Java Not to Everyone's Taste

Much of Industry's Interest Is Result of Marketing Machinery Run Amok

I yield to no one in my appreciation of advances in computer technology.

Java is not one of them.

Java has worked up the development community until they're all but incontinent over the perceived business opportunities this new language will provide. Java will topple empires, level playing fields, empower masses, drain swamps, and right ancient wrongs. It seems there are no miracles this programming language-cum-religion can't perform.

Java may be the grandest hoax perpetrated on the computer industry in a decade. Yet with all the dark inevitability of Greek tragedy, one company after another announces its plan for Java devices, or operating systems, or even silicon.

Don't get me wrong—as a programming language, Java is a fine and useful tool, a welcome change from C++. Its enforced discipline makes writing code more oops-proof than usual. Java applications are constrained to a playpen beyond which they can't do much harm, so code is relatively safe to run. Like any high-level language, Java can be compiled into the binary instruction set of just about any microprocessor. This is a good thing. Instead, conversion to bytecode is the rule.

Simply put, Java sacrifices performance at the altar of portability. Zealots chant the "write once, execute anywhere" mantra, stating the characteristic that supposedly makes Java perfect for networked systems. To solve the problems inherent in distributing source code, Java programs are tokenized (into "bytecode," for lack of a better term), a technique many will recall fondly from pubescent days spent writing GW-BASIC programs.

The beauty of Java is that it runs equally poorly on all microprocessors. Executing bytecode involves emulating not just an alien instruction set but an entire imaginary computer, the Java Virtual Machine (JVM). This requires either an interpreter or a JIT (just-in-time compiler), both of which are huge applications. (Even Java chips need an emulation library.) Quite apart from the memory footprint of the JIT itself, bytecode swells 2–5× in size during the conversion, according to those who have witnessed the spectacle.

Java bytecode cannot conceivably run as fast as code for a chip's native instruction set. Modern CPU architectures are designed specifically for efficient instruction execution (a legacy of RISC). With few constraints other than the pursuit of performance, these CPUs all adopt similar architectural features and basic instruction types. The fact that none of them resembles Java bytecode should surprise no one.

Sun's marketing staff is to be congratulated for hitching Java's PR wagon to that of the Internet. From the ashes of Oak, Java was recast as Internet technology, and its star has risen as rapidly as interest in the Internet itself.

Java is touted as a panacea for business, home, and embedded applications alike. But Java's strongest point—its portability—must be traded off against reduced performance, increased memory requirements, and a potential increase in cost. This tradeoff makes more sense in some situations than others.

Java is terrific at rotating icons—hardly a performance-intensive task—but missionary Javaphiles attempt to extend this paradigm to a completely network-based, diskless existence. Rather than storing applications locally, we should download them from the Web. Applets can be created by anyone and thrown, like a message in a bottle, into the Internet sea for everyone to use, free of charge. Applets are inherently safe, they're modular, and they play together without crashing.

Shareware may be fine for file-exchange utilities, but not for mission-critical applications. History teaches that few competent programmers distribute their wares for free. Companies like Corel have said they will sell complete Java application suites, but will end users really value portability enough to reduce their Pentium PC to the speed of a 486? Perhaps they'll just buy Microsoft Office instead.

For embedded uses (i.e., approximately 99% of the world's consumption of microprocessors), the tradeoff is even more clear. Most embedded applications are extremely cost sensitive and generally ship with their key software already installed. Since portability is of little value for ROM code, cost-focused embedded designers are well advised to compile directly to the native instruction set of their processor of choice.

Java bytecode is simply a nonsolution for these problems. As an object-oriented language, Java is great. But whenever performance or cost are important, programs should be cast in binary encoding, not bytecode. System designers who should know better are tossing memory space and performance out the window. It's time for the industry to wake up and smell the coffee. Pass the sugar. ▣