Java: Status Report and Language Overview

John Caron

caron@ucar.edu

December 14, 1995

Introduction

Java is a new general purpose language from Sun Microsystems for programming on the World Wide Web. In the last three months, it has been accepted by all of the major Internet developers for adding interactive content to web browsers. The language itself has garnered much praise for its design.

The first part of this paper presents a non-technical summary of the context and issues surrounding Java, as well as its current market status. The second part presents an overview of the language design, and some experience in using it. The last part presents a brief description of related products and languages that are being positioned for web programming. The annotated Bibliography and list of Java related net resources should be useful to any programmers interested in Java.

I. Background/History/Status

The Internet

The Internet began in 1969 as an experimental network connecting Department of Defense research groups. The development of the TCP/IP protocols in 1982, and their incorporation into the UNIX BSD 4.2 operating system in 1983 gave the system a sound infrastructure within which to grow. In 1986, the National Science Foundation (NSF) took over funding and administering the network, which had grown to several hundred hosts machines with connections to similar networks in Europe and Japan. NSFnet began with a 56Kbps (bits per second) backbone connecting universities and government research centers. In 1989 the backbone was upgraded to T1 speeds (1.5 Mbps). In 1990, NSF agreed to a proposal by IBM, MCI, and the backbone provider Merit, for the commercialization and privatization of the Internet. In 1992 the backbone was upgraded to T3 speeds (45 Mbps).

In 1995, more than 4 million hosts are connected to the Internet. NSF no longer directly subsidizes it, but instead funds a separate high speed (155 Mbps) experimental network linking a small number of national supercomputing centers. The Internet infrastructure is now provided by the major long distance phone companies as a commercial operation. Competition among carriers is expected to regulate the costs and capacities of the Internet.

The number of hosts connected to the Internet and to other, private, wide area networks (e.g. AOL, CompuServe) continues to grow exponentially. There will be a further explosion of growth when high-speed connections to homes replace the current 20-30 Kbs modems possible over existing telephone lines. How that will be delivered (phone, cable TV, or even power lines) is still an open question, but it seems likely that increased demand substantially driven by Web usage will produce cost-effective answers within 1 or 2 years.

The World Wide Web

The World Wide Web (WWW or simply "the web") grew out of a project started in 1990 by Tim Bernier-Lee at the European Particle Physics Laboratory (CERN) for allowing access to scientific papers across the Internet. He proposed a new language for the rendering of hypertext documents called Hypertext Markup Language (HTML), and a protocol for handling these documents called Hypertext Transfer Protocol (HTTP).

Also part of the project was a "web browser", which can display HTML documents to the user, and can use HTTP to obtain the documents over the Internet. A hypertext document is one which contains references to other documents, and a browser can follow those references around the Internet as the user desires. These hypertext documents with their embedded references to other hypertext documents thus make the Internet into a "web" of information accessible to the user.

Web Browsers

In 1993 the National Center for Supercomputing Applications (NCSA) in Illinois released a robust and portable web browser for UNIX workstations called Mosaic. They also added a number of extensions to HTML, including the ability to display images on the same page as the text. Mosaic with its point-and-click graphical interface and ability to mix text and images took the Internet by storm. Many saw the commercial potential of these types of documents, and there has been a continual elaboration of multimedia effects such as video and sound. The success of web browsers has been based on their ability to present information more flexibly than printed media.

In 1994 Netscape Communications was formed, hiring many of the original Mosaic developers, and delivered state of the art web browsers for the UNIX, Mac and Windows operating systems. By January 1995 their browsers accounted for 80% of browser use. Netscape continues to add features to their browser to maintain their technological lead over commercial rivals, and continues to maintain their market lead as well.

Since the browser is the way that users access the web, it becomes the environment in which all web applications operate, i.e. the operating system. In that sense Netscape threatens Microsoft's dominant position in the PC market that has resulted from Microsoft's control of the PC operating system DOS/Windows.

Executable Content / Agents

HTML describes how to present what is essentially a static page. Web developers want to add interactivity and dynamic behavior. To do so requires sending not just passive content over the net, but active or "executable" content, that is software programs. The web browser is then thought of as the environment in which such a program would operate.

An "agent" program operates more broadly than from within a web browser, and the term can include any task-driven program like gathering information from a database, or purchasing airline tickets, in which the program runs largely independent of user direction. Agents are most useful on large distributed networks like the Internet, where there is no single, central store of information. They may have the ability to "migrate" to different machines in order to fulfill their task.

Security

Both agent programs and executable content are programs from a (possibly) untrusted source that want to execute on your machine. Whether to allow them to do so is a tradeoff between benefit you might get from this new technology, and the increased security risk of allowing access to your machine. Its one thing to have a program bug inadvertently crash your machine. Its another thing to allow access to a software "virus" that deliberately and maliciously tries to damage your system. If you are a business with a product catalog, allowing an agent to examine your catalog by executing on your machine might bring you increased sales, or it might allow someone to steal your customer mailing list.

A number of solutions to these concerns are being developed. Java as a language offers a design in which these concerns have at least been considered. Java as a system offers a solution that most people seen to be accepting as "good enough for now", with enough flexibility to be able to respond to security flaws as they are discovered.

Java

Java was originally designed in 1991 to be used to control consumer electronics such as VCRs, cellular phones, interactive TV, etc. Its designer was James Gosling, who previously had designed the well-regarded NeWS windowing system for Sun. It wasn’t until 1994 that Sun realized that Internet programming was a natural domain for Java, and not until March 1995 that it was shown to Netscape and other potential customers. It was publicly announced in May 1995. Netscape announced support for it at the same time, thus giving it immediate visibility and credibility. The Beta version of the Java Developers Kit (JDK) shipped in November of 1995, as did a beta version of Netscape’s Java-enabled browser.

Market Position

Currently, Sun has released a beta version of the Java Developer’s Kit (JDK) on Sun Solaris 2.3 and greater, and on Windows NT and Windows 95. Sun is also working on a Macintosh version, due first quarter of 1996, and on a version for SunOS 4.x. Sun will also produce the "Java Workbench" consisting of a WSYWIG editor and an integrated development system, and is working on integrating Java with their networked objects.

Independent ports have been made to SGI Irix 5, and are in progress to linux, Next, and Amiga. OSF is porting to HP-PA / HP-UX; Dec Alpha / Dec UNIX; MIPS R4000 / Sony News OS; Intel 486 / Novell Unixware; and MultiPentium / ATT UNIX. IBM has announced ports to AIX, OS/2, Windows 3.1, and is rumored to be adding it to Lotus Notes.

SGI and Macromedia will define new multimedia and 3D Java APIs for combining Java with VRML to produce interactive 3D rendering and multimedia tools. Sun, SGI and Macromedia have all mutually endorsed VRML, Java and Javascript as web authoring solutions.

Netscape and Sun will jointly develop Javascript, based on Netscape’s HTML scripting language LiveScript. Javascript will allow the developer to glue together HTML documents, Java applets, and Netscape plug-ins, capturing user events and allowing some level of interaction and feedback, like data validation of forms. This attempts to fulfill the needs of HTML programmers who are not willing/able to learn the complexities of Java, and competes with Microsoft's Visual Basic strategy.

Other major licensees include Spyglass (web browser), Oracle (database access), Metrowerks (Macintosh development tools), Sega (multi-player games via the Internet), Borland (PC development tools), Adobe (Acrobat documents and PageMill web authoring tool), and Sybase (database).

Microsoft in a surprise move, announced that it was licensing Java for inclusion into its browser, InfoExplorer.

Licensing developments have been moving like a whirlwind. Last week (Dec 4-8, 1995) the Javascript, VRML, IBM, Adobe, Sybase, and Microsoft announcements all occurred.

II. Java Language Overview

Java consists of:

- a language specification,
- a bytecode compiler,
- a virtual machine that interprets the bytecode at runtime,
- a set of class library APIs,
- implementations of the class libraries specific to the target machine,
- a runtime environment in which the interpreter, bytecode verifier, class loader, etc. run, also specific to the target machine, and
- other development tools such as a debugger, a disassembler and an appletviewer for testing applets outside of a web browser.

There is also a web browser written in Java called HotJava.

Sun has finished the language specification, and a draft is available at [Sun 95].

The library APIs are frozen, and consist of the following packages:

- java.lang is considered part of the language itself, and is automatically imported by the compiler. It creates classes for math functions, (limited) process control, a security manager, strings, threads, and wrapping primitive data types. The class System creates an interface to system facilities, such as stdin/stdout/stderr, environment variables, garbage collection, exec'ing processes, etc.
- java.awt: GUI/windowing toolkit.
- java.awt.images: image processing
- java.io: file input/output, pipes.
- java.net: sockets, Internet addresses, URLs.
- java.util: hashtable, bit sets, dates, random numbers, vectors, etc.

Sun has beta version implementations completed on Solaris 2.3 and greater, and on Windows NT and Windows 95. Sun is actively encouraging ports to other platforms, apparently sharing class implementation source in exchange for the port. The SGI port is reputably stable.

Language Design Overview

Java is an evolution of current object oriented programming languages, influenced by the design of Eiffel, Smalltalk, C++, Objective-C and Cedar/Mesa, among others.

The syntax of the language comes from C/C++. The way that Java is presented gives the impression that it was designed as an improvement to C++, removing features that were judged to be redundant, confusing, or not worth the complexity. There is no preprocessor, since preprocessing can make the source code unreadable and non-local (i.e. you have to read all the include files before knowing what your code actually does). Its object design obviates the need for typedefs and structs. Unions and pointers are removed as a way to break type safety. The goto statement is removed in favor of structured control. Multiple inheritance, operator overloading, and automatic coercion are removed as not worth the complexity and obfuscation they entail. Finally, there are no standalone functions, but only class methods, in keeping with a purer object design.

Java looks a lot like Smalltalk: single inheritance class hierarchy with "Object" at the root of all classes, reference semantics, garbage collection, dynamic binding, and an interpreter. However, it is statically typed, and avoids the extreme of Smalltalk "everything is an object" philosophy, having primitive types as well as reference types for efficiency. By deferring memory layout until runtime, a class definition can be changed without breaking its subclasses

An important addition to the language is the use of interfaces, like Objective-C "protocols", that allows some separation of data abstraction and inheritance. An interface defines an abstract type, independent of the class hierarchy. An implementation of the interface can be provided by a class without revealing details of the implementing class.

Another useful addition is packages, which group classes together and allow "friend" access among the classes in a package. Packages also provide namespace encapsulation, which prevents name collisions and prevents untrusted (remote) code from overriding local classes.

Built in language support for concurrency and multiple threads of execution reflect the developments in this area in the past few years, although only nonpreemptive multithreading is assumed to be implementable. Garbage collection is now considered more tractable than when C++ was designed. Exceptions are supported and look much like C++. Native code can be linked in for efficiency.

Some features considered missing by some: version control for classes, templates (parameterized types), first class functions, and dynamic method dispatch.

Security against untrusted code pervades both the language design and implementation. It is important to distinguish between two types of Java programs. Applications are stand-alone programs whose source is local and trusted. Applets are downloadable programs that operate within a web browser or some other Java application, and whose code must be verified and resource access limited.

Language Features

Types

Java has four data types: class types, interface types, array types, and primitive types. All variables hold either a value of a primitive type or a reference to a dynamically allocated object.

The primitive types are integer (8, 16, 32, and 64 bit, two's complement), float (32 and 64 bit IEEE 751), character (16 bit Unicode) and Boolean (1 bit). The format of these primitive types is fixed by the language definition and is not implementation specific.

The other three types are called reference types because they are implemented using references to dynamically allocated objects. A dynamically allocated object is either a class instance or an array instance. A variable of type interface holds a reference to a class instance which implements the interface.

Primitive types can be converted between each other, with the exception of the boolean type. There are no conversions possible between primitive types and reference types. Class types can be converted only to types that are superclasses of itself. Similarly, the run-time type of a variable must be a subtype of its compile-time type.

With these restrictions, Java implements strong compile-time type checking, but allows subtypes to replace types at runtime. Objects are tagged and can be queried at runtime to discover their runtime type.

Arrays

Arrays are dynamically created objects containing some number of variables called components. All components have the same type T, and the array has type T[ ]. T may be any type: class, interface, array, or primitive. As in C/C++, arrays are always one-dimensional, but multidimensional arrays can be created from arrays of arrays. An array of objects is actually an array of pointers to objects. An array whose components are primitive types, however, are allocated efficiently in contiguous memory as in C/C++.

The length of an array is not part of the array type, so that a variable may contain references to arrays of different lengths. However, once an array object is allocated, its length is never changed. All array accesses are checked at runtime that the index is within bounds, or an exception is thrown.

Classes

Each class type has exactly one parent class ("immediate superclass") which it extends by overriding parent methods and adding new methods or variables. Like Smalltalk, classes thus constitute a single inheritance hierarchy, with all classes having Object at their root.

Classes may be declared abstract, in which case they cannot be instantiated, but only subclassed. Classes may be declared final, in which case they cannot be subclassed.

There are no standalone functions: everything is a method in some class. Methods that are declared static within a class ("class methods") are the closest to a standalone function, because they do not belong to a class instance, but to the class itself. Class methods can only refer to other class methods or static class variables.

All classes are themselves instances of the class "Class", and a class object is instantiated when the class is loaded.

Interfaces

An interface creates a new abstract type that specifies a set of methods and named constants, but does not specify an implementation, like Objective C's protocols . Interfaces may optionally extend one or more other interfaces, meaning that it implicitly specifies all of the methods and constants of the interfaces that it extends. Thus interfaces constitute a multiple inheritance hierarchy similar to C++.

A class may be declared to implement one or more interfaces, meaning that any instance of the class implements all of the methods specified by the interface.

Interfaces thus allow code reuse between classes independent of the class inheritance hierarchy. It allows the creation of a specialized "view" of an object for export. Since the implementation of an interface must be provided by a single class, interfaces do not allow you to combine the methods of multiple classes into a single type, independent of the inheritance hierarchy

Program Structure

A compilation unit is a single file containing one or more classes or interfaces, only one of which is declared public. The source file must be called <Name>. Java, and the resulting bytecode is placed in the file <Name>.class, where <Name> is the name of the public class or interface.

One or more compilation units may be grouped together into a package. Classes within the same package have access to each other, similar to "friend" classes in C++. Packages are given globally unique names to facilitate finding them on the web. All compilation units of a package are kept in the same subdirectory, and the package name becomes the subdirectory name, starting from some specified list of root directories. Thus on UNIX, the package EDU.ucar.unidata.netcdf is found in a subdirectory EDU/ucar/unidata/netcdf off the root directories specified in the CLASSPATH environment variable.

By convention, an organization's Internet Domain name is used, in reverse order, for packages that are to be accessible across the web. The first component (e.g. EDU) is capitalized.

Information Hiding

There are 4 levels of access control : 1) public: accessible to anyone who imports it; 2) protected: accessible to anyone in the same package, and to any subclasses; 3) friendly: accessible to anyone in the same package; and 4) private: accessible only within the class body where it is declared.

A class may be declared public or private, and defaults to friendly.

A class variable or method may be declared public, protected, or private, and defaults to friendly.

Interfaces can be thought of as a way to hide some of the methods of a class.

Method overloading and overriding

Like C++, Java allows the same method name to be declared with different number or types of parameters. This is called method overloading, and the compiler can correctly match the calling signature with the correct method.

Methods in a class are accessible to its subclasses unless declared private. A method in a subclass with the same name and parameter signature as a method in one of its superclasses is said to override the superclass' method. Overridden methods can still be accessed by using the syntax "<name>.<method>", where <name> is the name of the superclass, and <method> is the overridden method name. Methods can be declared final to prevent overriding.

All non-static methods are dynamically bound, i.e. runtime lookup is used.

Control

Java looks much like C, having for, while and do statements. The goto statement is eliminated completely, but unconditional jumps can be done within iterations using break and continue statements. Labels can be used to break out of nested iterations.

Exceptions

A throw statement signals a run-time exception. "Normal" exceptions are status returns to be passed up through possibly many nested method calls, typically signaling some unusual but legal condition such as disk error or file access denial. "Abnormal" exceptions are subclasses of class Error or RuntimeException which signal an illegal operation or method call. Both kinds of exceptions can be caught and handled by an exception handler.

Methods must declare any normal exceptions that are thrown. This declaration becomes part of the method signature, and is thought of as part of the "contract" with the calling routine. Overriding methods may only throw exceptions already declared by the overridden method.

Concurrency

The standard package java.lang provides a Thread class to implement multiple threads of control similar to the Cedar/Mesa languages. The implementation is platform-specific, and should be assumed to be nonpreemptive, that is the thread must explicitly relinquish control.

A method can be declared synchronized, in which case a monitor lock is obtained on entry , and released on exit. Each class instance has its own monitor.

The I/O class library is thread-aware, thus allowing parallelism during I/O requests, which tend to be very slow.

Security

A number of language features provide security against untrusted code:

- object access is restricted by use of the public/private class attributes and the public/private/protected method attributes.
- strong typing ensures that the runtime type is a subclass of the compile time type.
- pointers are eliminated altogether; the untrusted code has no way to point itself to specific memory locations.
- packages provide a namespace encapsulation so that the runtime can distinguish between remote and local classes; local classes are always scanned first, so that a remote class cannot override a local class.

- Garbage collection eliminates the possibility of object aliasing by clever use of allocating and freeing memory.

See [BankJ 95] and [Yellin 95] for more details.

Garbage Collection

Objects are explicitly allocated, but not explicitly deallocated. A garbage collection algorithm runs in its own thread, detects that an object is not referenced anywhere, and reclaims that memory. If the class has a finalize() method declared, that method will be called before garbage collecting the object.

The finalizer may resurrect the object by storing a reference to it. Thus an object with a finalizer is not reclaimed until the garbage collector determines it is garbage again, after the finalizer is called. The finalizer is called only once, however. I'm not sure what problem this resurrection is solving.

Because of the asynchronous nature of the garbage collector, the finalizer is called at an indeterminate time after the object is actually garbage. There is some criticism in the news group about this.

Implementation

Java Virtual Machine

Java source is compiled into bytecodes that run on a virtual machine implemented by the Java interpreter and run-time system. The use of virtual machines trades efficiency for portability: It is significantly easier to implement a bytecode interpreter for a given CPU than it is to write a native code compiler. Thus the Java compiler that compiles Java source code into bytecodes needs only be written once. Each different CPU architecture and operating system then needs to implement a Java bytecode interpreter and provide an implementation of the runtime classes.

Currently, interpreted Java runs about 30 times slower than an equivalent C program.

There is some discussion about the JavaVM being a target for other languages. There is already an implementation of an Ada to javaVM compiler, and a discussion of having Guile use the JavaVM. There are ways that the JavaVM is limited in its functionality, that would make this currently impossible for an arbitrary program written in C (e.g. lack of pointers in Java) or Lisp (e.g. lack of closures in Java).

Memory Layout and Runtime Loading

Memory allocation and layout is not accessible to the programmer, but is handled entirely by the Java runtime. References to objects are not compiled into memory offsets from a base pointer as in C/C++, but remain symbolic until runtime, when they are checked by the bytecode verifier and then assigned (actually overwritten by) a memory offset. Thus the memory layout is done by the Java interpreter specific to the target machine. This runtime layout allows classes to add new methods and members without having to recompile subclasses that reference them (sometimes called the "fragile superclass" problem).

Security

Before the Java bytecode is executed, the bytecode verifier verifies that it is a legal program. It checks for forged pointers, access violations, that runtime types are compatible with compile time types, that all method calls' actual parameters are compatible with declared parameters, and that there are no stack under/over flows. The bytecode verifier is supposed to work on arbitrary bytecode sequences, not just compiler-generated ones.

The SecurityManager class is provided by the runtime to provide a flexible way to implement a security policy. Access to environment variables, local files, etc. can be controlled in this way.

It is critical that the class library implementations be correctly written, as well as the web browser. If I was trying to break Java security, I would try to induce unsuspecting sites to install a web browser or library implementation which I had altered to provide a security hole that I could exploit.

Native Code

Part of the long term strategy for improving performance is to provide "just in time" compilation of the bytecodes into native code. Since the bytecodes have been verified to be secure, the compiled native code can be presumed to be secure also. Sun has hinted it will be another 3-6 months before any native code compilers are available.

Methods can be declared native, and implemented in C or some other native language. Native code is linked into the Java runtime system by a call to System.loadLibrary( ). For security reasons, applets can only load native code that is local to the host machine.

Garbage Collection

The current garbage collection routine uses a conservative mark and sweep algorithm, with an optional compaction stage. It takes approximately 70ms on Java's default 1M heap, and runs in a background thread that abandons its work when a user event occurs in order to maintain responsiveness. Comments from Java's developers that the design can support incremental and generational garbage collection imply that these might be added in the future.

Use Report

To test Java, I implemented several small "applets" to work on the Netscape browser 2.0b3. I used a Sparc 10 running Solaris 2.4. I downloaded the Java Developers Kit (JDK) from http://java.sun.com. After uncompressing and untarring, the distribution was 10 Mbytes in size.

Since the first uses for Java are to add some pizzazz to web pages, I decided to find out how simple it was to manipulate some graphics. I ended up writing two applets, one to change the colors on a graphics image, the other to show a rotating graphics image.

To date, there is no good programmer documentation. Presumably all of Sun's effort is going into a book series from Addison-Wesley due out in March 1996. As an experienced C programmer, I already knew most of Java's syntax, which was a real advantage. By following the news group, and examining some of the many posted examples of source code, I learned much of the semantics. I also found reading the language specification useful, both straight through (once) and as a reference as questions came up.

Before coding in Java, I wrote code in C to test an algorithm for rotating arbitrary images through an arbitrary angle using brute force computation on every pixel. It is instructive to see how close the code is in this case. In the C code, I use pointer arithmetic for the 2D image array, but that maps very directly to using index arithmetic in a 1D Java array. The image is assumed to be stored in an integer array in memory.

The C code:

```
#define PI 3.1415
static void Rotate(int *original, int nrows, int ncols, double rotate, int *rotated)
    {
        int *colorp = original;
        int x,y,ok;

        printf("rotate = %f deg\n", rotate);
        rotate *= PI/180.0;

        /* set the target array to original, since not all pixels get covered */
        memcpy( rotated, original, nrows * ncols * sizeof(int));

        for (y=0;y<nrows;y++)
            for (x=0;x<ncols;x++)
                {
                double xp = (double) x - .5 * (ncols-1);
                double yp = (double) y - .5 * (nrows-1);

                double theta = atan2( yp, xp) - rotate;
                double r = sqrt(xp*xp + yp*yp);
                double xnewp = r * sin(theta);
                double ynewp = r * cos(theta);

                int xnew = (int) (xnewp + .5*(ncols-1));
                int ynew = (int) (ynewp + .5*(nrows-1));

                ok = ((xnew > 0) && (xnew < ncols) && (ynew > 0) && (ynew < nrows));
                if (ok)
                    *(rotated + ynew * ncols + xnew) = *colorp;
                colorp++;
                }
    }
```

The Java code:

```
public class Rotate extends java.applet.Applet implements Runnable {
    int nrows;
    int ncols;
    double rotate = 0.0;
    double rotate_inc = 15.0;
    int original[];
    int rotated[];
    Image img;
        ...

    void rotate()
        {
        rotate += rotate_inc;
        System.out.println("rotate through angle of " + rotate + "deg");
        double rot_rad = rotate * Math.PI /180.0;
```

```
        // start from original, since not all pixels get set
        for (int i=0; i<nrows*ncols; i++)
            rotated[i] = original[i];

        // rotate through an angle of "rotate"
        for (int y=0;y<nrows;y++)
            for (int x=0;x<ncols;x++)
                {
                double xp = (double) (x - .5 * (ncols-1));
                double yp = (double) (y - .5 * (nrows-1));

                double theta = Math.atan2( yp, xp) - rot_rad;
                double r = Math.sqrt(xp*xp + yp*yp);
                double xnewp = r * Math.sin(theta);
                double ynewp = r * Math.cos(theta);

                int xnew = (int) (xnewp + .5*(ncols-1));
                int ynew = (int) (ynewp + .5*(nrows-1));

                boolean ok = ((xnew>0)&&(xnew<ncols)&&(ynew > 0)&&(ynew <nrows));
                if (ok)
                    rotated[ynew*ncols + xnew] = original[y*ncols + x];
                }
    }
```

In order to understand what an applet did, I took an example applet from the net and tried to follow its methods. It turns out that the applet class hierarchy goes Applet < Panel < Container < Component < Object (where< means "is a subclass of"). Each of these classes adds new methods. An applet as a subclass of all of them thus has over 100 methods that could be called. The lack of documentation completely stopped me at this point. I printed out the "official" API, which consisted of postscript files containing each class and their public fields and methods. The descriptions attached to them usually added nothing, for example:

public void setInsets(int n, int e, int s, int w);

Set the insets of this container.

Finally I printed out the source itself. This turned out to be the only way of proceeding to learn the Java library API. The source is well written, has some embedded comments, and, as the saying goes, doesn’t lie. I also constructed a list of Applet ’s superclasses with a list of all their methods. I could thus tell, when an applet made a method call, which superclass it was calling, and then could look at the source to see what it actually did. Studying the source was also an excellent way to learn the language itself.

I implemented a few simple applets, trying them out in both the appletviewer and in Netscape 2.0b3. Often the applets would work in the appletviewer but not Netscape. One problem encountered by every single beginning Java programmer (to judge by the number of comp.lang.java posts) is forgetting to set the CLASSPATH environment variable to the top of the source directory tree. Another problem is that Netscape wont read local files, but will read files specified by a URL. This restriction is presumably due to security. Another source of confusion is that after changing and recompiling an applet, clicking Netscape’s "reload" button will not reload the applet for you (you must use Shift-Reload, an undocumented feature).

Next I started to learn about displaying and manipulating images. It was quite easy to read in and display GIF and JPEG images (at least specified by a URL). I had the idea that simply moving the image bytes around and redisplaying would be an efficient and flexible way to proceed. As I followed the source code to find out how to do that, I quickly discovered the difference between the library API and its implementation. It turns out that most of the real work is done not by class instances to which the source is available, but by "peer" objects that are the platform-specific implementations. For example the setCursor method in class Frame simple passes your method call onto its peer:

```
/*** Set the cursor image*/
public void setCursor(Image img) {
        FramePeer peer = (FramePeer)this.peer;
        if (peer != null) {
             peer.setCursor(img);
            }
}
```

FramePeer is an interface implemented, in my case, by the sun.awt package, for which only the compiled bytecode is shipped. It finally dawned on me that this was the only way that you could write platform independent windowing/graphics: define a layer that constitutes the API, and implement it with separate, platform dependent classes. Sun had chosen not to "give away" its X/Motif implementation by releasing the source code.

The image manipulation methods available in the API are provided by the ImageProducer, ImageConsumer, and ImageObserver classes. These allow me to create images based on filtering existing images, or to create an image from an array of RGB values in memory. However, the actual representation of the image, and the way that it is rendered, is opaque. Thus, rather than moving some bytes around and redisplaying, I had to create a new image each time I wanted to change it. How efficient this method is is unknown, so I don't know yet how far one can push image warping or other advanced techniques.

In retrospect, given that Java has to support different graphics environments, the opacity of the image data is a good thing. My applet worked fine for both GIF and JPEG images, and presumably will work for other image formats as they are added. I ended up making the image name, number of rotations, and rotation speed parameters that can be set in the html file, so that the class can be used to rotate any image. (Don't bother with large images until the rotating algorithm gets more clever though).

In summary, I found the Java class libraries complex and somewhat daunting. You have to look in many different places in the source to find out what a method call really does, and often you find the call disappears into the proprietary classes with source unavailable. On the other hand, the availability of a platform-independent GUI API is of tremendous value to application programmers. Whether this particular GUI API design will provide the flexibility and power needed has not been answered yet. There is some grumbling about it on the net as the "weak link" in Java's class libraries, and comparing it unfavorably to Tk. The problem is that implementers are busy writing their platform-specific classes to this particular API, so design improvements will take some time and not be painless.

III. Other Languages

There are plenty of other candidates for providing executable web content and agent programs. Some of the more prominent are listed here, see [Connolly 95] for more details, and the Java Jive site (see "Net Resources", below) for other candidates.

Many of the languages here are interpreted, and it seems like every interpreted language ever designed has been considered a candidate for web programming. An interpreted language has an immediate advantage in providing security, since a program executes within the interpreter, which can deny access to system resources. But the real issue is what resources are needed to accomplish a given task (e.g. allowing database access to information gathering agents), and how to provide just that access without creating any holes that a clever program can use.

See [Connelly 95b] for other criterion for evaluating mobile code systems.

Blackbird / Visual Basic

Blackbird is the internal name for Microsoft's "online publishing" system to allow users to create fully interactive multimedia applications. It includes a server to be bundled with Windows NT, a free web browser, and a hypertext authoring tool. It allows OLE objects to be accessed over the net. The system will support HTML and SGML documents, as well as proprietary data and document formats. Blackbird will use Visual Basic as the "glue" language for the various components. Visual Basic is interpreted, extensible, and downloadable, but has no special security provisions. Microsoft's solutions in general have inadequate security provisions. Currently Blackbird in in beta release.

Microsoft strategy has been likened to competing with the Internet, rather than with Java. Certainly Microsoft is ambitious enough for that, but has recently backed off from that position by licensing Java and repositioning their private WAN (MN) as an Internet gateway rather than an alternative.

Internet developers in particular are wary of proprietary solutions, and are generally hostile to Microsoft. Many hoped that Microsoft would try to offer VB as an alternative to Java, so that they would lose ground in the web-related markets. Clearly Microsoft has decided to target Netscape to compete with, and their recent Internet strategy announcements were well received, causing Netscape stock to lose 28 points.

Obliq

Obliq [Cardelli 94] is a lexically-scoped untyped interpreted language for distributed computation, designed by Luca Cardelli at DEC, and implemented using Modula 3's network objects. Obliq follows the language Self in providing objects but no classes; instead of subclassing, an object is cloned and its methods are overridden.

In its current implementation, objects are local to the site they were created, but references to them can move around the network. In particular, actual computations that can move around the network take the form of method closures, i.e. source text plus its free identifiers bound to objects at the originating sites. This is the meaning of "lexically scoped": the binding of free identifiers is done explicitly in the closure's text, so that there is no ambiguity about the free identifiers. These closures in fact constitute agents: executable code that can move around the network. For security, these agents have access only to the free identifiers bound to the originating site, and any parameters passed in by the host machine.

Obliq has been available to the research community since 1993. Neither it nor Modula-3 has had any significant exposure in the commercial world. Obliq has been integrated with the FormsVBT toolkit in X Windows, and is being ported to Win95/NT.

Omniware

Like Java, Omniware [Colusa 95] defines a virtual machine, OmniVM, and uses an interpreter on the host machine to execute mobile code. Unlike Java, OmniVM supports all of the language constructs of C/C++ such as pointers, and thus can be the compiler target of C/C++. Omniware relies on a technique called Software Fault Isolation [Wahbe 93] to provide security. SFI adds instructions to check at runtime that addresses are within legal address space.

Omniware's advantage is that existing languages can be compiled into OmniVM code. Colusa Software would like to position OmniVM to be the "universal substrate" of mobile net programs, but it must convince hardware vendors and web browsers to implement the VM in their platform. Portable access to system resources remains a big need of web programmers, apparently unanswered by Omniware.

Python

Python is an interpreted OOPL designed after the ABC and Modula-3 languages, by Guido Van Rossum at CWI in Amsterdam. The type system looks like Smalltalk, except with multiple inheritance and some primitive types. It has a strong module design like Modula-3. Several GUIs have been glued into python, including X/Motif, Mac Toolbox, MS Windows MFC, and Tk. Python is used as an embedded/extension language in hypermedia and for perl-like scripts.Grail is a web browser written in Python using Tk.

Tcl/Tk

Tcl is a popular public domain interpreted language originally designed as a "glue" language to control various applications and allow them to communicate. It has attracted a sizable community of UNIX developers due to its being free even for commercial development, and because of the Tk library built on top of Tcl that provides a convenient way to build Motif-based GUIs. Tcl/Tk is used for rapid prototyping and GUI development, and as a scripting and glue language. Its lack of structure and slowness makes it unsuitable for large programming projects, but it does provide clean interfaces to native C code.

Safe-Tcl is an extension of Tcl for running agents. Safe-Tcl uses a "safe" interpreter for running agents, where access to system resources is controlled. TkWWW is a free web browser built with Safe-Tcl.

Many are wondering about the possibility of using Tk as the GUI building part of Java, particularly since John Ousterhaut is now working for Sun. I would guess that there will be a Tk/Java marriage, but it's unclear whether that will come from Sun, or if it will be integrated into the JDK.

Telescript

Telescript is an "agent operating system" from General Magic. Telescript lets users send executable programs in the form of agents through the network. The system is proprietary and technical details are sketchy. Telescript is an interpreted, object-oriented language, with a class hierarchy based partly on Smalltalk. It was originally targeted at the Personal Digital Assistant (PDA) market, and there are a number of commercial applications built using Telescript, including Sony's Magic Link, Motorola's Envoy, and ATT PersonaLink. Its very oriented to large corporate developers, and has pretty much missed the boat attracting small independent developers.

Virtual Reality Modeling Language (VMRL)

Version 1.0 of VRML is based on a subset of the Open Inventor ASCII file format from Silicon Graphics, Inc., with extensions to support networking. A VRML file describes a "scene" of 3D geometrical objects in a way that makes perspective-view rendering straightforward. From the Version 1.0 specification [Bell 1995]:

The Virtual Reality Modeling Language is a language for describing multi-participant interactive simulations -- virtual worlds networked via the Internet and hyperlinked with the World Wide Web...It is the intention on its designers that VRML become the standard language for interactive simulation within the World Wide Web.

Other than hyperlinking features using standard URLs, this first version of VRML does not support interactive behavior. Version 2, due by the end of 1995, promises to make a first cut at "support for arbitrary interactive behaviors". The VRML Architecture Group has provisionally chosen Java as the reference language for writing VRML applications. This differs from being a "standard" language in that VRML will remain open to all languages.

SGI and Macromedia have announced that they will develop VRML APIs for Java. SGI is the acknowledged technical and market leader in 3D technologies, and this area promises to be hot.

Conclusion

In the last two months, Java has changed from being an interesting new possibility, to being the accepted next generation web programming technology. There is still a chance that it will stumble, but most are convinced that the language design is good and that the security is adequate. It has enough resources from heavyweight and creative companies behind it that it can no longer outright fail. The only company powerful enough to offer an alternative, Microsoft, has decided to join instead of fight. Within the next year Java-enabled browsers will become the norm rather than the exception.

The reasons for its success I think are the following:

1) It improves upon C++, simplifying it, and at the same time adding new features to improve reliability and clarity. Many programmers (myself included), have avoided committing to C++ due to its complexity, despite recognizing the need for OO techniques. Java keeps the syntax of C/C++, thus making it immediately accessible to a large proportion of programmers. Obliq, in building on top of Modula-3, did not have this advantage.

2) Its specifications are open, it’s freely available on the Internet, and it’s portable to different platforms, thus fitting in well with the culture of the Internet community of programmers and academic institutions. Telescript clearly failed on this score, and Microsoft is perceived as an enemy of open solutions. Sun’s decision to provide Windows and Mac ports provided proof of Sun’s commitment to openness.

3) By providing a portable GUI/windowing API, Java answered such a huge need of application programmers, that they are willing to overlook its weaknesses for now. Delivering a working implementation of the awt library was a major contribution to Java’s rapid capture of mind-share.

4) Java is currently targeted to a well-defined application domain with well-funded companies intensely

interested in pushing into new technologies. Multimedia web applications will drive a new round of consumer interest in computer applications. Java offers an adequate technical solution to the problems of mobile code. In other languages (e.g. VB, Tcl), these concerns came after the language had been designed and established in other domains.

5) Early acceptance by Netscape was a key factor in its broad acceptance, as Netscape was itself riding a wave of media interest and leapfrogging competing products, and establishing itself as the operating environment for web applications.

6) Sun Microsystem's history as a technological leader (UNIX, NFS, etc.), and ability to carry out the implementation assured other companies that committing their resources to Java wouldn't be wasted. Some of the competing programming languages may have advantages over Java technically, but are from small companies (Omniware), or even one-person development efforts (Python, Tcl in some ways).

There is certainly much more to be told about the direction Java and web programming will take in the future. Implementations still have to be completed, browsers delivered, security tested, the GUI library matured. The Java class libraries are complex and Java programming will not be done by casual users. However Javascript and point-and-click application builders will provide higher level access to Java capabilities. Indeed, web programming tools may become the predominant way non-programmers will create documents, and web browsers may eventually be the glue for word processors, databases, graphics, etc. Thus Java may find a role anywhere that documents, especially with multimedia content, need to be made accessible across platforms, such as within large corporations.

It's possible Microsoft will end up dominating the Internet as well as the desktop, and it's still hard to say if Sun itself will derive much long-term revenue from Java. Certainly VRML and multimedia APIs from SGI and Macromedia will be very interesting. Fast home Internet connections will become affordable, and the demand for faster CPUs, audio, video and 3D graphics hardware will increase.

The web will be more interactive, animated and useful, as well as more commercial, distracting and frivolous. Ultimately the decentralized, international, and active nature of the web should provide us with a richer medium in which to grow our culture.

Annotated Bibliography

[BankD 95] David Bank. "The Java Saga". Wired. Dec. 1995. History of Java development.

[BankJ 95] Joseph Bank. Java Security. http:www-swiss.ai.mit.edu/~jbank/javapaper/javapaper.htmlAn online paper concerning security in Java. Not definitive, but a start at defining the technical issues.

[Bell 95] Gavin Bell, Anthony Parisi, Mark Pesce. The Virtual Reality Modeling Language. Version 1.0 Specification. Third Draft, May 26, 1995. http://vrml.wired.com/vrml.tech/vrml10-3.html Definition of VRML, version 1.0. Also a brief history.

[Byte 95] Byte. "The Internet and Beyond", July 1995, vol 20 number 7, p 69-80. History of the Internet.

[Cardelli 94] Luca Cardelli. Obliq, A Language with Distributed Scope. Digital Equipment Corporation, June 1994. http://www.research.digital.com/SRC/. Defining document for Obliq. Cardelli is the leading programming language type theoretician, and his work is very influential, even if not commercial.

[Colusa 95] Colusa Software. Omniware: a Universal Substrate for Mobile Code. White paper, Colusa Software. 1995. http://www.colusa.com. An alternative approach to agent security.

[Connolly 95] Daniel Connolly. Mobile code. http://www.w3.org/hypertext/WWW/MobileCode An excellent summary and evaluation of other agent programming languages from Daniel Connolly of the World Wide Web Consortium.

[Connolly 95b] Daniel Connolly. Design Issues in HyperMedia Development. http://www.w3.org/hypertext/WWW/DesignIssues/HyperMediaDev.html. Criterion for evaluating mobile code systems.

[Gosling 95] James Gosling and Henry McGilton. The Java Language Environment, A White Paper. Sun Microsystems. May 1995. http://java.sun.com/whitePaper/java-whitepaper-1.html Marketing paper with lots of buzzwords, but useful summary.

[Gosling 95a] James Gosling . Java Intermediate Bytecodes. ACM SIGplan Notices, Mar 1995, vol.30 number 3. Defines the Java Virtual Machine.

[Guile] http://java.sun.com/archives/java-interest/0122.html. A long email post describing the status of the gnu project Guile/GROW as of May 1995.

[van Hoff 95] Arthur van Hoff . "Java and Internet Programming". Dr. Dobbs Journal #233. August 1995. pp56-61. http://www.dobbs.com/issues/j508a.htm. Overview of Java by one of its developers.

[Lawler 95] Andrew Lawler. "NSF hands over the Internet". Science, 17 March 1995. vol 267 p 1584-5. Brief summary of the privatization of the Internet.

[Magid 95] Jonathan Magid, R. Douglas Mathews and Paul Jones. The Web Server Book..Ventana Press, 1995. First chapter has good summary of web history. Also an excellent intermediate technical reference.

[Ousterhout 94] John Ousterhout. Tcl and the Tk Toolkit. Addison-Wesley 1994. The reference book on Tcl/Tk, written by its designer and implementor.

[Snyder 86] Alan Snyder. Encapsulation and Inheritance in Object Oriented Programming Languages. OOPLSLA 1986 Proceedings. An important criticism of the confusion of data abstraction and class inheritance in OOPL design.

[Sun 95] Sun Microsystems. The Java Language Specification.. Beta 1.0 version, October 1995. http://java.sun.com/1.0alpha2/doc/javaspec/javaspec_1.html The current draft of the language specification.

[SunW 95] "Java, the Inside Story". Sunworld. July 1995. http:/www.sun.com:80/sunworldonline/swolo-07-1995/swol-07-java.html. Fluffy overview of Java history.

[Telescript] http://www.genmagic.com/MagicCapDocs/Concepts/Telescript.html. General Magic. On line docs about Telescript. Not very theoretical, more of a "how to get started coding" tutorial.

[McCool 95] Michael McCool. MetaMedia Project. http://www.cgl.uwaterloo.ca/~mmccool/Meta/index.htmlAcademic project investigating the dynamic definition of syntax and semantics of executable content languages.

[Valdez 95] Ray Valdez. "Internet gets a Java Buzz". Dr. Dobbs Journal Developer's Update, Aug 1995. http://www.dobbs.com/dddu/u508-lea.htm. Good early reporting on Java from an outsider who can put it in perspective.

[Wahbe 93] Wahbe, Lucco, Anderson, Graham. Efficient Software-Based Fault Isolation. Fourteenth ACM Symposium on Operating System Principles. Dec. 1993. This is what Omniware is based on.

[Wayner 95] Peter Wayner. "Free Agents". Byte March 1995 pp 105-114. Dated overview of agent technology.

[Yellin 95] Frank Yellin. Low Level Security in Java. Paper to the WWW4 Conference, Dec 1995. http://www.w3.org/pub/Conferences/WWW4/Papers/197/40.html. Sun's explanation of their security implementation.

Java Resources on the Net

Many of these sites have lots of pointers to other Java-related sites and information.

3D visualization. http://www.dimensionx.com Java meets VRML. http://www.cs.brown.edu/research/graphics/research/papers/vrmlbehaviors.html. A paper on VRML, talks about Java as one way to add behavior. http://sdsc.edu/SDSC/Partners/vrml/doc.html. Start here if you want to know about VRML itself.

Grammer. http://www.cc.gatech.edu/gvu/people/Faculty/hudson/java_cup/home.html. Generates LALR parsers from simple specifications, like YACC, except written in Java and produces Java code. A grammer for Java itself can be found at ftp://ftp.cc.gatech.edu/people/hudson/java_grammer.

FAQ: http://sunsite.unc.edu/javafaq. Frequently Asked Questions. Also links to a tutorial.

Gamelan . http://www.gamelan.com. A directory and registry of Java resources. Can register your own contribution. Very useful.

J* Notes.** http://www.io.org/~mentor/J___Notes.htmlWeekly summary of the comp.lang.java news group. Also has latest news, jobs, and Java related pointers. Better than reading 200+ postings per day.

Java Home Page. http://java.sun.com. Sun's home page for Java. Can get JDK, Sun documentation here.

Java Jive. http://www.dobbs.com/dddu/java.html. A Java information page put together by an editor at Dr. Dobbs Journal. Has many pointers to other Java information. Not updated recently.

Mail Archives. http://java.sun.com/archives/java-interest. Archives of the Java interest mail group. Covers the period from July 1995 - September 1995, when the mail group became a usenet news group.

MIDI. http://www.mit.edu:8001/afs/athena/user/j/b/jbreiden/mz/javamidi.html. Java meets MIDI.

Programming http://users.aol.com/thingtone/workshop/index.htm Thingtone's Programming Workbench: Advanced tutorial with annotated source code.

Usenet news group. comp.lang.java. Also a searchable news archive is available at http://kayak.npac.syr.edu. A summary of the news group is available at J*** (above).

World Wide Web Consortium. http://www.w3.org /hypertext/WWW/Consortium. Industry consortium for web-related research, run by MIT. Lots of useful information.

World Wide Web Virtual Library http://www.acm.org/~ops/java.htmLots of pointers to more Java-related resources.