# When to Combine ActiveX and Java

Both ActiveX and Java are viable technologies. Both can be used to create Internet applets for dynamic World Wide Web pages. Both produce code that can be located in centrally stored Web pages, offering centralized control reminiscent of the mainframe model. And both support object oriented programming techniques that produce re-usable code, potentially reducing the speed and cost of software development.

Microsoft will support both ActiveX and Java on future platforms, giving developers the option to develop solutions exclusively with either technology, or arrive at some combination of the two.

The goal of this article is to help IS managers identify the decision points for choosing one technology over the other, and for identifying those situations where it may be desirable or necessary to combine the two.

The article is divided into three sections:

- Developing with ActiveX
- Developing with Java
- When to consider combining ActiveX and Java

It's important to note that both of these technologies are new and evolving rapidly. While the recommendations made below are valid today, the direction of either technology could radically change within the next six months as new technology and tools are released.

## I. Developing with ActiveX

ActiveX springs from Microsoft's traditional Windows development platform. ActiveX code is developed using a Windows-based Integrated Development Environment (IDE)—such as Microsoft Visual C++ and Borland C++—and uses Microsoft's Component Object Model (COM) to describes how running program components communicate with each other.

ActiveX brings three key benefits to developers:

*Relatively mature development environment.* Developers can leverage their experience writing 32-bit OCX controls and Win32 applications to writing ActiveX controls. Developers also have a wide variety of sophisticated development environments to choose from, such as Microsoft's Visual C++ and Borland's Delphi.

*Ability to integrate with existing Windows components.* ActiveX components can access any COM interfaces built into existing Windows applications and OCX controls. For example, an ActiveX control can integrate with Microsoft Office applications through Object Linking and Embedding (OLE) COM interfaces.

*Full access to underlying operating system services.* Since each ActiveX component must be rewritten for each target operating system platform, an ActiveX component can use that platform's specific API's to access all of the client machine's functions and services. For example, an ActiveX multimedia control developed for the 32-bit Windows platform could access DirectX APIs for high speed graphics and sound.

ActiveX also has four weaknesses that developers must overcome or accept:

*Platform dependent code.* An ActiveX component only functions on the particular operating system platform for which it was created. ActiveX compels developers to author, test, and maintain separate versions of an ActiveX control for each target platform. This requirement makes ActiveX an

expensive choice for controls on the WWW. For example, an ActiveX control vendor who targets Windows NT platforms must still create, test, and maintain separate executables for the versions of NT that run on x86, Alpha, PowerPC, and MIPS processors. [This isn't correct. You can write ActiveX controls in Java which are single, cross platform binaries.]

*Incomplete platform availability.* ActiveX is currently available only on Windows 95 and x86 Windows NT. RISC NT and Macintosh versions are expected within the next six to nine months, but availability for other platforms is uncertain.[We have announced and have working versions of ActiveX support on Unix as well. In fact, COM/ActiveX support will be available on Solaris in the future.]

*Inability to prevent Web page applets from causing harm.* Once downloaded onto a client, an ActiveX control can conceivably wreak havoc on the client system and the corporate network, While Microsoft's trust security model verifies the source of an ActiveX executable before running it (see our June 996 issue, "Windows Trust Verification Service," page 11), it does not prevent the control from doing anything harmful, such as accessing private files or installing a virus.[The fact is that the ActiveX control authentication actually verifies the source of software more surely than shrink wrapped software in a store. The two security approaches are meant to solve different problems. There will always be a need for teal, fully enabled/trusted software, and the control security model allows this. The sandbox model addresses the need for an environment where anyone can contribute software to the world which is limited in scope due to safety constraints, but still worthwhile.]

*Weak client system maintenance.* In Microsoft's current ActiveX implementation, a control may download a supporting Dynamic Link Library (DLL), automatically overwriting a previous version. This uncontrolled substitution could cause the disruption of an application that relied on the previous DLL version. Also, downloaded controls can accumulate on the client, taking up significant space in the machine's hard disk and system registry. (*Ed. Note:* This is probably a short term weakness although Microsoft hasn't released specific plans for automated removal of unused controls.) [This isn't true. The versioning model which we've even enhanced Java with (since there isn't one), defines rules that allow upgrading of controls while preventing replacement of incompatible versions. This is actually one of the very cool strengths of ActiveX. The second weakness is true and is being addressed as you mentioned.]

When to develop with ActiveX and COM. The wide availability of Windows-based development tools, the ability to leverage existing Windows programming expertise, and the capacity to interact with existing code via COM makes ActiveX best suited for Windows-based client development. ActiveX development works best in circumstances where the target client platforms are exclusively Windows 95 and Windows NT.

## II. Developing with Java

Java code is developed through the combined use of the Java programming language, IDE for Java, and a Java Virtual Machine (JVM). The JVM must be implemented on each target platform that will receive the Java code (see "A Java Tutorial," page XX).

Java brings four benefits to developers:.

*Security and robustness.* Through byte-code verification and a "sandbox" architecture, Java prevents applets from performing any mischief on the client or corporate network. [Actually, Java does not at all address denial of service attacks as noted by Princeton (creating too many threads, using too much memory, etc). In fact, it is not difficult to write a Java applet that makes your machine unusable or crash in any of todays Java environments. What Java does address very well (in versions with solid security support) is the possibility of applets destroying or accessing data outside of the sandbox as well as installing viruses on the local machine. Since we've also added signing and security support, Java in our model supports signed/trusted applets, class library versioning, and extended the capabilities of trusted Java applets.]

*Platform independence.* Java's cross-platform byte code and abstraction of underlying system services such as graphics and threads allows the same Java executable to run on many different operating

system/CPU platforms.[This is not a point of difference because ActiveX and Java are not mutually exclusive. To be accurate here, it is true that Java is, by definition, cross platform, but with controls, that is a function of the implementation language. Java and VB controls are both cross platform.]

*Well suited for large scale development projects.* Java's object orientation encourages developers to divide code into pieces which can be re-used in other programs. Also, Java's automatic garbage collection frees developers from having to explicitly release resources such as memory and window handles, decreasing program complexity and increasing program reliability.[Also very well suited to writing ActiveX controls.]

*Support for centralized software distribution.* Java applets are downloaded each time they are needed, in contrast to ActiveX controls which become permanently installed on the client when downloaded. This allows corporations to centralize software in a fashion reminiscent of the mainframe model and thereby minimize client support costs.[Using the ActiveX download model enhancements, this promise becomes a reality. Downloading every time something is used is redundant and slow. There's now a whole company base on trying to do what ActiveX download support provides for Java without the versioning and authentication. If you look more closely, the ActiveX model actually provides vastly more powerful centralized maintenance.]

Java has four weaknesses that developers must overcome or accept:

*Slow runtime.* Since Java Byte-code must be interpreted by the runtime environment, a Java applet can run an order of magnitude slower than an ActiveX control. This drawback may be mitigated or removed entirely by JIT compilers promised in the future.[This problem is pretty much removed as soon as we RTM next week. Of course the runtime itself will always have to be loaded which is more overhead during instantiation and more of a memory hit. That's what you pay for cross platform.]

*Limited set of available system services.* The current set of Java APIs do not abstract all the system services a programmer might need. For example, video recording and playback services are not currently available through the Java API. To gain access to such services, a developer must create a platform dependent section of code that interfaces directly with the underlying operating system. Such a solution sacrifices Java's platform independence for the sake of increased program functionality. (Sun is currently extending the system services supported by the Java API, including Multimedia, data base access, and commerce APIs.)[Many of these APIs will need more security (signing and authentication) than Java currently provides in order to be truly useful.]

*Immature Development Environments.* Most development environments for Java are still in Beta. The dearth of sophisticated development tools limits the complexity that can be built into Java applications.

*Platform inconsistencies.* The Java Virtual Machine may be inconsistent from platform to platform. For example, different vendors' versions of the JVM can vary in how they implement the creation of a security sandbox. This variance can cause program failures when two or more Java Applets must interact. Inconsistency between runtime environments is a pivotal issue for Java's success since it threatens platform independence.

When to develop with Java. Java works best when creating dynamic content for the Internet and for heterogeneous corporate intranets. [This is an area where signing opens up a new world. In a corporate intranet, MIS departments can use a corporate signature to enable cool, internal or trusted Java applets/applications to merge local databases with the corporate database, access and update files on employee's systems, and other maintenance tasks.] Java is already is supported across a variety of client operating systems and CPU architectures by most vendors' browsers (either in final or beta form). Also, Java's security architecture makes it safe for corporate IS to allow users to cruise Web sites containing Java applets.

Java is also well suited for creating new corporate applications which don't need to be tightly integrated with existing Windows programs. [Without signing, there is currently a problem in deploying applications which maintain user's systems in a corporate environment without completely restricting access to the external internet. This is a serious flaw which we believe we have addressed.]For example,

Java would be the development environment of choice to build a real estate property listing viewer for agents in the field who might access that viewer using Windows, Macintosh, or a variety of other types of client machines.

## III. When to Consider Combining ActiveX and Java

Combining ActiveX and Java requires development in both a traditional Windows development environment and a Java IDE. It also requires that the target client's operating system and JVM are COM-compliant.

Combining ActiveX and Java brings one main benefit to developers:

*Adds Java to Windows development tools.* Organizations with large investments in Windows can use Java to develop new applications and integrate those applications with Windows clients.

Combining ActiveX and Java has three weaknesses that developers must overcome or accept:

*May introduce platform dependencies.* In most cases solutions that combine ActiveX and Java will inherit ActiveX's platform dependency (and forfeit Java's cross-platform advantage).

*May introduce security & robustness concerns.* Combining the two technologies introduces ActiveX's security & robustness considerations that wouldn't be present when developing exclusively with Java.[We have only enhanced Java security with our support.]

*Vendor dependent.* Solutions that rely on the integration of these two complex technologies rely heavily on Microsoft and others vendors to provide the technological infrastructure that makes this integration work. Much of this integration technology is still nascent; Microsoft, for one vendor, is still working on a number of critical technical details.[This is somewhat hard to understand.]

When to combine ActiveX and Java. As a general rule[,] developers prefer to keep solutions within the framework of a single technology[do you mean that developers don't mix languages???], combining multiple technologies only when absolutely necessary. Developers should consider combining these two technologies when they must integrate existing Windows-based code with new Java authored components/solutions.

There are five specific situations where combining ActiveX and Java technologies might be desirable or necessary. Developers should consider combining the two technologies when:

- Authoring a Java Applet to run inside a COM-based browser
- Creating a Java Application to be linked or embedded within an OLE container
- Using an existing ActiveX control from within a Java Application
- Constructing a Java Application capable of launching a Windows program
- Creating Java middleware that performs services on behalf of a Windows front-end

[Integrating mixed language or C++ development with Java]

[Accessing new ActiveX technologies such as DirectX (sound, 2D, 3D, MM)]

Each of these ~~five~~ situations are outlined below, including the consequences to development, code portability, and security & robustness that result from mixing the two technologies.

### A. Authoring a Java Applet to run inside a COM-based browser

Microsoft makes it possible for any standard Java applet to run inside a COM-based browser (or other application) such as Internet Explorer 3.0. Microsoft's Java Virtual Machine automatically maps the Java to COM without any special modification to either the Java applet or browser. The automated mapping allows COM-based programs to launch Java applets, and also allows Java applets to be controlled by ActiveX scripting engines such as Visual Basic Script.

Microsoft's JVM exposes a Java applet's public methods through OLE Automation, making the Java applet appear to the browser as a COM object, and fooling the Java applet into believing it is being called by another Java program. (*Ed. Note*: See scenario #1 in appendix for the technical details of how this works.)

### 1. Consequences

*Development implications.* A developer does not need to alter the either the Java applet or the COM-based browser for the applet to run.

*Impact on portability.* Since the Java applet is still completely in Java, it will still run an any OS/CPU with a Java browser. [It seems like a worthwile point to mention in the cross-platform, ActiveX comparison.]

*Impact on security & robustness.* Since the Java applet runs entirely in the JVM (which verifies the byte code and restricts file access), it maintains Java's inherent security and robustness.

## B. Creating a Java Application to be linked or embedded within an OLE container

Developers can use Java to create an OLE server--an application that can be linked or embedded into an OLE container (such as Microsoft Word). [Actually, we plan to have to docobj support in the SDK (available in about 2 months), currently, we support ActiveX controls which are slimmer and work in containers such as IE or VB 5.0.] For example, a charting application authored in Java can be used to embed a chart within a Word document. When the user opens this document in Word and selects the chart for modification, the Java charting application would take control of the menu, toolbar, mouse, and keyboard.

### 1. Consequences

*Development implications* The Java programmer must augment the Java source code to implement COM-based interfaces required for Object Linking and Embedding. (*Ed Note*: See scenario #2 in appendix for complete technical details of how this works.)[This is not true. Our development tools do not require you to augment Java source code.]

*Impact on portability.* A Java application linked or embedded within an OLE container still maintains Java's inherent portability. While the Java application contains code that implements the Object Linking and Embedding interfaces, this code will not affect how the applet runs on platforms which don't support COM—the code will simply be ignored. [You may want to mention the fact that COM connection is done through attributes, not code. This means that a class implemented in Java can replace an existing class which was implemented in COM with enhanced features on a COM supporting platform.]

*Impact on security & robustness.* While running, the Java application possesses all of Java's security and robustness. However, communication between the Java application and the OLE container does not share Java's security and robustness. Improper communications between the container and Java application passed via OLE interfaces could conceivably cause either or both components to crash. [Poorly written pure Java applets can crash just as easily.]

## C. Using an ActiveX control from within a Java Application

Java developers can leverage existing code packaged as ActiveX controls. The ActiveX control may take the form of a visual element, such as an spread-sheet control, or a COM-based system service such as MAPI. For example, a point-of-sale application written in Java can use an ActiveX control that displays a chart of sales by region.

### 1. Consequences

*Development implications.* A Java developer must incorporate a series of instructions explicitly designed to call the ActiveX control from within a Java application. Only ActiveX controls recently

authored using C++ and Microsoft's ActiveX Template Library (ATL) can be called. Version 1.0 of Microsoft's JVM does not allow developers to call legacy OCXs from within Java code. [This is not true. We just don't provide the frameworks which make it easy to host legacy OCXs from within Java applets, but you have access to all COM objects, including OCXs even in the current version. This framework could be built by developers (with a reasonable amount of Java code) because the basic access is there, but we will provide a framework in our SDK.]

*Impact on portability.* The Java application that incorporates an ActiveX control can no longer run on any Java platform. Its use is restricted to those platforms that support ActiveX controls, include a JVM that integrates ActiveX controls. A version of the required ActiveX control must be installed on the client.[This is not true either. The specific class or classes which were implemented in COM would need to be available as a Java class on non-COM platforms. If this was done, the whole application could run just fine.]

*Impact on security & robustness.* Java's security is breached and its robustness is disabled whenever an ActiveX control is accessed. [This is absolutely false. Unsigned, untrusted Java applets do not have direct access to any COM classes which are not specifically marked as safe for access by untrusted applets. We have actually extended COM security WRT Java. I do not remember being asked about this issues, and I'm beginning to wonder by whom this information was conveyed. Since this is factually incorrect, I'm concerned that someone in our company may need to have this explained to them.] Trusted ActiveX control operations are not restricted by Java's "sandbox." An ActiveX control has full access to the user's privileges. A malicious or errant ActiveX control can wreak irreparable harm on a system or network.

## D. Constructing a Java Application capable of launching a Windows program

When running on Windows clients, a Java application can launch any Windows program. For example, a browser authored in Java could launch Microsoft Word whenever it encounters a Word file on the Web.[Only signed/trusted applets could do such a thing since this level of access would breach our security model if done from an untrusted source.]

### 1. Consequences

*Development implications.* To launch a Windows program, the Java developer must incorporate special instructions that implement Object Linking and Embedding COM-based interfaces. (*Ed Note*: See scenario #3 in appendix for complete technical details of how this works.)

*Impact on ~~security & robustness~~ [cross platform].* The Java program can no longer run on any Java platform. Its use is restricted to those platforms that support COM, include a JVM that integrates COM, and a have browser (for Java Applets) that uses this JVM. A version of the required Windows program must be installed on the client.

*Impact on security & robustness.* [Factually incorrect, and damaging to the understanding of our real security model. If you really believed we hadn't thought of such a gaping hole, why not just ask?]The ability for a Java program to launch a Windows program through a COM interface breaches Java security. While Java's sandbox prevents a Java applet from circumventing security restrictions, an installed Windows application has no such restrictions. A malicious developer could create a Java applet that directs a Windows application to violate these security restrictions on the applet's behalf. For example, a Java applet could download a malicious Word macro file and launch Word with instructions to execute the macro. (*Ed. Note*: For this reason, Internet Explorer 3.0 allows users the option to prevent Java applets from accessing ActiveX.[Not true. Since we do not expose any new security holes in Java, and, in fact, have been told by Princeton security experts that we are more secure than other JVM implementations, we have only provided the option to disable Java if desired. While we believe that our implementation is so secure that this wouldn't be necessary, publicity of other Java VM's security holes has caused IS managers to want this.])

## E. Creating Java middleware that performs services on behalf of a Windows front-end

Java developers can create sophisticated middleware that provides Windows-based front-end programs access to back-end services. For example, systems developers at a health care provider could use Java's object oriented development environment to create sophisticated middleware solution that talks to a medical patient database distributed across multiple servers. Front-end developers could use familiar tools such as Delphi or Visual Basic to rapidly create and maintain the large and continually changing variety of forms required by medical insurance companies.

### 1. Consequences

*Development implications.* Since the Windows-based front-end and Java middleware communicate via COM, the Java developer must augment the Java program with source code that implements COM-based interfaces. (*Ed. Note:* See scenario #2 in appendix for complete technical details of how this works.)

*Impact on ~~security & robustness~~ [cross platform].* The Java middleware will function on any platform that supports both Java and COM. The middleware may execute on the client machine, or on a server (the latter requires that both the client and server support DCOM).

*Impact on security & robustness.* While running, the Java middleware possesses all of Java's security and robustness. However, communication between the Java Server and the Windows client does not share Java's security and robustness Improper communications between the client and server could cause either or both components to crash.[Again, any improperly written applet can crash.]