Java

HotJava

How hot is it ?

Plan:

- Java
- Security in Java
- ActiveX
- Java vs. ActiveX
- Java Beans

The design goal for Java are:

- Simple, Object Oriented and Familiar
- Architecture Neutral and Portable
- Hight Performance
- Interpreted, Threaded and Dynamic
- Robust and Secure

Why is it Simple, Object Oriented and Familiar ?

- The language is very simple, and fundamental concepts are grasped quickly, and it’s very easy to programme
- There are a lot of existing libraries
- A lot of bad “stuff” has been removed to make it simple
- Since object orientation is a big buzz-word and major programming languages are object oriented - java is that too
- The syntax is C/C++ alike and hence familiar to a lot of programmers

Java is close to C and C++, things that were removed include:

- No typedefs, Defines or Preprocessor; typedefs implemented as classes, defines as constants
- No structures or Unions; classes capture this
- No functions; methods instead
- No goto; multilevel breaks
- No operator overloading
- No automatic Coercions
- No pointer

Primitive Data types, are equal on all platforms:

Type	Size/Format	Description
byte	8-bit two’s complement	Byte-length integer
short	16-bit two’s complement	Short integer
int	32-bit two’s complement	Integer
long	64-bit two’s complement	Long integer
float	32-bit IEEE 754	Single-precision float
double	64-bit IEEE 754	Double-precision float
char	16-bit Unicode character	A single character
boolean	true or false	A boolean value

The rest of the language:

- Arrays are first-class language objects (a real object), that means they have a runtime presentation
- Strings are also object divided into 2 kinds:
  - String class (read-only objects)
  - StringBuffer (mutable string objects)
- We don’t have pointers to arrays or strings
- Multi-Level Break since there is no goto

- Memory Management and garbage collection (automatically), that means
  - No pointer arithmetics
  - No malloc or free
  - The new operator allocates memory for objects, but no explicit free
  - Memory management model based on:
    - ∗ Objects
    - ∗ References to objects through symbolic handles
- The garbage collector is running in the background, as a low priority thread

HellojavaWorld.java:

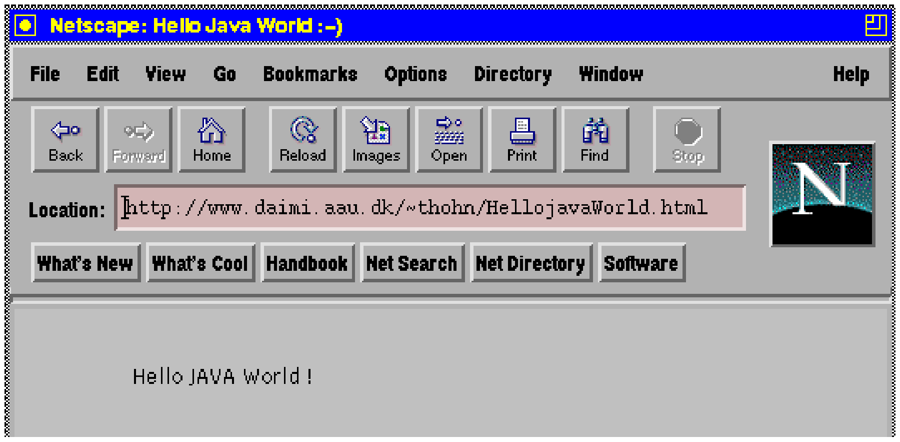
```
import java.applet.Applet;
import java.awt.Graphics;
public class HellojavaWorld extends Applet {
  public void init() {
    resize(200,50);
  }
  public void paint(Graphics g) {
    g.drawString("Hello JAVA World !",50,25);
  }
}
```

Notes:

- Source files must have same name as the class being defined !
- Only one public class in each file

HellojavaWorld.html:

```
<html>
<title>Hello Java World :-)</title>
</head>
<body>
<applet code=HellojavaWorld.class width=200 height=50>
</applet>
</body>
</html>
```


Netscape: Hello Java World :-)
File Edit View Go Bookmarks Options Directory Window Help
Back Forward Home Reload Images Open Print Find Stop
What's New What's Cool Handbook Net Search Net Directory Software

Location: http://www.daimi.aau.dk/~thohn/HellojavaWorld.html

Hello JAVA World !

Generic applet-tag:

```
<APPLET
   CODEBASE CDATA #IMPLIED  -- code base --
   CODE CDATA #REQUIRED     -- code file --
   NAME CDATA #IMPLIED      -- applet name --
   WIDTH NUMBER #REQUIRED
   HEIGHT NUMBER #REQUIRED
   ALIGN (left|right|top|texttop|middle|
          absmiddle|baseline|bottom|absbottom) baseline
   VSPACE NUMBER #IMPLIED
   HSPACE NUMBER #IMPLIED
</APPLET>
```

Why Architecture Neutral and Portable ?

- Java should support development of applications in heterogeneous environments
- The idea of binary distribution is dead !
- Java was not original designed for the WWW, but WWW browsers typically run on different platforms
- The java-compiler generates “byte code” - which is architecture neutral intermediate format - instead of native machine instruction (machine code)
- Since its intermediate code - it’s interpreted in a Virtual Machine, which is a abstract machine

- A true portable system should be
  - Architecture neutral
  - Have a strict definition of basic data types - that gives no data incompatibilities across platforms
- Java should be able to run on every architecture, since it follows the POSIX standard
  (POSIX is a standard way for application programs to obtain basic services from the operating system)

What about Performance ?

- It's well know that interpreted code is slower than compiled code
- No need for type checking at runtime
- Automatic garbage collector - prevents memory leaks and hence starvation
- They claim that a Java program in average is 20 times as slow as a C program (Java in a Nutshell)
- The garbage collector is running in the background as a low priority thread, and does thereby not affect the execution of the program

It’s Interpreted, Threaded and Dynamic.

- Since the language is interpreted it can run on any machine on which the interpreter and run-time system has been ported
- It’s ideal for prototyping, since you avoid the heavyweight compile, link and test-cycle
- There is support for concurrent threads of activity, since we have threads
- Classes can be linked at runtime, the classes can be downloaded at runtime from across networks - the downloaded code is of course verified, so it can be considered secure

It's Robust and Secure:

Robustness

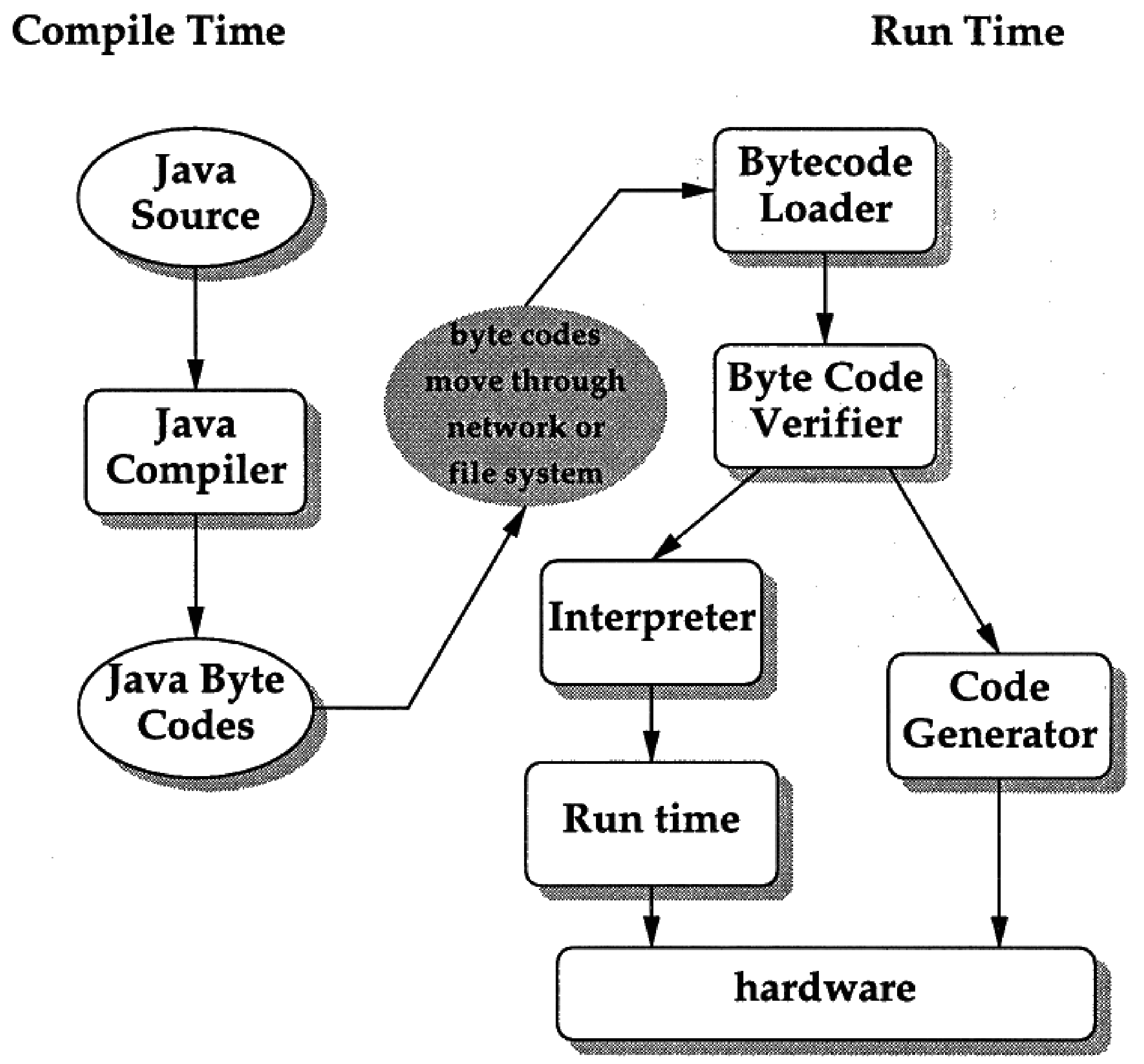
- Java is strongly typed
- Hence syntax-related errors will be caught at compile time
- Since we have true arrays and strings, it's possible for the interpreter to check array and string indexes which eliminates corruption of data and overwriting of memory
- Many compile-time checks are carried over to the run-time system. This provides greater flexibility but also gives bigger byte code
- The linker understands the type-system

Security

- Based on sandbox approach
- The memory layout decisions are not made by the Java compiler as they are in C/C++
- There are no pointer, references to memory are done by symbolic “handles”, these are resolved to real memory addresses at run time by the Java interpreter
- Programmers can’t forge pointers to memory, since the memory allocation/referencing model is opaque to the programmer
- Late binding of structures to memory
- Foreign code (downloaded from the net) is verified by the Byte Code Verifier

The Byte Code Verification process:

- Newer trust downloaded code, you can’t ensure that it was produced by a hostile Java compiler
- So the runtime system has to verify the byte code, the tests that are done include:
  - Check that the format of the byte code is correct
  - A simple theorem prover is applied to the byte code that ensures:
    - ∗ It doesn’t forge pointer
    - ∗ Access restrictions are not violated
    - ∗ Access object are what they are
- A language that is safe plus run time verification of generated code is our guarantee of no violation !


Compile Time
Run Time
Java Source
Java Compiler
Java Byte Codes
byte codes move through network or file system
Bytecode Loader
Byte Code Verifier
Interpreter
Run time
Code Generator
hardware

After the Byte Code Verifier, we know:

- There are no operand stack overflows or underflows
- Object field accesses are known to be legal - private, public or protected
- The types of the parameters to all byte code instructions are known to always be correct

Security in the Byte Code Loader:
While code is executed in the run time environment it can request a particular class/classes to be loaded - What happens:

- The classes are verified by the byte code verifier
- Classes are separate into different name spaces
  - Name space for local classes (built-in classes)
  - Name space for classes loaded across the network (imported classes)
- When references to classes are made the built-in classes is the first place to look, thereby built-in classes can’t be spoofed
- No cross references between different name spaces
- Classes imported from different places are also separated from each other

How secure is

Java ?

Despite that Java was designed to be a robust and secure language a variety of errors has been spotted - even though.
These include:

- Implementation errors
- Unintended interactions between browser features
- Differences between the Java language and the semantics of the byte code
- Weakness in the design of Java and the byte code format

Implementation errors:

- DNS weakness
- Buffer overflow
- Disclosing Storage Layout
- Inter-Applet security

DNS weakness:
The JDK and Netscape implementation of Java only allows the applet to make a TCP/IP connections back to the server from where the applet was loaded. The policy was enforced in the following way, it seems as a sound policy:

- Get all IP-addresses from the host the applet came from (A)
- Get all IP-addresses from the host the applet wants to connect to (B)
- Allow connection host in A $\cap$ B

Problem in step 2, the applet can ask to be connected to a arbitrary host on the Internet including spoofed DNS-servers, which supplies set B.

Buffer overflow:
The use of sprint() statements that are unchecked, the execution stack can be overwritten and hence hostile code could be executed.

Disclosing Storage Layout:
Even thought we don't have directly access to memory through pointers, applets can find out where in memory objects are stored. Since all Java objects have a hashCode() method, and it casts the address of the objects into an integer and returns it, this could be a problem.

Inter-Applet Security:
Applets can get informations on other applets running on the system and invoke one of the methods stop() or setPriority()

Differences between the Java language and the semantics of the byte code:

- Illegal package names
  - Punctation in package names (java.io) are replaced by a '/' and '/' is illegal as the first char in package names
  - Even thought if the first char is a '/', the runtime system will try to load the package from a absolute path
  - The problem is even bigger with data cached from the local file system
- Superclass constructors
  Java requires that all constructors call either another constructor of the same class or a superclass constructors as their first action. The system classes ClassLoader, SecurityManager and FileInputStream all rely on this

behaviour for their security.

Java prohibits the following code, the byte code verifier doesn't

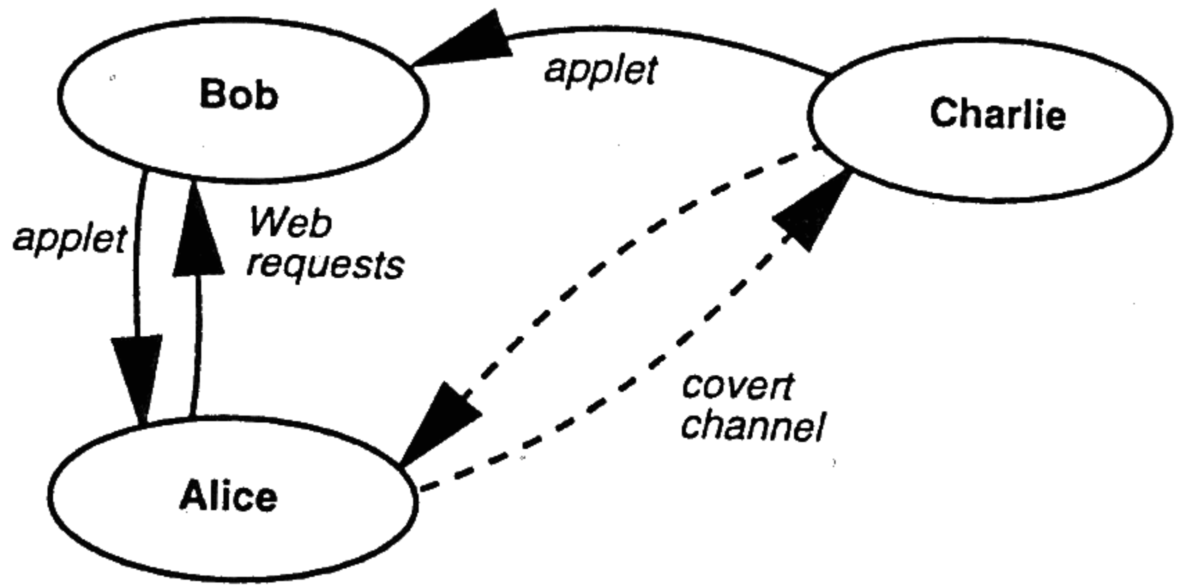
```
class CL extends ClassLoader {
  CL() {
       try { super(); }
       catch (Exception e) {}
       }
}
```

This allows us to

- Build partially initialised ClassLoader or system classes
- Any code loaded asks the ClassLoader to load additional classes
- Hence the System name space is NOT searched first
- The applet can construct a customised name space

Unintended interactions between browser features:

- Denial of Service attacks
  - Attacks like busy-waiting
  - Locking critical pieces of the browser
  - Degeneration of service
- Two or Three Part Attacks
  - Two party attacks requires a Web server; on which the applets resides, to participate in the attack
  - Three party attacks can originate from anywhere on the Internet and are thus more dangerous



- Covert Channels
  Make it possible to establish a two-way communication with arbitrary third parties on the Internet, the problems arises because of:

  - A weakness in the accept() system call
  - If the WWW server runs a SMTP-daemon
  - Redirect through the URL-redirect feature

- Information available to applets
  - System.getenv() has no security checks !
  - Access to system clock
  - A applet can consume all free space on clients file system

Java Language and Byte code weakness:
The definition of the Java language and the byte code are to week, seen from a security viewpoint. Why is this the case ?

- Java has no formal semantics or description of the type system
  - Hence it’s impossible to make any formal reasoning about Java
  - The security in Java relies on the soundness of the type system
- Information hiding
  - The byte code verifier doesn’t enforce the semantics of the private modifier for local loaded code
  - This implies that local code has access to other applets variables

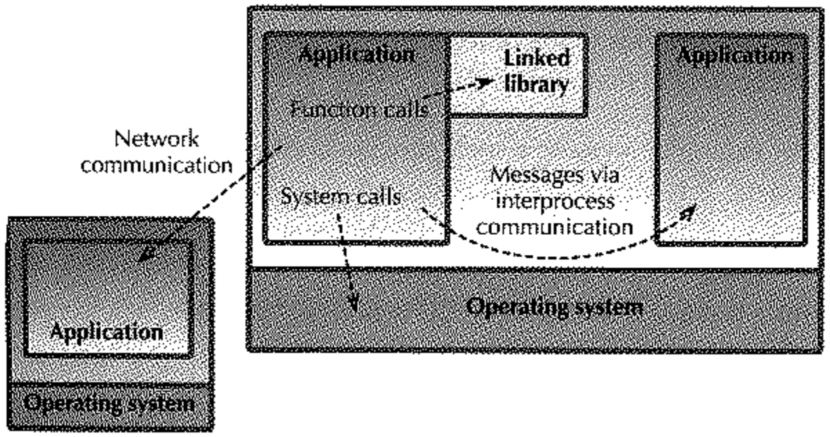
  - Hence the definition of Java could be altered not to leak information like this

- The generated byte code is weak since
  - Byte code is in a linear form. Type checking requires global data flow analysis
  - Analysis is complicated through exceptions and exception handlers
  - Traditional type checking is compositional
  - The type verifier can not be proven correct since the lack of a formal description of the type system
  - Object oriented type systems is a current research field

ActiveX

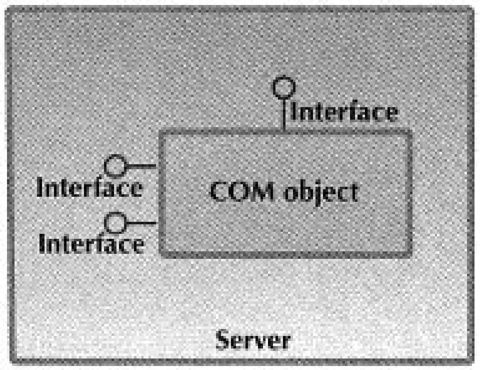
COM (Component Object Model) is a foundation for interaction between software libraries, applications, system software etc. ActiveX is based on COM, but what is COM really, Let's first try to figure out how software can access software services provided by another piece of software:

- Application could use functions in a library
- Through Inter-process communication, via a protocol
- System calls to the OS
- Network service, like RPC

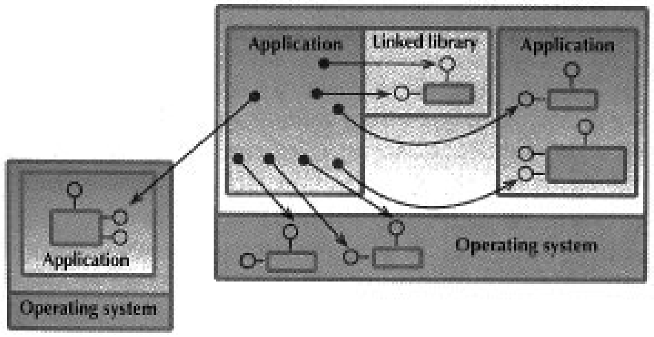


The mechanism for getting these services are different depending on the type of service.

COM defines a way to access software services. A COM object has a number of interfaces that each includes a number of methods that can be invoked, the object is implemented on a “server”, which can be a DLL or a separate process.



Methods are invoked through pointer to the interface.
With the COM model it’s transparent to the programmer/user how the service is actually achieved, it’s always the same abstraction that is used.



The Component principle

Software is build from components, But at least 3 problems arise:

- No standard for linked library code, source code must be distributed along with the library
- Reuse of objects in different languages is often impossible
- Relinking/recompiling an entire application, if one object is changed, this problem could maybe be solved by “incremental compiling”

The COM model solves these problems, since:

- COM objects can be packed into libraries/EXE files, without distributing the source code
- Standard way to access binary objects
- Objects are instantiated at need, the objects are always the newest version of the object

The benefits of COM are:

- You get all benefits from object orientation
- Provides a system for consistency
- Language independent
- Versioning through interfaces
- Available on many OS; currently Windows, NT and Mac
- There is as form of distributed COM (DCOM), relies on RPC

ActiveX is a computing technology composed of different components each doing a specific task. The components are:

- ActiveX controls
- ActiveX scripting
- ActiveX documents
- ActiveX ISAPI

The capabilities of ActiveX controls are:

- Compute and Manipulate data
  - Controls can be used in both WWW pages and as stand-alone applications
  - Controls can be written in any language which could/should yield faster applications
- Communicate
  - Controls can share data or give instructions to other OLE-capable objects
  - Controls can read/write files on any file system !
  - Controls can be TCP/IP enabled, which means a great power

  - Its actually possible to do workgroup computing including the whole Internet

- Saving programming effort
  - Controls can “easily” be created from scratch
  - Controls can be written in any language including Java !
  - Controls are modular objects, and hence functionality can be specialised depending on the context (Web/Application)

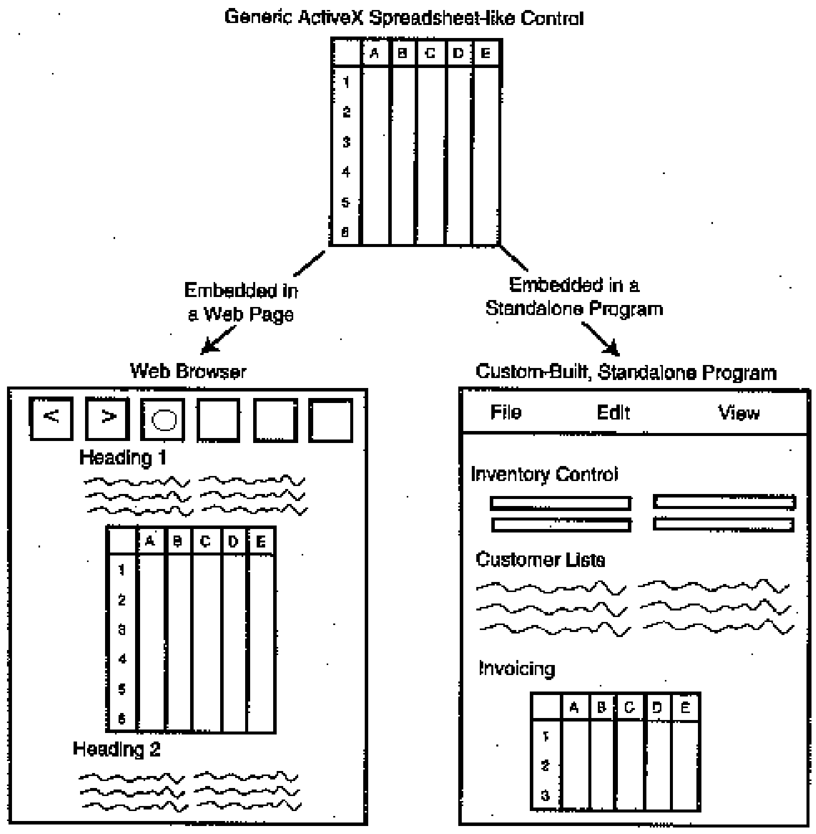
ActiveX scripting:
If you don’t want to create ActiveX controls, you can still make fancy things with the help of VBScript.
Why VBScript ?

- Web masters are anxious about custom created ActiveX controls
- It’s really easy to learn (subset of Visual Basic)
- VBScript can communicate via OLE to other OLE applications
- It’s embedded in the HTML-document

ActiveX docs:

- The HTML file-format is too “simple”, even though it became popular because of it’s simplicity
- Some programs produce output that is well suited for the Internet, but the output can’t be represented in HTML. There are at least two possible solutions:
  - Make your program a helper application
  - Use ActiveX documents
- Your output can be displayed inside the browser in a “container”. Virtual Reality could for example be seen on the Internet as a ActiveX document
- OLE links will be extended across the whole Internet and not just on a local hard drive

- Programs can be used as ordinary Windows programs or inside ActiveX documents


Generic ActiveX Spreadsheet-like Control
Embedded in a Web Page
Embedded in a Standalone Program
Web Browser
Heading 1
Heading 2
Custom-Built, Standalone Program
File
Edit
View
Inventory Control
Customer Lists
Invoicing

How to access Active Documents:

- Controls are accessed via the HTLM tag $< OBJECT >$
- When the $< OBJECT >$ is met, 3 things are done:
  - If the controls aren't present, download them from a code server
    - The code is downloaded by the URL moniker asynchronously, the location is determined by the Internet search patch mechanism. It's also possible to prevent code from being downloaded from the net

- Check that the code is “safe” to install
  - ∗ The Windows Trust Verification service is used to verify that the code is safe
- Install the code
  - ∗ Code is installed into a download cache
  - ∗ DLL or EXE programs register themselves

Security in ActiveX:

- We need to know that the code, we will execute, is safe in some sense
- The sandbox approach significantly limits the kind of things you can do
- The code should indicate that it’s safe to run
- The code has attached a digital signature
- Signatures are validated through the Windows Trust Verification service

Windows Trust Verification service:

- Search for a signature block (digital signature). It contains:
  - The author of the file
  - A public key
  - Encrypted digest of the file’s contents
- Validate the certificate, if not indicate it
- Decrypt digest with public key and regenerate digest to ensure that the files has not been tampered
- Install the component

Java

VERSUS

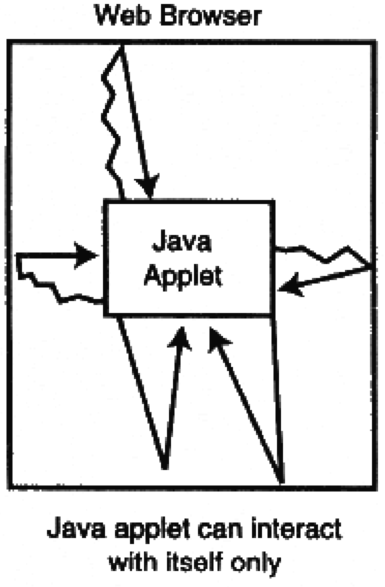
ActiveX

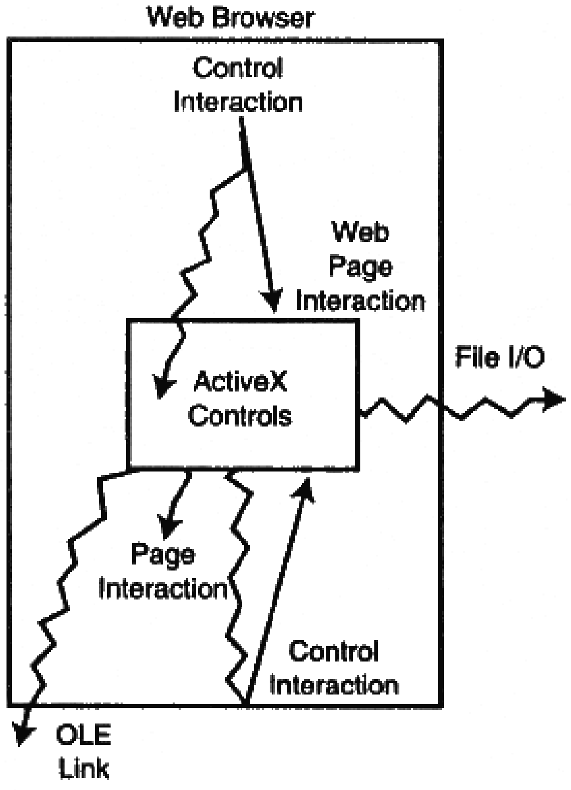
Java pursuits:

- Java is a programming language
- It’s easy to use and “clean”
- Java is nothing special without standard classes and Internet support
- Java is the first truly cross-platform programming language
- Java has a lot of security stuff

ActiveX pursuits:

- ActiveX is a technology, closely related to OLE
- The technology can integrate existing product on the WWW
- All code base on OLE is ActiveX enabled
- ActiveX controls may be written in any language, that supports COM


Web Browser
Java Applet
Java applet can interact with itself only


Web Browser
Control Interaction
Web Page Interaction
ActiveX Controls
File I/O
Page Interaction
Control Interaction
OLE Link

It’s amazing how different Java and ActiveX are in their implementation, since they have the same goal - to add “interactivity” to the WWW.
Lets look at some common issues:

- Security
- Practical usage
- Development strategy
- The Crossroads

Security:

- Security in Java is based on verification of the produced byte code, cost of performance
- The Byte code is interpreted and no I/O access, which limits the use of Java
- ActiveX uses digital signatures
- Verification through Windows Trust Verification service
- Microsoft learned from the problems with Java security

Practical usage:

- No real applications have been build with Java
- In ActiveX; applications can be build as they are and as WWW applications
- ActiveX has inherited a lot of practical applications and controls since it's based on OLE

Development strategy:

- Java is powerful with respect to network application development
- Unfortunately there has been a lack of development tools
- These problems are not present when you make ActiveX applications

The Crossroads:

- Its not necessary to know both ActiveX and Java, since Java applets can be integrated with ActiveX controls
- ActiveX controls can be accessed from Java applets

Sun strikes back with Java Beans !

What are the goals of Java Beans ?

- To define a software component model for Java, that enables third party members to create and ship Java components, which can be composed into applications by the end user
- To provide API's that support ActiveX/OLE/COM, Opendoc and LiveConnect
- To be able to create large scale application in Java
- To enable dynamic integration. This means components can:
  - Become interactive
  - Be able to capture events
  - Call methods in other components
- To be a platform neutral component architecture

- Be reasonable simple, like Java
- Use the standard security model that is also used by Java

Java Beans Specification 1.0 just released.