# Java mobile code

## by David G. Messerschmitt

Supplementary section for Understanding Networked Applications: A First Course, Morgan Kaufmann, 1999.



An illustrative example of MC middleware is Java from Sun Microsystems [Arn96][Fla96]. Java is several things. First and foremost, it is a “pure” object-oriented language, one that unlike C++ requires programs to be composed exclusively of objects. Java also defines an entire *mobile code system*, including a *virtual machine* and a set of *libraries*. MC and MO are generally based on objects and components, and Java is an example of this.

The Java language is modeled after C++, but with some important differences. First, Java is unabashedly object-oriented, as it dictates that programs are composed of collaborating objects (C++ doesn’t). Second, Java attempts to “clean up” C++ by removing features that make programs more difficult to write and read, and more importantly and make it unsafe as a mobile code language (see the sidebar "Java and Security"). Third, Java adds direct language support for some useful features for distributed computing applications, such as threads.

The Java *Virtual Machine* (VM) defines the instruction set of a virtual microprocessor—one different than most actual microprocessors—and then *emulates* this virtual processor on the real microprocessor. (A microprocessor that natively implements VM bytecode instructions is also possible.) The primitive instructions for the VM are written in *bytecode*, composed of instructions for the VM. A Java compiler translates the Java source code into bytecode that can then be exe-

### Java as a De Facto Standard

Although Java began as an elegant solution to the technical problem of software portability, it soon became a weapon in industry competitive battles, creating winners and losers. Because programs written in Java can in principle run under any operating system, Java potentially turns the operating system into a commodity (a good differentiated by price and quality, but not functionality or features). SUN advanced Java using industry alliances—in part to undercut the dominance of Microsoft Windows—following a multi-prong strategy:

- Gain broad industry support by licensing under attractive terms. Encourage others to add enhancements and complementary technologies.
- Avoid the *de facto* standard from splitting into incompatible versions by a licensing provision that affords SUN control over what is included in a single “pure Java” (in effect creating a *de jure* standard based on copyright protections).
- Try to establish the Java environment as its own operating system with thin-client products that run only Java.

Java has strong network externalities and is only useful (beyond a programming language) if widely adopted. Thus, Java is an interesting case study in establishing a *de facto* standard through a strategy of industry cooperation.

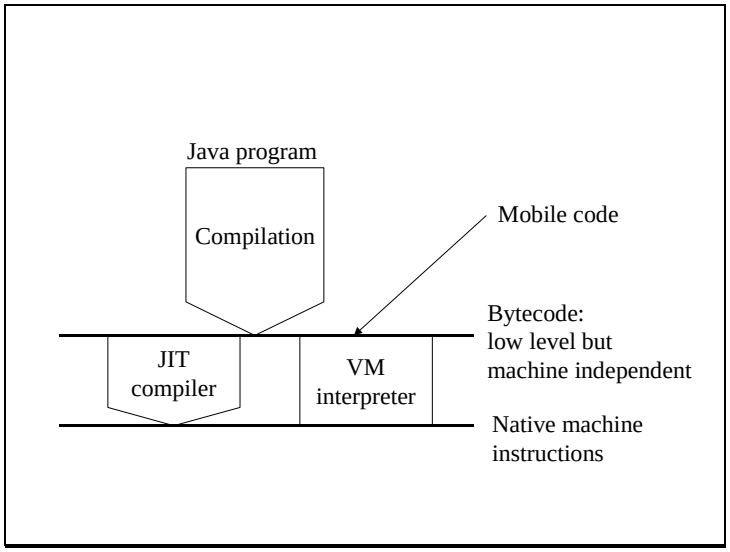


**Figure 1. Some options for compilation and execution of Java programs.**

cuted on any platform with a VM. There are a couple ways the bytecode can be executed on the target platform (see the sidebar "Other Options for executing Java"). This model of execution is called “write once, run anywhere” (see the sidebar "Java as a De Facto Standard").

Unfortunately portability requires much more than platform-independent program execution. Programs require access to network, machine (memory, storage etc.), and user-interface (display,

## Java and Security

MC is a security threat because—like a computer virus—it is an executable program originating from a potentially unknown source. Fortunately MC is *knowingly* executed (unlike viruses which propagate surreptitiously), and thus it is easier to enforce security policies.

MC security is addressed by appropriate policies (enforced by the VM) that establish a trade-off between safety and functionality. The unit of security in the Java system is the VM, and the policies govern the allowable access of VM to various resources at the granularity of objects (typically different for objects with a local or remote origin). An object’s ability to read and write local files or access to the network may be restricted. A VM acts like a firewall between MC and the remainder of the environment.

There is no guarantee that bytecode actually originated as a Java program—a clever intruder might write a nasty program directly in bytecode that bypasses any security measures in the compiler. For this reason and others, the VM includes a bytecode *verifier*, which tries to detect problems with the bytecode, such as operations that were supposed to be ruled out by the language and compiler.

The Java system is complicated enough that loopholes for determined adversaries are inevitable. Security policies tend to be invasive, precluding desired functionality. An effective security measure for the future may be authenticating the origin and integrity of MC using a digital signature. Less restrictive security policies might be imposed on MC authenticated as originating at a reputable source (like a major software company) if its integrity can also be verified.

### Other Options for executing Java

As shown in Figure 1., there are two ways to execute bytecode on the target platform. One is interpretation, and the other is a second stage of platform-dependent compilation that translates Java bytecode into native instructions. Without interpreter overhead, the resulting native code will execute considerably faster than bytecode. Since this second stage of compilation is typically done "on the fly" as the bytecode is executing the first time, it is called *just-in-time (JIT) compilation*.

It is also possible to provide a *single* stage of compilation directly to a native instruction set—using Java for conventional (non-mobile) application programming. Some think that Java will become the standard programming language. In part, this is because of its nice features, and in part because of networked externalities in programming languages. Using the most widely used language gives access to a trained pool of programmers, a better suite of development tools, and a clearer path to ongoing maintenance. Java is currently benefiting from the positive feedback effects.

keyboard, mouse, etc.) resources. Thus, a mobile code system must provide similar resources, but in a platform-independent way. To this end, Java provides a set of libraries, with well-documented interfaces for a suite of standard actions (like accessing files, creating graphical objects, connecting to the network, etc.). These libraries must be maintained on each platform. A mobile code system must also provide security measures.

## Discussion

D1 Discuss the strategy SUN Microsystems has employed to establish Java as a de facto standard while maintaining control over its features.

## Concepts

Java:

- bytecode
- virtual machine
- ust-in-time compilation