Write Once, Run Anywhere: Why It Matters*

Matt Curtin
interhack.net
cmcurtin@interhack.net

April 9, 1998

Abstract

There has been a lot of talk, and a lot written about Java. Sun's assertions that Microsoft is attempting to sabotage Java's promise of "Write Once, Run Anywhere" began a flurry of discussion about this promise, Microsoft's business practices, and what Sun really wants out of its Java technology.

But what does all of this mean? Who cares? Why does it even matter?

This document is an attempt to find answers to these questions. It's intended primarily for information technology managers and end users. However, it might also be of interest to developers.

1 What "Write Once, Run Anywhere" Means

From the beginning, Sun intended Java to be a *platform-neutral* language. That is, Java programs would be written to run on a *Virtual Machine* instead of a physical computer. The virtual machine, then, would run on a real computer. Because Java's Virtual Machine has to strictly adhere to interface and behavior definitions, a programmer can develop a program for the Java Virtual Machine on the sort of computer that he likes, and expect it to run on the sort of computers that his customers like, inside of their Java Virtual Machines.

This means that when you go to the store to buy an application, you don't buy the "Mac version" or the "Windows version." You buy the "Java version." And as long as you have the Java Virtual Machine—which is free, and available from a large number of vendors—you can buy the program without having to worry whether it's going to run on your computer.

2 "Different Computers Use Different Software"

When you buy a car, you select it based on a number of things, like how much it costs, how cool it is, and how practical it is. Car dealers need to compete on things such as the value of the car itself and the after-sale support.

When you buy a computer, there is some question of how much it costs, how cool it is, and how practical it is. Inevitably, into the question of "practicality" comes the issue of *applications*. How many people run Microsoft Windows instead of OS/2 or a Macintosh because "that's where the applications are" or "that's what we use at work"?

What makes your car go? Gasoline. What makes your computer go? Software.

So tell me why you can pull into any gas station, pump the stuff in, and go on your merry way without worrying about its "compatibility" with your car. And then explain to me why it's different for software. There's no such thing as "Ford Gasoline" or "GM Gasoline." Why should there be such a thing as "Windows software" or "Mac software"?

There shouldn't, and that's where "Write Once, Run Anywhere" comes in.

*This article also appeared as a feature on the JavaSoft web site in January 1997. The latest version is available on the web at `http://www.interhack.net/people/cmcurtin/rants/write-once-run-anywhere/`

3 Badness In the Computer Industry

Many users have been conditioned to believe some ridiculous things about computers. It never ceases to amaze me what sort of things users will tolerate in their computers, simply because they've been conditioned to believe that the behavior is somehow acceptable. (If you think I'm making this up, ask yourself why we hear phrases like "To err is human, but to to really screw up, you need a computer"?) How many Windows users remember rebooting their machines three, four, or 10 times a day? How many still do it? Why? Would we think it acceptable if our cars stalled three, four, or 10 times a day? Would we happily "reboot" our cars by restarting them and continuing on our way until we finally make it to our destinations? So, why is it any different for computers?

The answer, of course, is simple: *It isn't any different for computers.* If your computer crashes, something is wrong. If an application "hangs" or dies unexpectedly, something is wrong. If this happens routinely, it needs to be fixed.

Users need to understand that it's *okay* to demand stuff that works. You are, after all, paying for it. Users shouldn't have to accept brokenness in their computing environments any more than they should in anything else.

4 But Aren't There Problems With Java?

Of course.

Java is still a new technology. As such, not everything works the way its designers, developers, and users would like. Further, it's limited in the number of things that can be implemented, simply because it hasn't been around long enough for the people who are working on it to add everything that they want to see in it.

However, for such a young language, Java has generated an unprecedented amount of interest, and is rapidly progressing in its maturity. Problems that do exist are being quickly identified and fixed. As more organizations direct their resources toward Java, the number of problems will continue to decrease, even as functionality, stability, and performance continue to increase.

5 "Write Once, Run Anywhere" Is a Promise

What's important here is an understanding that "Write Once, Run Anywhere" is a promise that Java will work the way computers should work. Specifically, that they should be able to work together to do as their users want, regardless of who makes them or the software they run. Sun, IBM, Novell, Netscape, and hundreds of other companies—thousands and thousands of developers—are working hard to deliver on that promise.

Much of the promise of Java has been delivered. But it's going to take a lot of work by a lot of people to take Java from being a relatively immature—though useful—tool for a variety of tasks to a mature, robust system which can be applied to almost any computing problem.

A large degree of the interest in Java is because Sun has presented it as a "Write Once, Run Anywhere" technology. Sun has an obligation to the organizations and individuals supporting Java to keep it unified, and to keep the "Write Once, Run Anywhere" promise.

Microsoft will have none of this. Microsoft realizes that a ubiquitous Java Virtual Machine is a direct threat to the virtual monopoly that they've been able to build primarily through savvy marketing and questionable business tactics. A recent example of a questionable business tactic is the surreptitious behavior and interface modification of some of Java's core classes in their own implementation of Java. Programmers who do not recognize these undocumented changes can build their applications expecting them to run anywhere that Java can be found, only to discover that their code works only on Microsoft's own Virtual Machine, which is only available on Microsoft's own operating systems.

Sun Microsystems, so far, has been doing an admirable job of keeping its "Write Once, Run Anywhere" promise. This means refusing to tolerate "alternate" implementations. Java must be united if it is to "Run Anywhere". If Microsoft wishes to propose new functionality to Java, it's perfectly free to do so, just as countless other companies have done. However, it must *cooperate* such that the goals of Java—including "Write Once, Run Anywhere"—are brought *closer* to achievement, not further away.

6 Why It Matters to Information Systems Managers

IT managers, particularly those in large, diverse environments, have a very difficult job. Weighing the costs and benefits of one platform over another, trying to gaze into the future of technology and make purchasing decisions that don't waste the organization's resources, and trying to determine how one system vs. another will impact the business are all significant problems.

Many organizations have been moving to progressively smaller and smaller machines. Finally, with the popularity of the PC, the cost of hardware and software became low enough that everyone could participate. What the industry failed to recognize, though, is the total cost of ownership: how much will it cost to support, maintain, and upgrade the machine regularly, in addition to its base cost? PCs cost significantly more to support than other sorts of computers, including Unix-based desktops, minicomputers, and mainframes. Instead of having one system administrator for every 50–150 users, PC environments have one administrator for every 30–50 PCs. Trying to keep everyone's desktops current with the software they need is a major headache. Now all of the PC vendors are talking about total cost of ownership. Now it's so important that a three letter acronym (TLA) has even been assigned: TCO.

In organizations with diverse needs, things become even more complex. Instead of buying x copies of product *Foo*, the IT manager needs to buy x copies of *Foo for Windows*, y copies of *Foo for MacOS*, and z copies of *Foo for Solaris*. Typically, the sort of software available on platforms like the AS/400 and mainframes is entirely different from that available on "desktop-oriented" machines, and even more insanely expensive.

The cost of developing software across platforms, or for a platform with a relatively small number of installations can quickly get very high. Software developers are in the business to make money. A large reason why PC software costs so much less per unit than mainframe software is because the *volume* allows a vendor to still make a profit by charging less. The cost of software, then, is a burden ultimately carried by the *customers* of the software.

This is the important part of the Java Virtual Machine. A vendor of a computer or an operating system simply needs to provide support for the Java Virtual Machine, and all of the applications written for Java simply work. Vendors of application software no longer need to manage multiple platforms. Newly released software suddenly becomes available not to just your Mac users, or just your PC users, or just your Unix users. The software is available to all of your users. And because the cost of developing and maintaining the software is lower, the prices stay constant, and low. Vendors are now addressing a larger audience than they ever have, and can rely on volume to make money, instead of huge margins. Further, because vendors no longer need to concern themselves with porting their software from platform to platform, there is more time for increasing the *quality* of the software.

Another important thing has taken place. IT managers can put computers on the desks of people in a way that makes sense. No longer will a secretary require a desktop supercomputer just to run a bloated operating system that is also running a web server, providing support for databases, and doing 400 things besides writing that memo. People who need a computer that will just let them write memos and use email will be able to do so. People who need a computer to build applications will be able to get one that makes sense for their needs, whether it's a mainframe, workstation, PC, or network computer. But they all work together, and have the same platform independent, modular programs available.

7 Why It Matters to End Users

Putting the right sort of computer on the desk of users isn't significant only to IT managers, but also to end users.

How much time is wasted by end users every day trying to fix their registry settings, reinstall software, and other ridiculous tasks? While someone might be able to call a help desk or administrator at work, at home, there's almost no way around either doing it yourself or paying someone else to do it for you.

Why does a user who wants to perform the simple operations of writing memos, pulling tidbits off of the web, and using email need to spend time managing such an incredibly complex system? The beauty of Java is that someone who *needs* such a machine can still have it, without forcing that complexity upon those who don't need it and can't manage it.

People who are buying computers for home use can buy whatever they want, without having to worry about whether it will be compatible with what they're using at work, or with what the kids are using at school.

"Write Once, Run Anywhere" means freedom to choose whatever you want, based on criteria that makes sense. "Where the applications are" won't be an issue, because the applications will be everywhere that Java can be found.

Returning to the car analogy, you can buy a Porsche if you like, or a station wagon, or a pickup truck. You make your choice based on what you're going to do with it, not what kind of fuel it takes.

8 But Java Won't Be the *Only* Language

That's correct. Other languages certainly will exist and be used.

And that's why it's important to understand the difference between the *Java Virtual Machine* and the *Java Programming Language*. The important part about the Java Virtual Machine is that it's everywhere. The Java Virtual Machine is what means freedom to choose.

As it happens, Java is the first programming language that can generate the *bytecode* that the Java Virtual Machine understands. But there isn't any reason why this has to always be the case. Builders of tools for programmers are free to create their own tools that will generate bytecode for the Java Virtual Machine.

In fact, many of these exist today for languages including COBOL, Lisp, BASIC, Prolog, Eiffel, and Rexx. Work is being done to bring the same functionality to other languages like C++ and Perl.

By and large, however, the vast majority of programs written for the Java Virtual Machine will probably be written in Java. That's fine, though, because Java is a very useful language. It's a good language for the development and maintenance of programs, even very large and complex ones.

Java doesn't have many things that are entirely new in it. Java is the result of lots of good ideas from different programming languages coming together, in such a way that programmers can do what they need to do without needing to jump through hoops and circles.

9 So What Does It All Mean?

"Write Once, Run Anywhere" is an important, liberating concept. Though Java isn't the first language to offer runtime platform independence, it is significant that Java has gotten the support of vendors of computing devices from supercomputers to embedded systems and smartcards. Never before has such a diverse set of companies worked so hard together to make their systems be able to run the same code. By and large, developers want to make "Write Once, Run Anywhere" a reality. This will be a good thing for computing, the people who use the technology, and even those who use the services of organizations that use the technology.

Sun, in promoting Java's "Write Once, Run Anywhere" philosophy, have amassed a tremendous amount of support. Without question, the level of support Java enjoys is unprecedented in this industry. Because of this support, organizations have poured resources into Java, resulting in a rapidly maturing environment. (I recently read that IBM has 2,500 developers working on Java, compared to Sun's 1,500.)

Does this mean that Sun alone should forever hold the Java specification? No! Java, if it is to be truly open, must be controlled by a standards body or consortium. Sun has shown itself to be deeply committed to delivering on the promise of Java. As a PAS submitter of the Java standard to the ISO, there are certain procedures which Sun must follow for their specification to be accepted by the ISO. To hand the control of the Java specification over to a standards committee now would be foolish. We have the same benefit—approval as a published ISO standard—without the same delays that a committee would inevitably bring about. (C++ didn't get through its standardization process until 1997. We don't have the luxury of waiting that long if Java is to fulfill its promise.) Having Java defined by a standards committee would only serve to provide a means for Java's opponents to break it and make it fail.

A sophisticated language like Java, after having time to mature, running in a secure and scalable environment like the Java Virtual Machine holds the potential to usher us into a new age of software. An age of

freedom, where users can use whatever computers they want, without fear of whether applications will run. An age where information technology managers can intelligently provide for the needs of their users.

Allowing anyone to break "Write Once, Run Anywhere" would be a tragedy. Sun has an obligation to make sure that the investment in Java of organizations and individuals around the world is protected. Sun has an obligation to work to deliver on the promise it has made to us all.

It is important that users, technology managers, and others understand the potential of Java. It's important that these understand *why* Java holds the potential it does.

Recognize the potential of Java, support Java's development, and refuse to use software from vendors who claim Java compatibility without supporting "Write Once, Run Anywhere." Hold Sun to its word. Hold Sun's partners to their words. Do not purchase or support nonstandard implementations.

"Write Once, Run Anywhere" is achievable. We must have it. Do not accept anything less.