Java on AIX—A Strong Brew

By Jeff Jilg

Java is a rapidly evolving technology. Its platform independence and Internet interoperability features help to enhance its popularity. Java on AIX allows programmers to exploit this technology and users to access newly created Java applets. Security is a concern to many, and this has been taken into account through integration of security features as part of the base infrastructure in the technology. Although Java allows flexibility, it requires both programmers and administrators to be aware of the potential problems that can result from careless implementation.

The Java™ technology has created much excitement in the past year, but it has also resulted in some confusion. What is Java? Can it be used to deliver only Internet or Web page-based programs? Is the environment similar on different platforms? Where should a programmer begin? What do system administrators need to know? How about security? The objective of this article is to answer these and other Java-related questions and to provide references for further exploration.

The quickly evolving nature of any new technology often creates some confusion, and Java is no exception. Java was introduced to a marketplace that was looking for options and adaptability. At the same time, companies still expect some standardization and interoperability to reduce platform dependencies and provide easy migration paths. Let's examine some Java technology components and the benefits they deliver.

A Brief History of Java

The Java technology, initially developed in 1991 at Sun Microsystems® under the Green group, was aimed at the television set top box market of the commercial electronics industry. After a couple of name changes, Java and HotJava™ were announced at SunWorld '95 in San Francisco on May 23, 1995.

Interest in the technology grew rapidly through the remainder of 1995. IBM announced licensing of the technology in December 1995 and Microsoft® also announced their intent to license Java. IBM has formed a Centre for Java Technology in Hursley, England where they are actively porting the technology to various platforms including AIX.

In May 1996, Sun®, IBM, and others sponsored the JavaOne™ conference, which attracted more than 5,500 attendees. With the enormous amount of activity focused on Java, it is definitely still in the growth phase. As with any new technology, initial delivery is followed by availability on various platforms, enhanced development tools, then new applications that occur throughout the various phases as the technology matures. Many believe that major applications will begin appearing in numbers during late 1996 and early 1997.

The Java Technology

The base Java technology has two primary components: Java Virtual Machine (JVM) and the Java language. The Java Virtual Machine (JVM) is an abstract CPU specification that provides a runtime environment for all Java programs. The specification looks much like a traditional CPU with a defined bytecode instruction set that can interact with the abstract CPU. Since abstractions cannot be used to run physical programs, this abstract specification must be converted to an executable environment on a real system.

Sun has a reference port of the JVM (and other related components) available. Various companies, including IBM, have licensed this reference code and ported it to their respective platforms. Thus, IBM's AIX port of the Java technology includes the JVM. Most programmers and site administrators will not need to learn the details of the JVM because higher-level interfaces provide easier access to the technology.

Most programmers use the Java language, the primary interface to the JVM, to take advantage of Java technology. Because the language is a C++ derivative, it can be used to write a variety of different programs. The Java language is also object-oriented, which provides several advantages including reusability and encapsulation of data and program behavior into objects.

Java Versus C++

Java and C++ differ in several areas. The Java language lacks pointers—the first big difference most programmers will notice. These pointers were specifically removed to reduce the security exposures because of their ability to trash memory. Another problem with C++ pointers is that if they are misused, the result is often a disproportionate amount of bugs. The Java language does pass all arrays and objects by reference in place of pointers.

Another difference is that C++ supports multiple inheritance; Java does not. Multiple inheritance allows programmers to create object classes that inherit data and/or behaviors from more than one parent class. This feature is implemented through Java interfaces that provide method descriptions without implementations.

Platform Independence and Other Benefits

Java technology is object-oriented, but it also has other benefits that are helping to drive its popularity into the stratosphere. The primary benefit is platform independence. Programs can be developed on one platform, then executed on any platform that supports the JVM and associated runtime components (as shown in Figure 1). This powerful concept allows a company to develop a program in one language (Java) and avoid the expense of porting it to various target platforms.

Platform independence also affects support issues. Because Java programs have a single code base, support personnel can now focus their expertise on the problem, not platform-specific issues and the various code bases associated with different platforms. This benefit also extends to end users, who can now receive a single compiled Java program that can run on any machine in their heterogeneous environment.

Dynamic executable content allows users to easily download and execute a Java applet. It is enabled through incorporation of the JVM and Java runtime engine into numerous Web browser environments, such as appletviewer (discussed below) and Netscape Navigator® (Version 3.0 on AIX). End users can receive programs over the Internet through a very simple process. A user can click on a hypertext link on a Web page and automatically download a Java applet (program), as shown in Figure 2.

The HTML associated with the Web page must contain the <APPLET> tag with a reference to the Web server holding the applet and the applet name. When the user clicks on the link, a request is sent to the Web server to send the applet to the browser. After the Web server has transmitted the applet (and any images) to the browser, the browser then executes the Java applet.

Versioning control is another benefit of this scenario. Suppose a user already has a Java applet that is used daily. Each morning the applet is started. It could communicate with the host at the sponsor company, obtain any new updates or bug fixes, and automatically install them on the client. Some updates could possibly advertise new features or additional components

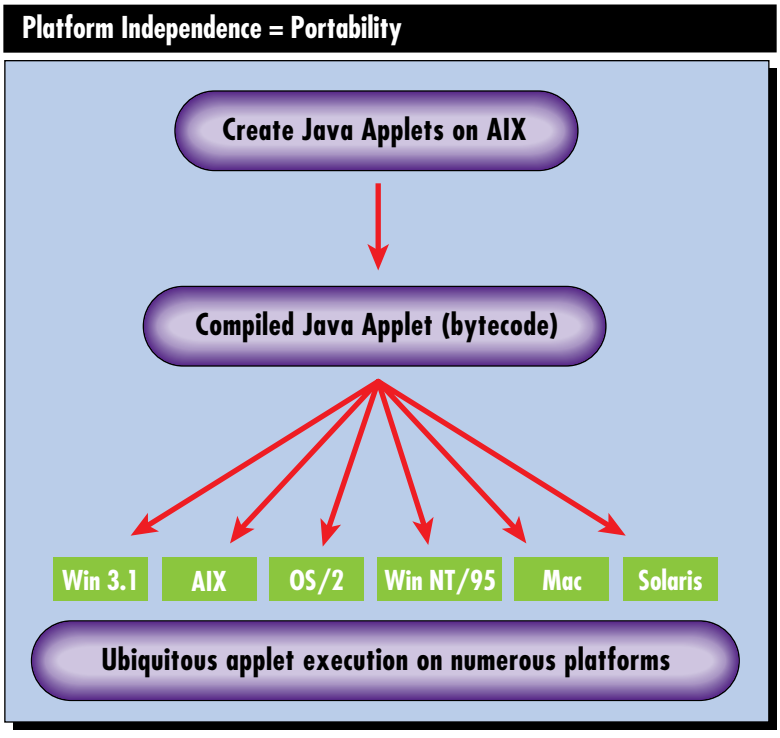

Figure 1. Platform independence = portability

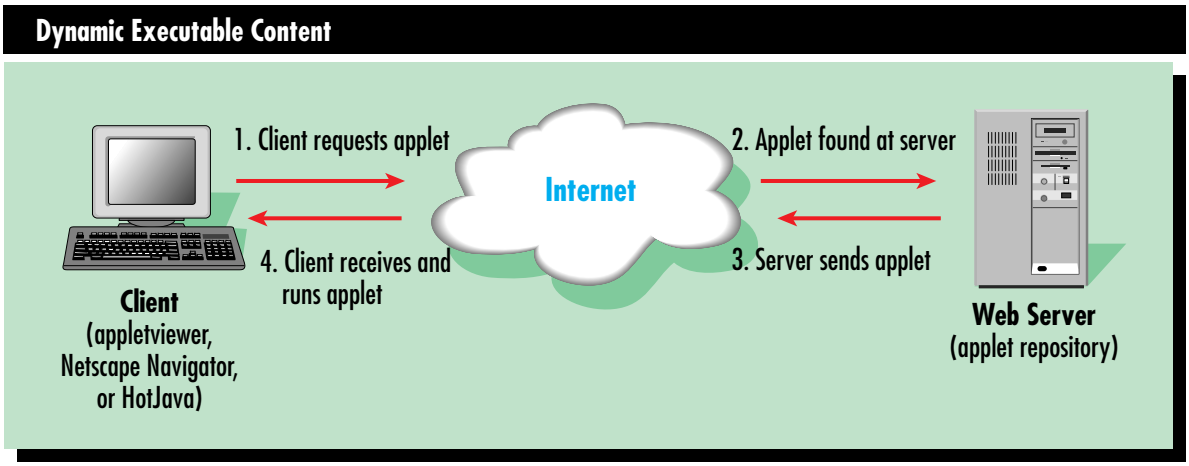

Figure 2. Dynamic executable content

that end users could then choose to purchase over the Internet.

Although Java does not directly deliver this scenario in the native runtime environment, some creative programming can easily extend the features of the environment. The possibilities are almost endless.

The support staff can also benefit from this type of functionality. Online installation and updating can virtually eliminate the costly physical mailings that support has relied on to deliver program bug fixes and updates.

Development Components	Description
compiler, debug runtime, debugger	Compiles Java source to bytecode and debugs it
javadoc tool	Creates HTML files from comments in Java source
header and class files	Used by compiler to create compiled Java classes
class file disassembler	Extracts information from compiled Java class file
javah tool	Generates C headers and stubs for C programmers
Runtime Components	**Description**
java interpreter	Dynamically interprets bytecodes to native platform
runtime libraries	Precompiled libraries get accessed by applets
appletviewer	Graphical execution environment to run applets without need for browser
demo applets	Source code and runtime to show capabilities
readme	Base documentation showing platform specifics

Figure 3. Java Development Kit contents

Java on AIX

Sun provides a set of code for Java licensees called the Java Development Toolkit (JDK). The JDK provides a complete development and runtime environment, including the JVM discussed above. IBM has made the JDK available on the AIX, OS/2, and Windows 3.1 platforms.

AIX 4.2 includes the JDK as a component of the Bonus Pack (along with others like Netscape Navigator). All Bonus Pack components are optionally installable, and the Bonus Pack itself is provided at no additional charge with AIX 4.2. Bonus Pack components are documented in both the Bonus Pack `readme` and respective component `readme` files. The JDK is also available on AIX 4.1.3 from IBM's Java Web page, referenced at the end of this article.

The JDK contents can be subdivided into two logical groups: development and runtime components. The contents are listed in Figure 3.

Java Compatible

The JDK on AIX (and on OS/2) represents a Java-compatible port of Sun's Java. Java-compatible ports are easily recognizable because they have the rights to show the "Java compatible" logo with the characteristic Java coffee cup icon.

Java compatibility is important because IBM has tested and passed the Sun test suite for the AIX and OS/2 JDKs. The objective of the test suite is to ensure interoperability between JDK ports on various platforms. Java compatibility means that AIX developers can create Java applets on AIX, and those applets should exhibit the same behaviors on AIX that are exhibited when the applet executes on another platform. The inverse is also true; that is,

applets developed on another platform (such as IBM's Windows 3.1 JDK) will exhibit the same execution behavior when run on AIX's JDK.

Newcomers to Java

If you are new to Java on AIX, try out the new features. Use `installp` or the System Management Interface Tool (SMIT) in the normal AIX fashion to install Java from the Bonus Pack. The install will deliver the files into `/usr/lpp/Java`. Before executing any of the development or runtime tools, be sure to set up your environment. Currently, to do this, add the Java binaries to your system path from the command line by typing the following:

```
export PATH=$PATH:/usr/lpp/Java/bin
```

The AIX JDK (standard) has 23 sets of precompiled demos. All can be accessed through any Java-enabled browser; they can also be executed using the appletviewer. For example, to run the BouncingHeads demo[1] type the following:

```
appletviewer /usr/lpp/Java/demo/
  BouncingHeads/example1.html
```

or

```
cd /usr/lpp/Java/demo/BouncingHeads
appletviewer example1.html
```

Note that the argument we sent to the appletviewer is an HTML file. The appletviewer cannot directly receive a Java applet, but must parse an HTML file and find an `<APPLET>` tag in the file that refers to a Java class (compiled applet). The appletviewer can also execute Java applets directly from the Internet. For example, the following command will access the IBM Hursley home page to run the multiple applets contained within that URL (currently two).

```
appletviewer http://ncc.hursley.ibm.com/
  javainfo/
```

Since the appletviewer is intended as a tool to run Java applets, it does not display the Web page referenced by the HTML. Its purpose is to run applets, so it finds all the `<APPLET>` tags embedded in an HTML file, starts independent windows on the desktop, and executes the applets in those respective windows.

Development and Runtime Tools in JDK

More intrepid AIX users will want to try out the development tools included in the JDK. The simplest place to start is by recompiling one of the existing demos, such as the ArcTest demo. This demo displays an arc segment of a circle and optionally fills the area. Compilation is simple. The compiler takes source code with the `.java` file extension and delivers compiled bytecodes contained in Java class files as output. In this example, type the following:

```
cd /usr/lpp/Java/demo/ArcTest
javac ArcTest.java
```

Use the appletviewer to test the newly compiled Java class files:

```
appletviewer /usr/lpp/Java/demo/ArcTest/
  example1.html
```

This compilation example takes the single `ArcTest.java` source file and produces three compiled class files as output (`ArcCanvas.class`, `ArcControls.class`, `ArcTest.class`). It is a common practice to define multiple object classes in a single Java source.

One of the tenets of object-oriented programming is reusability. The way to achieve that is to create modular programs that output multiple compiled object classes, which can be used in another Java program.

The primary benefit of Java technology is platform independence.

At least four different options are available to those who want to run Java programs:

- Appletviewer is included with the JDK for direct applet execution on AIX.
- Netscape Navigator (Version 3.0) has been Java enabled. This Navigator version renders both the HTML from a Web page and also executes any applets referenced (and available) by `<APPLET>` tags in that HTML. AIX 4.2 currently includes Netscape 2.0. Since IBM is very active in this area, be sure to monitor announcements for updates to AIX. Other browsers will probably be Java enabled over time.
- The HotJava browser is another option for applet execution. HotJava is a Web browser environment written in the Java language. The

[1]The BouncingHeads demo shows some heads bouncing around in a window. This demo was written by Jonathan Payne, the original author of the HotJava browser.

browser executes in the Java runtime environment. When AIX 4.2 was shipped, HotJava was only available in the alpha stage, so it is not currently included with AIX 4.2.

- Full Java applications (not applets) provide another way to run Java programs. Java applications can be executed directly from the command line by typing the following:

```
java application_name
```

The most well-known Java application is HotJava. Some differences between Java applets and applications will be discussed in the next section.

Applets Versus Applications

Distinct differences exist between applets and applications. Most Web users to date have executed applets, which are downloaded via the Web after a Web browser takes action on the <APPLET> HTML tag. Important differences between the two Java program types are highlighted in Figure 4.

Applets execute within the context of an environment whereas applications do not. Most projects that have a target audience that uses the dynamic executable content model will be programmed as applets. With applets, the program can be referenced by a Web page and loaded over the Internet. However, this model has security issues, which will be discussed in the next section.

With applets, most users expect the download time from Web server to Web client to be minimal. On intranets, bandwidth congestion is probably not a major concern. But using the Internet can be frustrating if too much time is spent during a Java applet download. As a result, most applets are small because of necessity, since users will terminate a download if too much time is taken to transfer the applet. Consequently, applications are typically larger in size than applets. New Java programmers might consider reducing the size of graphics files sent with applets.

Resource access in Figure 4 pertains to both the Java runtime environment and the respective browser environment used. Applets have many resources restricted from them. The intent behind this restriction is to keep the (few) bad guys on the Internet from creating hostile applets that can penetrate your security defenses. Programmers who must access a variety of system resources from a Java program should design their programs as full-fledged applications.

Java Applet
1. Requires a browser to execute
2. Runs within the context of the browser (usually Netscape Navigator)
3. Uses network model—applet downloaded/executed dynamically
4. Typically small
5. Has restricted access to local filesystem and to network resources

Java Application
1. Executes independently of browser
2. Requires Java runtime support; runs from command line
3. Has access to network, but invoked outside of browser

Figure 4. Differences between Java applets and applications

Security

Security of data and systems is a primary concern for everyone. To date, most security concerns related to Java have been discovered by a few researchers who have publicized these security flaws on the Web. However, few, if any, actual security incidents have been attributed to hostile Java applets, intentional or otherwise.

Java has several levels of security embedded in the technology.

Levels of Security

Java has several levels of security. In addition, different environments apply additional security policies on top of those already provided. As noted earlier, Java is delivered with AIX 4.2 as an optionally installable package. So an extremely conservative security policy would dictate that Java never be installed. Fortunately, the technology is relatively secure so this approach may be too conservative. Core security is encapsulated in the technology delivered from Sun (as shown in Figure 5).

The first level of security is found in the Java language. The language is a C++ derivative, but does not include pointers as was discussed in the Java technology section earlier. This lack of pointers helps protect memory from being accidentally (or purposefully) trashed by errant memory access outside the scope of an applet. The compiler also enforces strong typing, which ensures that the runtime variable is a subclass of the compile time type.

The next level in the security model is encapsulated in the Java runtime environment. The runtime environment is embedded in the JVM implementation as delivered from Sun in the reference port. The runtime includes a bytecode

verifier, a secure (object) class loader, and a security manager module. These modules ensure that the bytecodes comply with built-in security policies and the bytecode specification for the JVM. The policies include access restrictions, memory access constraints, system resource access, and compliance with network access policies.

Java interpreter implementation adds a third level of security. Critical policies are listed in Figure 6 for applets running under appletviewer and Netscape Navigator (Version 3.0) and also for applications.

The policies listed in Figure 6 indicate the maximum amount of authority that can be granted to an execution environment. While not all functions are configurable by the end user, default configuration for most functions is very conservative. For example, the appletviewer cannot read, write, or delete any files unless the end user explicitly allows this through configuration changes.

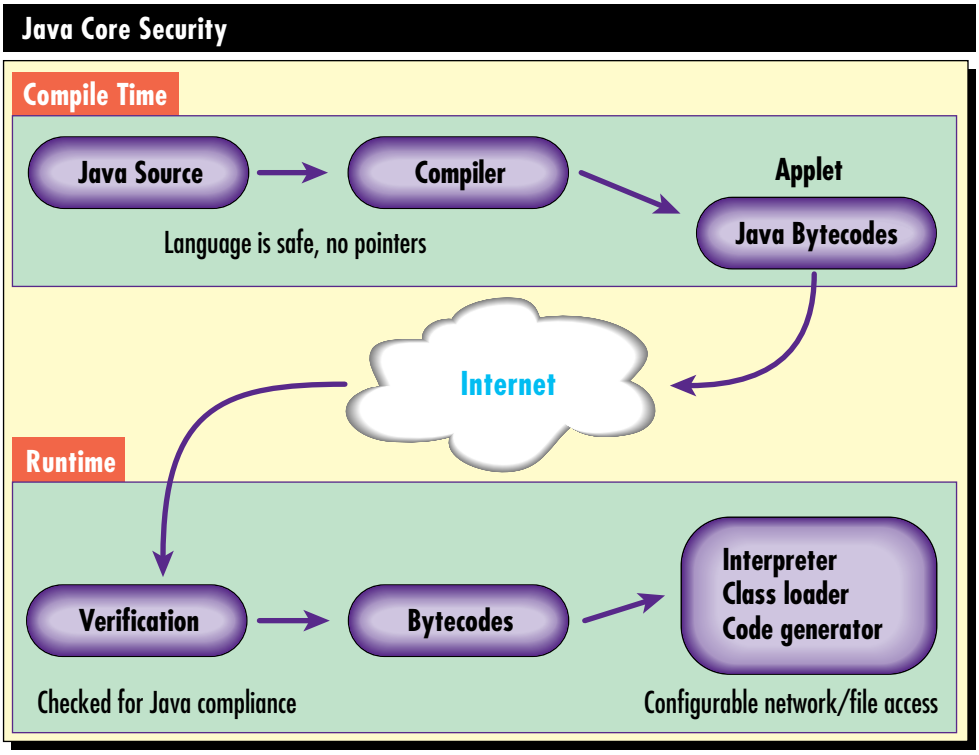

Figure 5. Java core security

There are two ways to load applets into the appletviewer and Navigator. When a user loads a URL, the result is a set of Internet transactions. Since this explicitly opens traffic to the Internet, default policy for both environments is more restrictive than the `load file` mechanism. When a user loads a Web page using `load file`, the result is retrieval of a Web page from the local (or distributed) filesystem. In this case, the appletviewer can be configured to have more access to the system, which allows programmers to take advantage of more resources.

The appletviewer is not a fully functional Web browser; Navigator will generally be used when available. The appletviewer can be liberally configured. Some sites might consider using it in an intranet environment for delivering Java applets without a full Web page environment;

Action/Function	Appletviewer load URL	Appletviewer load file	Navigator load URL	Navigator load file	Java Application
Read/write file	Yes	Yes	No	No	Yes
Get file information	Yes	Yes	No	No	Yes
Delete file	No	Yes	No	No	Yes
Spawn another program	No	Yes	No	No	Yes
Load Java library	No	Yes	No	Yes	Yes
Connect to network port on local system	No	Yes	No	Yes	Yes
Connect to network port on downloaded host	No	Yes	No	Yes	Yes

Figure 6. Applet policies

Online Resources about Java

http://www.hursley.ibm.com/javainfo/	IBM's Java Center for Java Technology
http://www.alphaworks.ibm.com/	IBM's Alphaworks (has Win 3.1 code)
http://java.sun.com/	Sun's JavaSoft™ home page
http://java.sun.com/sfaq/	Sun's JavaSoft applet security
http://home.netscape.com/	Netscape's home page
http://www.javaworld.com/	JavaWorld™ online magazine
http://www.gamelan.com/	EarthWeb's Gamelan™, applets

Figure 7. Java resources

however, it should be used with care since it allows file access.

The appletviewer ability to read and write files is restricted to files or directories specified by the Access Control List (ACL) defined in your `~/.hotjava/properties` file. By default the ACL is set to null. This prevents any open security problems for first-time users because files cannot be read or written.

Netscape Navigator is even more restrictive, with little access to the filesystem allowed. Administrators have two configuration options with Java: ON or OFF. If Navigator is configured to OFF, then Java applets are not downloaded or executed. Nevertheless, the Web page containing the Java applets is still displayed.

The last column in the table shows the power and flexibility of full Java applications. As indicated earlier, Java is a full-featured programming environment. Programmers wishing to fully exploit the power of Java will probably implement full applications.

The first two levels of security, embedded in the Java language and the runtime environment, can be exercised by competent programming. But conversely, "sloppy" usage or malicious programming has the potential to bypass even the best security architecture. The next level of defense will be found in the configuration of the interpreter environment (refer to Figure 6). Turning Java access OFF will probably turn off your users. A more realistic site policy would specify suggested Web sites with no "blind" downloads from untrusted parties. Most sites would not recommend downloads from untrusted parties in a non-Java environment. The same common sense rules apply in this situation.

Conclusions and Resources

This article has described the Java technology and its major components. One primary advantage of Java is its platform-independent execution. Another is the delivery of dynamic executable content, allowing a click on a Web page to result in the download and execution of a Java applet. The flexibility of this model has several positive implications including automatic versioning control.

In this age of Internet innovation, we must all be conscious of security while new technology is being deployed. Security policies should be extended to encompass Java applets and their execution environments.

IBM has deployed Java on both AIX and OS/2. The technology is evolving almost daily. IBM has made available a Just-in-Time (JIT) compiler for AIX—intended to resolve the runtime interpreter performance. The JIT compiler can be linked to the AIX appletviewer for enhanced runtime performance up to 25 times faster than non-JIT execution.

IBM has also delivered a Windows 3.1 port of Java, now available via a Web page. If you are interested in some of these topics, you can pursue more information or code through the Web pages listed in Figure 7. IBM also sponsors a question and answer forum on the IBM Hursley Web page. USENET news groups are also a great source of information, including the comp.lang.java.security group.

Jeff Jilg, IBM Corporation, 11400 Burnet Road, Austin, TX 78758. Dr. Jilg is currently working in AIX System Architecture at IBM Austin on leading edge technology. His recent work areas include systems management, object-oriented tools, and the Internet. His current responsibilities include release architecture, Java on AIX, Internet integration, and the AIX Bonus Pack. His PhD in Computer Science from Texas A&M complements his master's degree in the same field from the University of Texas at El Paso.