as in-person and Web-based training for IBM developers, including IBM Global Services (IGS).
www.parasoft.com

4 *QNX Momentics Simplifies Embedded Programming*

(*Ottawa, Canada*) – QNX Software Systems is shipping QNX Momentics development suite, an integrated toolset for embedded development, that's available in two editions: Professional and Standard. The Professional Edition includes C, C++, Embedded C++, and Java code developer tools.
www.qnx.com

SUN UNVEILS SUN ONE PORTAL SERVER 6

(Santa Clara, CA) – Sun Microsystems, Inc., has announced the Sun ONE Portal Server 6, a portal server solution that includes fully integrated, secure identity management capabilities.

The Sun ONE Portal Server 6 (formerly iPlanet Portal Server) also offers Web single sign-on, policy and access control, service provisioning, and unified user management – via the embedded Sun ONE Identity Server – and provides all the development tools and portlet ISV support required to quickly and efficiently deploy employee, customer, and business partner portals.
http://sun.com/software/products/portal_srvr/home_portal6.html

ROCOCO RELEASES BLUETOOTH SIMULATOR WITH J2ME SUPPORT

(*Dublin, Ireland*) – Rococo Software is shipping Impronto Simulator 1.1, a Bluetooth application development tool that runs Java applications in a simulated Bluetooth environment.

Impronto Simulator 1.1 now includes J2ME and J2SE support, security control through the Bluetooth Control Center, and an industry-leading JVM. It's 100% Java and simulates a complete Java APIs for Bluetooth Wireless Technology (JABWT) environment without Bluetooth hardware or stacks.
www.rococosoft.com

'Should I Stay or Should I Go?'

Switching careers midstream

WRITTEN BY
BILL BALOGLU & BILLY PALMIERI

We were recently looking for a skilled Swing engineer for one of our clients. Three people who had been referred to us looked like good candidates for the job.

One of them was happily employed. The other two had left the business – one is now an aspiring rock star, the other, a monk.

In the wake of historically massive layoffs at technology companies, many tech professionals have reconsidered what they want to do with their lives.

For many people, this process of reconsidering career goals has followed a familiar pattern. A common scenario for those who were laid off goes something like this:

1. Within the first few weeks comes the disorientation and shock of losing that daily routine and sense of responsibility and self-esteem that goes along with an important position.
2. Still in fast-forward mode, your frantic networking gene kicks in to "get you back in the game" as soon as possible.
3. Everyone in your network is glad to hear from you and sympathetic, but they've been laid off too – and no one is hiring.
4. You decide to use some of that overtime you earned pulling all those prerelease all-nighters and take some time off to do a little traveling.
5. That trip to Hawaii, South America, Europe, or Australia reminds you that there are places in the world where time moves slower than in the hotbed of high tech.
6. You relax, unwind, and realize that there may be more important things in life than coding, debugging, and beating the competition to market.
7. Two or three months go by. You check back on the job market. Still nada. And then it occurs to you: *Do I really want to go back?*

"A third of the people in my network ultimately saw their layoff as more of a blessing than a nightmare," says Kim Mason, former design director at E*TRADE. "I saw it as a chance to change my whole paradigm."

Mason had led multiple groups responsible for E*TRADE's brand identity and product positioning in the U.S. and eight international markets.

After doing contract work and accepting invitations to lecture and lead short-term workshops, Mason went into secondary schools, mentoring students in media design.

Her recent move to Oakland, California, coincided with Mayor Jerry Brown's plan to revitalize city schools with charter conservatory schools, including the Oakland School for the Arts. Mason will be head of media arts for the new school, which opens in September.

"It's been a 180 for me," says Mason, who is herself a graduate of the Duke Ellington School of Arts in Washington, D.C. "Education is one of those careers – you either choose it or it chooses you."

Many tech professionals have been trying to make the transition from the private to the public sector, applying their technical and management skills toward the goals of nonprofit organizations.

After five years as a manager at one of Silicon Valley's largest educational software companies, Michael Chertok left the corporate rat race to make a difference in the nonprofit world.

He's now managing director of Global Catalyst, a private foundation that initiates and grants funding for projects that bring technology, software, and Web access to underserved communities, including third-world countries.

"I went from a large corporate atmosphere to the smaller, nonprofit environment," says Chertok, "and it works better for me. This is what I always wanted. But today there's a lot more caution on the part of nonprofits in hiring people with private sector experience."

"Nonprofits typically operate with a consensus management style, while private companies use a top-down management style," says Chertok.

"A lot of the people who've left are making life decisions versus career decisions," says Anthony Ha, whose titles have included director of application development and director of Web technology.

Currently working on a long-term contract at Sun Microsystems, Ha sees plenty of opportunities on the horizon for engineers who are open to exploring new technologies.

"Most people I know are still in the industry, but they might be moving into newer technologies like wireless or Web services," says Ha.

"If people really love technology, they'll find a way to stay and do what they love, even if it's for less money. And they'll always be able to find jobs doing it." ●

jdjcolumn@objectfocus.com

Author Bios

Bill Baloglu is a principal at ObjectFocus (www.ObjectFocus.com), a Java staffing firm in Silicon Valley. Bill has extensive OO experience and has held software development and senior technical management positions at several Silicon Valley firms.

Billy Palmieri is a seasoned staffing industry executive and a principal at ObjectFocus. His prior position was at Renaissance Worldwide, where he held several senior management positions in the firm's Silicon Valley operations.

The Beauty of Java

WRITTEN BY
BLAIR WYMAN

This is the thirteenth installment of Cubist Threads, but ironically I'm feeling pretty darned lucky to be writing it. Who would have thought this blatantly self-aggrandizing auto-theoretica would survive a whole year?

Not me, though I must admit that time has really flown by. Every month, Alan's thoughtful e-mail reminder catches me a little off guard.

I've already used up most of my "brushes with greatness" stories, so I find myself at a bit of a loss as to just what to write about this month. For the life of me, I can't seem to get into "flowery verbiage mode" today, so please bear with me as I try to adopt a less formal tone. (After all, we've been through a lot together this past year, dear reader, so perhaps I should start addressing you less as an abstract audience of strangers, and more as the amalgamated "friend" you've become.)

I've been very busy at work lately. Schedules have a way of creeping up on me, and the project I'm currently working on is no exception. Compared to recent projects, the nice thing about this one is that it is absolutely thick with Java code. The not-so-nice thing, as usual, is the looming deadline. There's something about the word *deadline* that really twiddles my bits.

As I've told you before, my links to Java began tenuously. My work has mostly been about writing bits and pieces of the C++ "underpinnings" of our wonderful iSeries JVM, but lately I've been studying (and writing) a lot of Java code – good and bad – and have never been happier. Oh sure, writing C and C++ code is fun and all, but the relative drudgery of worrying about structure alignment and storage management really makes me appreciate the beauty of a garbage-collected language like Java.

Of course, part of me worries that having a garbage collector makes me a "namby-pamby" programmer. It just isn't hard enough to implement convoluted program logic when you know you have a good GC behind you, right? How can I maintain my illusion of guru-ery if I don't have to malloc() and free()? Shouldn't I mask the occasional sign bit, grok the double floating point format, or XOR repeatedly and recursively? "Proper" programming simply can't be simplified too much, or else it turns into some abominable exercise in visual button-pushing, doesn't it? I mean, you can't let just *anybody* be a programmer, *can you?* Egad!

Yeah, right…hogwash. Programming is about taming unruly slivers of etched silicon, using whatever best practices are available (at the current state of the art) to get the job done.

When I started programming, one of the first things I really sank my teeth into was chaos and fractal visualizations. It seemed like my (4.77MHz 8088-based) hardware was never even close to fast enough – especially for Mandelbrot set renderings – so I remember taking great pains (can you say ASM?) to implement an M-set iterator directly in the 8087 math coprocessor chip's fabulously expansive 8-register onboard stack.

The interface between Turbo Pascal 3.x and ASM was as clean as could be hoped for: the compiler pushed floating point arguments to my little assembler routine (creatively named "iter") directly into the 8087 stack. All my ASM code had to do was merrily FDUP, FMUL, and FCMP to my heart's nascent content. My "improved" iterator ran in just one-third the time taken by the high-level language algorithm (even fully optimized), and I was in programming Nirvana.

Of course, the real world reasserted its cantankerous nature when the next release of the compiler up and changed the ASM linkage. Suddenly, parameters were no longer getting pushed directly into the 8087 registers, so I had to rework the linkage, losing a good bit of my hard-won performance improvement.

One true beauty of programming in Java, in my not-so-humble opinion, is that we get to think about bigger pictures, without getting caught up in relative trivialities. Instead of ensuring correct argument alignment, chasing pointer bugs, and forgetting to delete our temporary structures, we can spend our time changing the world.

Okay, I've obviously swung my paradigm pendulum too far. I have a ton of respect and admiration for the bit-twiddlers and storage managers in the programming world. Frankly, it is on their vast shoulders that we stand, boldly new-ing where no programmer has new-ed before, and we must never forget that. Java's decidedly credible beauty of form is the result of incredible intellectual effort and the culmination-du-jour of programming practice.

What's next? "Computer, please cue up some Blue Oyster Cult and brew me a stout ale, will ya?" Cool.

blair@blairwyman.com

AUTHOR BIO

Blair Wyman is a software engineer working for IBM in Rochester, Minnesota, home of the IBM iSeries.

Java Dudes