Does Java Really Matter?

November 4, 1998

Does Java really matter? If you've been asking yourself this question, welcome to the club. Propelled by incessant streams of hype, the technology certainly has attracted far more attention than its few readily apparent successes would seem to warrant. Yet our look behind the hype clearly shows Java momentum is on the upswing--and in places that might not be apparent. In our review of Java from four different product angles--applications, environments, applet tools, and development packages--we find the answer to the question we pose above is: Yes.

To be sure, the list of hyped expectations is not short. Java seems unlikely to displace Microsoft Windows as the dominant desktop operating system, as some had predicted it might. And the technology has not taken over the world of embedded devices such as cellular phones. NCs--the much-vaunted network computers that are supposed to simplify PC operation and management while lowering costs--are almost nowhere to be found.

What about Java's boldest, most fundamental claim for programmers, that of "write once, run anywhere"? This phrase, so central to the Java message that Sun has trademarked it, embodies a beautifully simple idea: Programmers could develop software once that would run on any machine and on any operating system--without modification or additional testing. But as any experienced Java programmer will tell you, "write once, run anywhere" is just not real in today's world, with

its volatile virtual machines and dueling browsers, none of which implement Java in exactly the same fashion.

Even some of Java's earliest supporters, Web publishers eager to add interactivity to their sites, have backtracked. Their frustrations are twofold: Java programming can be resource-intensive (at least relative to HTML development), and the slow speed of most users' current Internet connections makes the download time of anything but a trivial Java applet (such as a ticker) too much to endure.

Okay, so why do we say that, yes, Java really matters? Because in spite of all that, and nearly three years after being thrust into the spotlight, Java is finally emerging in a tangible way. But rather than as a revolutionary technology, Java is emerging as a promising evolutionary one that will subtly but significantly change the course of computing. It's time to sort through the hype and make an informed decision about where Java fits in your organization.

In this four-part cover story, we evaluate the promises and the products. Java *is* important, but like any new technology, it is not a "silver bullet" that's going to solve every challenge your organization faces.

What Is Java, Really?

To understand what Java can really do, it's important first to grasp what Java really is. On one level, Java is a programming language. But just as Windows is more than just a graphical user interface, Java is more than just another way to write code. Still, it makes sense to start there.

The Java language is superficially similar to the wildly popular C++, so C++ programmers generally find it easy to learn Java's syntax. And as our story "Java Development Packages" reveals, Java development packages mirror the RAD approach popularized with C++. But unlike C++, Java is object-oriented from the ground up. This helps programmers enforce good object-oriented practices, which leads to more maintainable code.

As with C++ and C, much of what is usually lumped together as Java-the-programming-language consists of

runtime libraries. These libraries provide a standard set of facilities for manipulating the user interface, communicating across a network, and so forth. Consistency in the runtime libraries is important for achieving the "write once, run anywhere" claim and is in fact the focus of one Sun lawsuit against Microsoft. (The lawsuit alleges that Microsoft fails to provide two Java runtime libraries--RMI, for*remote method invocation,* and JNI, for *Java native interface*--and that Microsoft has engineered subtle but impermissible changes into the behavior of core Java libraries.)

Indeed, in late 1996, Sun launched its 100% Pure Java initiative, which examines each "Java product" against a tightly guarded certification process. Products that pass are labeled as 100% Pure Java and carry the guarantee that they use only standard Java features and should, theoretically, run on any properly implemented version of Java.

Sun currently lists more than 120 such products at its 100% Pure Web site. But there's one hitch: No widely used Java VM meets Sun's compatibility requirements. Netscape pulled the Java logo from its browser in late 1997, because its implementation was not fully compliant; Sun and Microsoft remain embroiled in legal battles over Microsoft Internet Explorer.

Not Just a Language

If Java were just another programming language, the industry would have greeted its introduction with a resounding yawn. What makes Java intriguing is that it is also a runtime environment embodied in what is called a *virtual machine* (VM).

This VM sits, in essence, between the Java program and the machine it is running on, offering the program an "abstract computer" that executes the Java code and guarantees certain behaviors regardless of the underlying hardware or software platform. Java compilers thus turn Java programs not into assembly language for a particular machine but into a platform-neutral "byte code" that the machine-specific VM interprets on the fly.

The Java VM also enforces security policies, providing a sandbox that limits what the Java program can do. A Java applet cannot, for example, peek into arbitrary files on the

machine it's running on. The most recent version of Java from Sun, known as Java Development Kit (JDK) 1.1, though, provides no consistent method for an applet to request restricted system resources. This capability will be available in JDK 1.2, due out later this year.

The lack of a consistent mechanism for accessing local files is one major impediment to Java's success as a replacement for traditional productivity applications on the desktop.

The Java VM is not the industry's first attempt to develop a platform-independent computing environment. But Java has gained widespread attention where other attempts have failed, in large part because the VM was distributed with browsers beginning with Navigator 2.0, thus giving Java unprecedented initial market penetration.

As our story "Java Environments" shows, however, the VMs on different platforms today simply don't behave similarly. This makes debugging a serious problem for Java deployment in a heterogeneous environment.

The compatibility situation is likely to get worse before it gets better: JDK 1.1 is a substantial change from JDK 1.0, which means that compatibility between the two is not assured. And the delivery of JDK 1.1 support in VMs has been surprisingly slow: JDK 1.1 is only partly implemented in Netscape Communicator and is largely implemented in Internet Explorer 4.0.

Moreover, even if all the VMs operated properly, "write once, run anywhere" is not necessarily a panacea. In fact, it places significant hurdles in the path of developers. For one thing, the Java VM generally presents applications using a given machine's native user-interfacewidgets (components such as command buttons and drop-down boxes). Thus, Java programs act in a fashion familiar to users on a given platform. But this also destroys user-interface consistency from platform to platform, which makes testing a serious problem. (The forthcoming JDK 1.2 contains support for "pluggable" user interfaces intended to alleviate this problem.)

In addition, "write once, run anywhere" implies a lowest common-denominator approach. To offer one often-cited example, not all computers (specifically Macs) offer a right

mouse button. Should programmers have to forgo offering context-sensitive pop-up menus for every platform as a result?

Compared with native code, Java VMs are excruciatingly slow. The advent of just-in-time (JIT) compilers in Version 3.0 browsers, at least on the PC platform, helped alleviate this shortcoming. But for many applications, Java still cannot compete with natively compiled C++ code.

A key addition to JDK 1.1 is support for JavaBeans, a component model for Java. JavaBeans make it easy to write reusable components that can be strung together with a minimum of additional coding. Although Microsoft's ActiveX offers similar advantages, Beans are less focused on a Windows-centric world and are somewhat more portable. Still, ActiveX does have the advantage of supporting any underlying programming language, while Beans are designed solely to encapsulate Java.

Java as a New Platform

Java and the Java VM together provide a set of services that Java programs can rely on, regardless of the underlying hardware and OS. In this respect, Java has become positioned as a new platform--a target environment for applications.

Why does the world need a new platform when Windows is clearly the PC platform of choice? For one thing, although Windows is dominant, it's hardly universal. The significant potential benefit is the ability to write code that will work on a variety of systems--especially in an extranet, for example.

In addition, any new platform has the advantage of starting with a clean slate technologically. The Windows/Intel PC architecture, ubiquitous as it may be, is hobbled by its heritage, which stretches back to the 8088 processor and DOS 1.0, introduced with the original IBM PC in 1981.

Of course, this hobbled heritage is also Windows' biggest strength when it comes to software compatibility. This is also why Java will never replace Windows as a general-purpose desktop OS: There's simply too much existing investment and inertia in the Windows platform and too little to be gained by

rewriting all that software.

Indeed, as our story on Java applications illustrates, today's Java applications generally don't provide compelling alternatives to existing, highly refined Windows software. For example, Corel's early attempt to take its Windows applications and simply make them Java applets (Corel Office for Java) didn't work, because Windows applications don't just simply translate into some lighter versions of themselves. Java developers are finding that Java applications work better when they are put together as bits and pieces, as in Lotus's eSuite.

Where JAVA Makes Sense

In the business world, Java makes some sense for deployment in extranets. Java's cross-platform promise means that theoretically you can create applications to use with your customers and suppliers and not have to worry about not having control over the environments they're running. The fact that Java applets are downloaded on the fly eliminates the explicit software installation process and extends their reach. But hurdles are high here, both because HTML can offer an even better cross-platform story than Java (though it's not interactive in the same way) and because cross-platform testing and debugging remain so difficult.

Another compelling use for Java may be in the middle tier of transaction-processing systems--not the servers or the clients but the "business logic" in the middle that actually describes the processes of filling an order or tracking a request. Java's platform neutrality means, theoretically, that you can write code once and have it run on any Java-enabled system. Although you may not need to run the applets simultaneously on different platforms, having the flexibility to move from, say, a Windows NT to a Unix server without rewriting code could be beneficial.

It also would be unwise to ignore Java's prowess as simply a good back-end development environment. JavaBeans offer a promising mechanism for building general-purpose business components that you can reuse with minimal modifications. And Java's simplicity (relative to C++) may improve programmers' productivity. In any case, whatever Java's pros and cons, there's no

disputing that many programmers simply enjoy using it.

Another place Java may make sense is in highly vertical, single-purpose applications that are essentially custom-developed front ends to transaction-processing systems, often as replacements for traditional "dumb" terminal applications. This is the one area where NCs stand a real chance of succeeding in a big way. IBM, in fact, has made some sales into precisely this sort of market.

In these environments, a general-purpose PC may be unnecessary or even counterproductive. And it's worth noting here that the idea of NCs for the home is a nonstarter: Low PC prices negate any cost advantage, and network connections simply aren't fast enough to rely on remote storage.

What about embedded devices? Java has some high hurdles to clear here, as embedded devices are often limited in purpose and constrained in power and size. That tends to argue for dedicated hardware, especially given that platform-independence in this area is less crucial. But TCI has signed a deal to use Java in some of its television set-top boxes, and Nokia has signed a similar deal to use Java in cellular phones, so we may yet see progress in this area.

What It All Means

At this point, it's clear that Java technology holds promise—but also that Java has been overhyped and needs a reality check. Sun could help assure Java's success by focusing less on its potential to dethrone Microsoft and more on its use to solve real-world business problems.

Furthermore, Sun needs to make the Java Compatibility Kit public and allow third-party verification of compatibility claims. Such a public test would help catalyze more compliant VMs. And Sun needs to reassure developers investing in Java that future releases of the JDK will maintain backward compatibility with existing code. Ignoring backward compatibility has been a deal killer in the past. (According to Sun documentation, there's no guarantee that a JDK 2.0, for example, would support JDK 1.*x* programs.)

Microsoft?

If you're in position to decide what to do about

Java now, our advice is definitely to begin experimenting with it on a limited basis, particularly on the server side. You should document idiosyncrasies and develop a "safe list" of techniques that work reliably on *all* the platforms you need to support. Adhere to rigid object-oriented programming practices and design your objects' interfaces carefully, with a clear focus on reusability. Then if (or when) Java turns out to be the right solution for larger projects, you will have a solid foundation to build on.

In the pages that follow you'll find comprehensive evaluations of Java products to help you determine exactly where you should dive in and ride the Java wave.

APPLICATIONS

After a slow start, Java is finding its way into off-the-shelf applications. SunSoft lists more than 120 applications at its site as being "100% Pure Java."

Research asked...What types of applications (will) use Java? (Source: 50 Fortune 1000 companies interviewed by Forrester Research.)

PC Labs' Test Results

Our survey of Java applications demonstrates that Java is at last making some progress on the desktop. We discovered a wide range of applications, including productivity suites, financial tools, and e-mail clients.

A Brief History Of Java

From household appliance networking tool to Internet darling and back, it's been a helluva seven years for Java.

January 1991

Java starts as "Oak," headed by James Gosling, intended to network household appliances. March-August 1993

Sun refocuses Oak for a Time Warner interactive TV project. SGI wins the deal instead. Another deal falls through in August.

June 1994

FirstPerson (part of Sun) begins "Liveoak,"

again refocusing the technology for small OSs. A month later, the team retargets Oak's potential for the Internet and points toward Java's current function.

May 23, 1995

Java launches at SunWorld 1995. The first company to endorse Java: Netscape. Sun makes available an alpha version of its Hot Java browser.

September 18, 1995

Public beta of Navigator 2.0, with support for Java applets, giving Java a major platform and presence on the Web.

October 1995

EarthWeb launches Gamelan, a Java resource directory and news site for developers.

October 24, 1995

Java discussed in *PC Magazine* for the first time. "InternetWorking" column says Java is crucial for distributed computing, instant software upgrades, and making Web pages interactive, more seamless than browser add-ons.

December 4, 1995

IBM licenses Java, making a commitment to Java that will extend to hardware and software.

December 7, 1995

Microsoft announces it will license Java. Microsoft's intent to optimize Java for Windows sets the stage for a long simmering feud that boils over two years later.

December 13, 1995

Symantec licenses Java and releases the first Java development environment for Windows 95 and NT, Symantec Café, an immediate success.

December 19, 1995

Java wins *PC Magazine*'s Technical Excellence Award. Reason: It gives developers freedom, a streamlined language, and versatility.

January 9, 1996

Sun forms JavaSoft to develop Java-based products and work with third-party developers.

January 23, 1996

Sun ships the Java 1.0 programming environment (JDK 1.0). Includes Java Applet Viewer, Java Compiler, and Java Virtual Machine for the first time.

April 9, 1996

PC Magazine reviews JDK 1.0: It shows promise because of its cross-platform capability, but as a language and development environment, it is "as cafeteria coffee is to Italian espresso."

May 29, 1996

More than 6,000 people attend the first JavaOne developers' conference. Corel announces its intent to develop the Java office suite, with Java versions of WordPerfect and Quattro Pro. The plan ultimately fails.

June 11, 1996

PC Magazine takes its first significant measure of Java and Java Tools in PC Tech section. Conclusion: Developers are right to be excited about the potential, but the language and tools are still very primitive.

October 22, 1996

PC Magazine speed trials on Java environments: Microsoft Internet Explorer 3.0 has the best Java performance by far. Navigator 3.0 is significantly faster than 2.0 because of the new just-in-time compiler.

October 29, 1996

The Java Enterprise Computing launch includes the announcement of JavaStation network computers. Sun extensively demos the Java station but doesn't ship it.

December 11, 1996

At Internet World, Sun announces the 100% Pure Java initiative, endorsed by more than 100 companies, with one notable absence: Microsoft.

February 18, 1997

JDK 1.1 released. A dramatically improved language, including JavaBeans, print, and database access.

March 17, 1997

Sun applies to the ISO/IEC Joint Technical Committee 1 to become a recognized Publically Available Specification (PAS) submitter for Java.

April 2, 1997

The second JavaOne has more than 10,000 attendees. Sun releases JavaStudio, a visual component builder for JavaBeans. Announces Enterprise JavaBeans--further potential for Java in business--and PersonalJava, a version for consumer appliances, bringing Java back to its roots.

May 27, 1997

PC Magazine's first cover story on Java: Java environments and first-generation applications. Conclusion: The excitement of using Java office applications can not overcome limited functionality and notorious instability.

August 27, 1997

Sun CEO Scott McNealy interview in *The Wall Street Journal:* "By any comparison--fire, electricity, the wheel--the rollout [of Java] has happened at an unparalleled speed."

October 7, 1997

Sun announces lawsuit against Microsoft for breach of its Java license, citing the failure of Microsoft products such as IE 4.0 to pass Java compatibility tests.

November 3, 1997

Lotus announces eSuite WorkPlace (begun in January as Kona), a set of Java-based, platform-independent desktop productivity applets. The applets are designed for IBM's NC--the Network Station 1000.

December 16, 1997

Java is the big winner in *PC Magazine*'s Technical Excellence Awards: JavaBeans and Visual Café for Java 2.0. James Gosling and

the Java Team win the Persons of the Year award.

January 1998

Tele-Communications Inc. (TCI), the nation's largest cable provider, selects PersonalJava as one operating system for its digital set-top boxes, bringing Java full circle to interactive TV. But TCI also licenses Windows CE for the same purpose. Stay tuned.

Environments

Java environments, also called virtual machines, are available for most operating systems. But Java applications do not behave the same across all OS platforms.

Research asked...What are the benefits of Java? (Source: Forrester Research.)

PC Labs' Test Results

In our testing, Java Windows environments were by far the fastest performers and delivered the best overall compatibility. Still, no Java environment could run every application we tried, and no application ran across every environment on every OS.

Java application compatibility by operating system

Applet Tools

Java still has much to prove on the desktop, but anyone can easily create Java applets that add interactivity to Web pages.

Research says...As of February 1998: 9,045 Java-enabled Web sites.

PC Labs' Test Results

As Java authoring tools mature, they become specialized. On our tests, BeanMachine ranked *excellent* on ease of use; it is suitable for creating a range of applets. Jamba excelled at creating multimedia applets and is geared toward professional Web designers.

The sites include these Java recources:

244 business and finance, 329 arts and entertainment, and 1,515 games. A resource is

generally defined as a place in a Web site containing Java applets, examples, or information. (Source: Gamelan: The Java Directory.)

Development packages

Nearly all Java development packages offer the preferred RAD approach. Several provide tools for server-side Java development.

Research says...Some 500,000 programmers are using purchased Java programming tools on a regular basis--roughly double the 1996 figure. 125,000—or 25% of them--call themselves "hard-core." (Source: IDC Research.)

PC Labs' Test Results

In evaluating the products, we looked for the best combination of power and ease of use. We found that each package in our roundup delivers sufficient support for most development tasks, but they vary in their flexibility, extensibility, and number of visual creation tools. Here's how they line up.

- Sybase PowerJ Enterprise Version 2.1
- JBuilder Client/Server Suite
- Visual Café for Java 2.1
- VisualAge for Java
- Sun Java WorkShop 2.0
- CodeWarrior Professional

— *John Clyman*

Find more reviews: -- Choose a product category: GO!