JavaOS™: A Standalone Java™ Environment

A White Paper

JavaSoft
A Sun Microsystems, Inc. Business

2550 Garcia Avenue
Mountain View, CA 94043 U.S.A
408-343-1400

May 1996

JavaOS™: A Standalone Java™ Environment

A White Paper

Peter W. Madany

With contributions by
Graham Hamilton and Jim Mitchell

JAVASOFT
A Sun Microsystems, Inc. Business

2550 Garcia Avenue
Mountain View, CA 94043 U.S.A.
408-343-1400

May 1996

1.0 Introduction

JavaOS™ is a small and efficient operating environment developed by JavaSoft that executes Java™ applications directly on hardware platforms without requiring a host operating system.

The Java Platform enables the development of secure, high performance, and highly robust *applications* on multiple platforms in heterogeneous, distributed networks. The goal of JavaOS is to enable the development of secure, high performance, and highly robust intelligent and dynamic *network devices* built on multiple *hardware* platforms in heterogeneous, distributed networks.

By intelligent devices we mean that they are capable of executing programs locally, even though they are inherently network-oriented. By dynamic we mean that instead of being limited to a static set of programs contained in local storage, these devices can automatically load and run new programs over their network connections. These devices must also be easy to install, administer, and use. Installation should be as simple as plugging them into a network and a power supply and turning them on.

Since the benefits of the Java Platform fit many of the goals of building simple, intelligent, and dynamic network devices, the challenge became providing the Java Platform in devices with limited hardware and software resources. So we designed JavaOS to provide just enough operating system features of just the right kind to support the Java Platform.

The reason for limiting hardware resources is that lesser hardware requirements allow more types of devices to be built and sold. One of the best ways of reducing a device's hardware requirements is to remove the overhead caused by requiring a host operating system.

2.0 Java with a Host Operating System

The Java Platform can be provided in many environments by embedding the Java Virtual Machine and foundation classes within an application, like a Web browser, or embedding them within an operating system. Both of these methods rely on the support of a host operating system.

Figure 1 illustrates the software architecture used when running Java programs on a conventional operating system. The most important line in this diagram is the line that defines the Java API. Above that line all Java programs are platform independent. It doesn't matter what the underlying operating system or hardware is, the API is the same on all platforms. The API is implemented by several packages of classes written in the Java programming language, including the language and utility classes, the Abstract Window Toolkit, and the network and IO classes. The Java run-time, which includes the interpreter and garbage collector, is largely written in platform-independent C code. To port the run-time to a new platform, however, some specific code must be ported. Also there are various libraries that map Java classes to OS-specific library or system calls.

Each of the major features provided in the Java Platform directly or indirectly places requirements on a host operating system. Many of the language and utility classes assume operating system services are available.

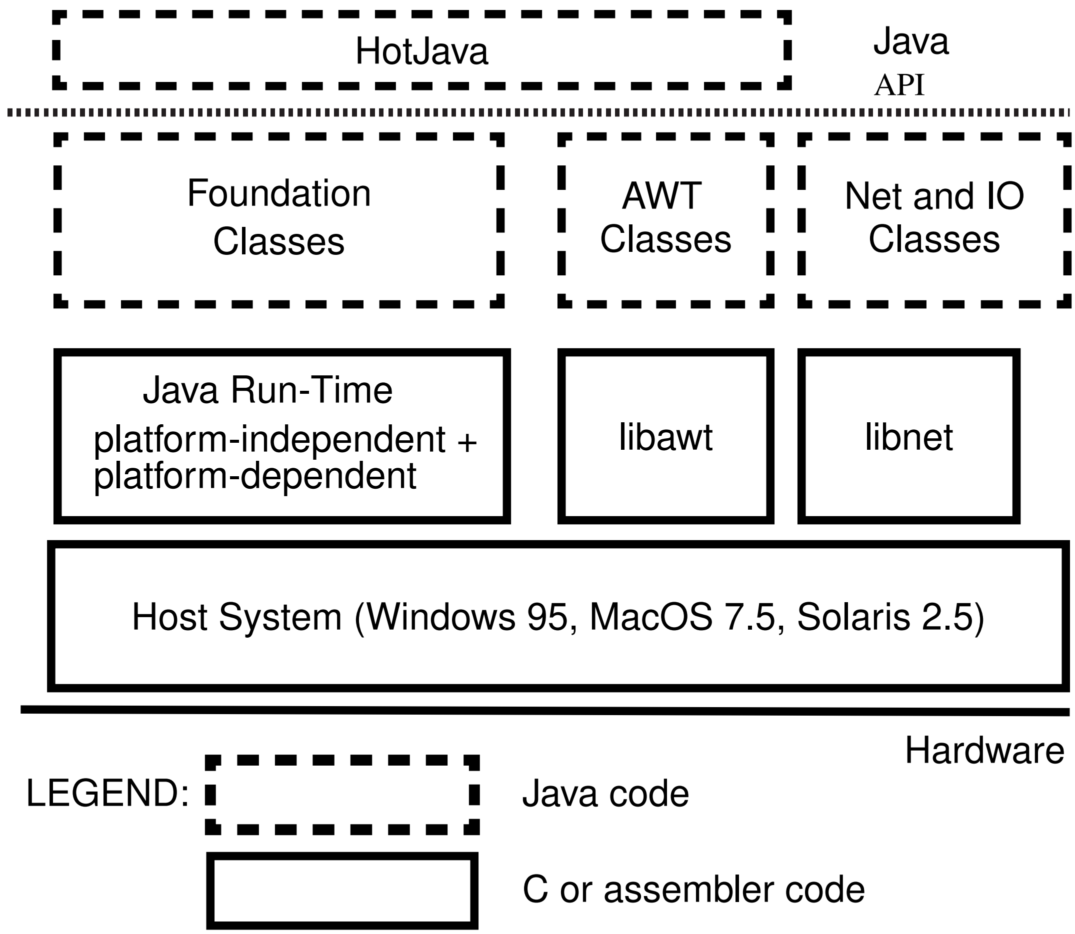
In order to support the Java run-time's multi-threading, the host system must provide at least some primitive support for context-switching. If the system provides better support for threads, that feature can be leveraged. Though the Java run-time manages its own heap and includes garbage collection, it still needs a mechanism for allocating the memory that it will manage.

The Abstract Window Toolkit provides an abstract graphical user interface, but it relies on a particular host operating system to implement the windowing and graphics primitives.

The networking classes provide convenient abstractions for programming in the Internet or in an intranet, and they also rely on the host system to provide an implementation of a suite of standard network protocols.

The file-related IO classes assume that there is an underlying file system, and indirectly Java applets assume that a computer has drivers for devices like a keyboard, a mouse, and a display.

FIGURE 1. Java on a Host Operating System



To match the Java run-time and classes to a particular host operating system, one must accomplish several tasks, including the following:

- map the Abstract Window Toolkit (AWT) to the window and graphics subsystem provided by the host system.
- map the networking classes to the native networking code on the system, which for example could have different system calls for operations on sockets.
- map the file-related IO classes to the host file system, which might use a different syntax for file-names.
- port the platform-dependent part of the virtual machine to the particular system calls for memory allocation and thread management.

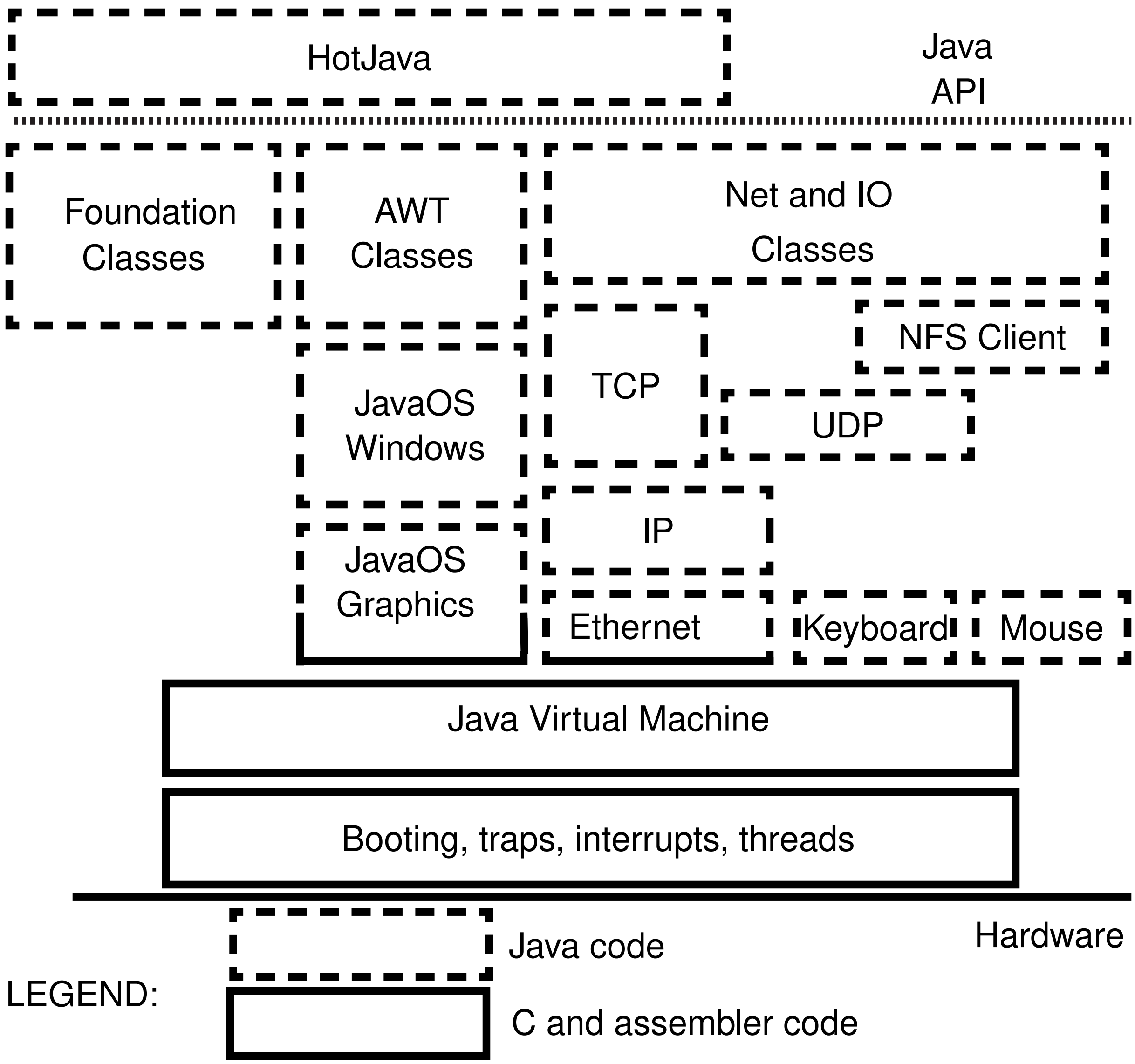
3.0 Java without a Host Operating System

In order to meet the goal of providing the Java Platform without a host operating system, we had to perform the following tasks:

- implement just enough kernel features to support the Java Virtual Machine.
- leverage those kernel features and the Java Virtual Machine so that we could write just enough Java code to support AWT, and the networking and file-related IO classes.
- provide the drivers needed to control a display, network interface, mouse, and keyboard.
- ensure that we still support the full Java API.

Figure 2 shows a high-level view of the JavaOS architecture. As in Figure 1, the most important line in this figure is the line that defines the Java API. Above that line all Java programs and applets are platform independent. It doesn't matter what the underlying operating system or hardware is, the API is the same on all platforms. The API supported by JavaOS is implemented by the exact same packages of classes written in the Java programming language.

FIGURE 2. JavaOS architecture



3.1 A Java Kernel

Within JavaOS, the Java run-time executes not only user-level applications, but also the system-level windows, graphics, networking, and driver code. The lowest layer of code handles the tasks often found in micro- or nano-kernels. But since the entire architecture of JavaOS is driven by the goal of supporting Java applications, the bottom layer is probably better called a *Java kernel*. The Java kernel contains the low-level functions that are required by the Java Virtual Machine.

While the bootstrap code is running, it allocates several memory regions including one for the Java heap and others for various IO device registers and DMA regions. The bootstrap also handles mapping hardware devices that it has detected to their corresponding device drivers.

The main purpose of the trap and interrupt handling code is to service traps and interrupts and make the information accessible to the appropriate Java device driver.

The thread support code enables the Java Virtual Machine to switch contexts between the dozens of threads typically running in the system. Because of the protection provided by the Java programming language, JavaOS and all applications can run in a single address space. This simplifies and optimizes the context-switching code in JavaOS.

JavaOS does not require a Memory Management Unit, but it can use an MMU to make several disjoint ranges of physical memory appear contiguous, which simplifies memory allocation. In any event, JavaOS never modifies the MMU after boot time.

Because the Java Virtual Machine provides a safer programming environment than traditional systems, the entire system, including downloaded applets can be run in the supervisor mode of the CPU. Running in supervisor mode only and not requiring an MMU both allow JavaOS to be simpler and more efficient than traditional operating systems.

3.2 Java Virtual Machine

Within Java-compatible systems like JavaOS, the Java Virtual Machine is obviously used to interpret Java byte-codes, but it is also used as infrastructure for much of the rest of JavaOS. It executes the byte-codes in all classes within the system, handles exceptions, manages almost all of the RAM in the computer, and handles the simultaneous execution of multiple threads.

The implementation of the virtual machine that we use in JavaOS is very close to the standard one provided with JDK 1.0 for Solaris, but we have tuned the memory allocation mechanisms, and added the reclamation of storage used by classes that are no longer needed by any objects in the system.

3.3 Device Drivers

All device drivers in JavaOS are written in the Java programming language. This may surprise some people, but so far we have been successful and are satisfied with this approach, including being satisfied with the performance of the drivers.

Obviously there are certain things almost every driver needs to do that cannot be done in pure Java code, but we have abstracted those tasks into two small support classes written in C. The MemoryObject class enables drivers to access and modify specific bytes and words of storage. The other class handles interrupt dispatching. Obviously the methods of these classes are *not* made available to any Java applications.

We currently have over a dozen drivers written in Java for several different classes of devices, and we are developing many more. We are also in the process of defining a Java Device Driver Interface to enable third parties to build loadable drivers for any device.

3.4 Network Protocol Suite

JavaOS includes a large suite of network protocols, all written in the Java programming language. These protocols include the basic transport and routing mechanisms specified by the TCP, UDP, IP and ICMP standards. JavaOS uses both DNS and NIS for looking up hostnames and supplying user names and passwords used during login.

JavaOS supports both Reverse ARP and DHCP for discovering the network address of a device. This enables the JavaOS machines to be installed with little or no per-machine administration.

A machine running JavaOS can access files as a client of a Network File System server and can be managed using the Simple Network Management Protocol. JavaOS machines can get the time of day from a network server, which also simplifies installation and administration.

3.5 Window and Graphics

Besides the network protocol suite, the largest piece of operating system functionality supplied by JavaOS is the windowing and graphics subsystem.

JavaOS uses the so-called "Tiny AWT" library to provide Java-based implementations of widgets like buttons, menus, and scrollbars. Tiny AWT is not called "Tiny" because it is smaller than AWT, but because it enables AWT to place far fewer requirements on the underlying window system.

We have developed a simple and memory-efficient window system in Java. Besides supplying a system to manage the display of overlapping windows, we also had to write a package to draw and fill lines, arcs, and polygons, and to render bitmapped fonts. The lowest levels of the graphics code need direct access to the frame buffer memory locations; therefore, they are written as native methods.

3.6 JavaOS API

The Application Programming Interface of JavaOS is implemented by the exact same packages of Java classes that implement the Java API on other platforms. Therefore applications written to run on JavaOS will run on any other Java-compatible platform and vice versa.

3.7 HotJava as the JavaOS Desktop Environment

Because HotJava is written in the Java programming language, it can run on JavaOS. When combined with HotJava, JavaOS can function as a complete multi-tasking, graphical operating environment. HotJava can serve as the manager of the desktop metaphor, supporting multiple windows, each of which is capable of browsing HTML pages and running one or more Java applets.

JavaOS can also run other main programs besides HotJava, and HotJava's customizability and extensibility make it an ideal framework for JavaOS applications.

4.0 Is JavaOS an Operating System?

Is JavaOS really an operating system? Whether it is or isn't depends on one's perspective. JavaOS differs from conventional operating systems in several ways:

- it does not need a file system,
- it does not need virtual memory,
- it does not need separate address spaces,
- it does not support more than one programming language, and
- it does not have its own set of system calls.

JavaOS also resembles an operating system in several ways:

- it is bootable,
- it supports a password-protected login feature,
- it safely runs several applets at a time,
- it includes several device drivers,
- it communicates using many standard network protocols,
- it has its own window system,
- it has an API---the Java API, and
- it will run the thousands of applets and applications that have been written.

5.0 Performance

5.1 Speed

Currently JavaOS has had little performance tuning, has not used a "Just-In-Time" compiler to translate byte-codes into machine code, and makes minimal use of native methods. Therefore one might expect JavaOS to perform poorly, but our first measurements and benchmarks indicate that performance is not only better than expected but also better than some more mature systems written in C or C++.

The TCP/IP throughput on JavaOS is already double our original goal and far more than adequate for Web browsing. We have also run Pendragon Software's CaffeineMark™ benchmark on several systems to compare their performance when executing Java applets, and we have observed very encouraging results. The main reason for these good results is that JavaOS jettisons the many layers of software architecture that other systems have built up to make programming in languages like C safer and less platform-dependent. Ironically, while those layers do hurt performance, they do not come close to providing the safety and platform independence of Java!

5.2 Space

How much memory is needed to support JavaOS? One could build a complete system with a total of 4MB of ROM and 4MB of RAM.

In the ROM would be all the code for JavaOS itself, including the kernel code, drivers, Java Virtual Machine and standard classes, plus the JavaOS windows, graphics, and networking components, plus the code for HotJava. The ROM could also include approximately 1MB of bitmaps for fonts with all the combinations of various types like serif, sans-serif, and typewriter, several point sizes, and various styles like bold and italic.

Assuming that the ROM could be executed in place, then the system could use two and one-half MB of RAM for the dynamic requirements of JavaOS and HotJava and still have about one and one-half MB of RAM for downloaded HTML pages, applets, and images. Systems built using JavaOS that do not require windowing and HotJava code could run in less than half the space.

6.0 Advantages of JavaOS

There are several advantages of using JavaOS to provide the Java Platform directly on hardware, including:

- JavaOS achieves the goal of eliminating the overhead of a host operating system. Because JavaOS contains no extraneous features found in other operating systems, it allows smaller and simpler devices to be built that execute Java programs more efficiently than other systems.
- JavaOS is ROMable, enabling simpler and lower-cost systems that can boot faster
- JavaOS benefits from being written in Java: new components can be developed for it faster because Java code is easier to debug, is inherently portable, and is dynamically extensible.
- JavaOS enables systems that are as easy to install and maintain as terminals, yet are nearly as powerful as traditional desktop machines. Perhaps the most expensive part of owning a computer is the cost of configuring and maintaining one. JavaOS can dramatically reduce that cost compared with a typical personal computer.

7.0 Target Systems

JavaOS is ideal for several types of devices, including intranet computers, Internet computers, and embedded devices. As the name implies, JavaSoft develops software not hardware, and it supplies JavaOS to hardware companies to enable them to build intelligent and dynamic hardware devices.

7.1 Intranet Computers

Intranet computers are computers connected to an enterprise's network infrastructure. Most companies can deliver Ethernet connections to each desktop machine, which provides plenty of performance for transferring Java classes and other data. Many companies also have higher speed backbone networks.

Servers are another key element of enterprise networks. They support centralized administration of intranet computers. For example, if a server implements the DHCP protocol, which supports dynamic allocation of IP addresses, no administration is required for installing an additional intranet computer.

The Java Platform is ideal for developing and deploying MIS applications, since applications can be automatically downloaded over a network and since it is so easy to write network-aware applications in the Java programming language.

7.2 Internet Computers

The first Internet computers will have to work well with the relatively limited bandwidth provided by today's high-speed modems and even ISDN. When cable modems become more prevalent, then home computers will have the network bandwidth common in today's local area networks.

Since most people don't have sophisticated network servers at home, Internet Service Providers will have to provide the necessary infrastructure so that using an Intranet computer is as easy as using an appliance. For example, you just plug the computer in to an outlet and a phone jack, turn it on, and you are automatically on the Internet and ready to go surf the World Wide Web.

7.3 Embedded Devices

What if a device has only 1 or 2MB of RAM and 1 or 2MB of ROM, and possibly no graphical display, yet you want it to be able to load and run Java applications?

JavaOS can be tailored to fit particular devices like set-top boxes, PDA's, and even electronic devices without any graphical display. Note this does not mean that one could subset the language itself or remove features from the language or utility classes. But, for example, if there is no display, one could remove not only AWT for that device, but also remove the window and graphics code from JavaOS. Similarly if the device did not have a need for certain network protocols, they could be eliminated.

In order to meet some embedded system requirements, we still have to tune the Virtual Machine and garbage collection to support some soft real-time capabilities.

We are working with software tool vendors to build a rich software development environment for JavaOS, including a remote debugging capability.

The memory footprint for JavaOS in its smallest possible configuration will be about 128K of RAM and 512K of ROM. Note that this is the memory required for JavaOS itself; additional memory would be needed for applications.

8.0 Availability

JavaOS currently runs on two sets of platforms. The first set of platforms is based on SPARC microprocessors. The second set of platforms is based on microprocessors compatible with Intel's x86 instruction set. JavaOS is also being ported to systems based on other microprocessor instruction sets.

We currently plan to make JavaOS available to OEMs so that they can build and sell devices using JavaOS by October 1996.

The current version of JavaOS is based on the 1.0 version of the JDK.

Some of the key features we will be adding in future versions of JavaOS include:

- new Java Platform API's as they are added to the future versions of the JDK,
- the device driver interface that we are currently specifying,
- enhanced window and graphics components with features like scalable fonts, and
- an enhanced network protocol suite with PPP-related features.

9.0 Summary

In summary, JavaOS is a new operating environment that enables Java applications to run directly on hardware without requiring a host operating system.

JavaOS includes the Java Virtual Machine, the standard packages of classes, and just enough OS code to support them. The OS code includes low-level code written in C or assembly language, plus device driver, networking, windowing, and graphics-rendering code written largely in the Java programming language.

Though it is predominantly written in the Java programming language, and has yet to be tuned for performance, JavaOS already executes Java programs faster and more efficiently than more mature operating environments.

The main advantage of JavaOS is that by eliminating the overhead and complexity of host operating systems, it enables new classes of simple, intelligent, and dynamic network devices that will be cheaper to build and own.

JavaOS is targeted at systems such as intranet terminals for enterprise desktops, consumer Internet computers suitable for Web surfing, and embedded devices where hardware resources are even more restricted. It is currently running on two sets of platforms, is being ported to other platforms, and will be available in devices built later this year.

JavaSoft
A Sun Microsystems, Inc. Business

2550 Garcia Avenue
Mountain View, CA 94043
408-343-1400

For U.S. Sales Office locations, call:
800 821-4643
In California:
800 821-4642

Australia: (02) 844 5000
Belgium: 32 2 716 7911
Canada: 416 477-6745
Finland: +358-0-525561
France: (1) 30 67 50 00
Germany: (0) 89-46 00 8-0
Hong Kong: 852 802 4188
Italy: 039 60551
Japan: (03) 5717-5000
Korea: 822-563-8700
Latin America: 415 688-9464
The Netherlands: 033 501234
New Zealand: (04) 499 2344
Nordic Countries: +46 (0) 8 623 90 00
PRC: 861-849 2828
Singapore: 224 3388
Spain: (91) 5551648
Switzerland: (1) 825 71 11
Taiwan: 2-514-0567
UK: 0276 20444

Elsewhere in the world,
call Corporate Headquarters:
415 960-1300
Intercontinental Sales: 415 688-9000