Java Future

Sun Stewardship

Java is controlled by Sun, which is not as appealing as control by a non-profit such as the W3C

However, there is precedent -- C and C++ were controlled by AT&T without harm

The history of Sun's guidance of Java in the last 5 years has been pretty prudent and reasonable, so they have earned some trust.

Hopefully, Sun is happy to develop Java as OS agnostic platform -- app writers may code to Java, and their apps will run everywhere.

There is a stereotype that Sun is run by engineers, and Java may be an outgrowth of that.

Microsoft has the most to lose from applications not being OS specific

By the same token, every other vendor (Sun, IBM, Oracle, ...) benefits from a Java as a healthy, non Microsoft-specific platform for development.

Java Open Development

How to find about Java future directions?

Sun actually does Java development very much out in the open

Get a free account on java.sun.com, then...

1. Read the top 25 bugs on the buglist
2. Read the top 25 Request For Enhancements (RFE)

Go the JCP (Java community process) site (http://www.jcp.org/) and look at the proposals in the various stages of development. You can observe movements and arguments for a couple years as features make their way into the language.

Things which are showing up now in Java have been visible in some form or other above for years.

Don't be discouraged by the complaining tone in the forums above -- Sun gets a lot of credit for making their bug database and its arguments public. No complex system can exist in the open like that without a lot of flaming, complaining, and posturing (this seems to be a truism of online communities).

Java Development Themes

Major themes in Java...

Backward compatible -- old code continues to run, even as new features are added

Portable -- write once, run everywhere

Large library -- more and more off-the-shelf features get added to the library

Elegant/Structured style vs. "quick 'n dirty" like Perl

Slow progress -- Sun's guidance has tended to be slow and prudent

Sun seems to have a bias toward the "Elegant, Full-featured" solution instead of the "Simple but fast" solution. Time will tell if this is a good strategy. I suspect it

is, considering the pace of hardware improvement vs. 20+ year lifetime of a popular computer language

Java Niches -- Server vs. Client

Niche: server-side internet apps -- Java is very popular here already -- portable, secure, programmer efficient -- show well in this niche
- "Business logic" applications using Java and its JDBC library to connect to the database and fiddle around with the data. Note: possibly no GUI, just strings, ints, dates, etc.

Niche: "custom" applications
- A custom GUI application that is part of a larger custom system -- e.g. the "View Order Status" application used by the foo.com customer service people

Possible niche: Client side java

Possible niche: Small devices -- palm pilots, TVs, ...

JDBC

Java package to allow SQL database access.

Allow the Java code to be independent of the particular database (Oracle, MySQL, ...) in use

Java Servlets

An advanced form of CGI written in Java

The servlet engine can do basic session management, which makes the servlet simpler

Java Server Pages (JSP)

If servlets are the business logic, JSPs are pure presentation -- HTML with a little Java markup in them to pull up the data.

Similar to ASP and PHP, but with a Java bent.

Java Client ?

Jar Files

Java Runtime Environment (JRE) installed
- If everyone had Java installed, then you could send .jar files around -- chicken and egg problem (like PDF, like Flash)

Java Web Start
- Access a Java app through a URL
- Supports signing, etc.

Java 1.4

Released Feb 2002

http://java.sun.com/j2se/

New features --
http://developer.java.sun.com/developer/technicalArticles/releases/j2se1.4/

New IO - NIO (1.4)

http://java.sun.com/j2se/1.4/docs/guide/nio/index.html
http://developer.java.sun.com/developer/technicalArticles/releases/nio/
Non-blocking IO for sockets
    vs. the old 1-thread-per-socket model
New buffering system
    Like a big array of binary data

Regular Expressions (1.4)

http://developer.java.sun.com/developer/technicalArticles/releases/1.4regex/
Similar to Perl -- match and extract char regions
Syntax is more structured than in Perl -- more steps, more syntax

```
    // Pattern used to parse lines
...
    private static Pattern linePattern  = Pattern.compile(".*\r?\n");
        CharBuffer cb = ...;
...
        Matcher lm = linePattern.matcher(cb);   // Line matcher
        while (lm.find()) {
            CharSequence cs = lm.group();
            System.out.println(cs);
        }
```

Quick 'n dirty variant
    Less efficient
    boolean b = Pattern.matches(".*\r?\n", string)
    Good example of "client oriented design" -- the API should make common, obvious case easy to code.

Assert (1.4)

http://java.sun.com/j2se/1.4/docs/guide/lang/assert.html
assert i==0;
Throws AssertionError
Can be turned on and off by class or package at CT or RT
Never put code that needs to run in an Assert, since the assert may be effectively deleted in some cases, but the program should still work
Yes:
    int err = foo();   // do the computation outside assert()
    assert(err == 0); // check error condition in assert
No -- bad. In this case, if the assert is deactivated, foo() is no longer called.
    assert(foo() == 0);   // NO bad

Java Image IO (1.4)

http://java.sun.com/j2se/1.4/docs/guide/imageio/index.html
More sophisticated APIs for reading and writing image data

AWT/Swing (1.4)

FocusManager -- keyboard typing

Which component is currently "focused" -- getting keystrokes
New centralized handling of focus -- acquire / release / refuse-to-release focus
Image drawing slowed down significantly in 1.2 vs. 1.1, since the handling of images was made more abstract. (Perhaps another example of Sun preferring "elegance" a little too much over practical issues.)
A big issue for Swing apps, especially when using X Windows
1.4 gets some of the speed back
Slow startup time is still an issue in my mind, but every year computers get faster, so maybe in a year or two it won't matter.
New "fullscreen" mode

Java Generics (maybe in 1.5)

http: / / developer.java.sun.com / developer / earlyAccess / adding_generics /
Compile time types
The RT is the same -- really it's checking the type every time, but you don't have to put the cast in at CT
Cleans up the code and finds some erors at CT, which are now masked by all the casting

```
// Suppose Foo responds to the bar() message
ArrayList<Foo> list;
Foo f = ...
list.add(f);...
...
Iterator<Foo> it = list.iterator();
while(it.hasNext()) {
   it.next().bar();  // NOTE: no cast required, it.next() has correct CT type
   ...
}
```

Java 2d / Java 3d / Imaging

Java 2d -- resolution independent drawing (like PDF)
Java 3d -- Java interface to a fairly rich 3d support
Image IO -- package for manipulating image data specifically
Advanced Imaging -- manipulation of large bitmap images (http: / / java.sun.com / products / java-media / jai /)
Scalable Vector Graphics (SVG) -- W3C standard for vector graphics(similar to PDF) -- SVG will be very useful if it catches on. The Batik project links SVG and java (http: / / xml.apache.org / batik /)

Java Media Framework (JMF)

http: / / java.sun.com / products / java-media / jmf /
Present and manipulate media such as images, sounds, and video

RMI

Distributed processing -- make objects that are on "remote" JVMs (on other machines) look like ordinary objects in your local JVM.
Depends on portability to send bytecode around the network.
Depends on serialization standard to move objects around the network.
Depends on "sandbox" security to run the inbound code safely.

Performance is a little slow, since it depends on serialization machinery, however the network itself probably represents most of the delay, so who cares.

JINI

"Federation" layer allowing little devices to cooperate. Everybody thinks this niche is going to be the next big thing, but it doesn't really exist yet.
Example --
- Your CD player sends its GUI code (java bytecode) to your palm pilot. The GUI code understands the CD player. On the Palm, the GUI code presents all the songs that are on the CD player, and you use the GUI to communicate back to the CD player. Your Palm and your CD player interact without being pre-designed for each other by exchanging code.

Java Beans

Actually really simple -- like an ADT
Bean
- Has an empty ctor
- Has getFoo and setFoo methods for each of its public properties

Unit of exchange
- Module A wants to package information for others to use
- Set up a "bean" class that uses getters and setters in the standard way
- Then other programmers can use it easily
- e.g. Business logic layer creates a "cart" bean that contains a bunch of product beans. Pass the cart bean to the JSP layer, it extracts the various bits of info and renders out the HTML.

Bean tools
- Tools can understand the create/get/set nature of the bean to allow people to manipulate it without writing code.

J2ME/MIDP

Mobile Information Device Profile
http://java.sun.com/j2me/
http://java.sun.com/products/midp/
Works on PalmOS 3.5
Subset of Java for small devices -- not as heavyweight as Swing
Allow you to write small apps that work on Palm, Windows CE, ...
Also, Connected Limited Device Configuration -- CLDC -- phones, etc.
Many vendors are excited about the "small device" space -- a new frontier vs. the desktop

Old Serialization

Design -- how to you serialize off a Java class?
Old serialization: write out its ivars
Problem: what if the class changes impl?

New, XML "Persistence"

http://java.sun.com/j2se/1.4/docs/guide/beans/index.html

http://java.sun.com/products/jfc/tsc/articles/persistence/
http://java.sun.com/products/jfc/tsc/articles/persistence2/
http://java.sun.com/products/jfc/tsc/articles/persistence3/
Only serialize state that is accessible through public get/set methods (the "bean" view of an object)
This is the technology that underlies the new GUI/Bean/XML layout editor technology (not yet released)
Be smart about constructor defaults. To serialize Foo f...
1. Construct Foo s;
2. Compute what setXXX() messages are necessary so that s looks like f.
3. Record the arguments for the ctor/setXXX sequence -- that **is** the persistent form of f
Advantages: totally independent of implementation. In fact you could serialize as Foo, and then read back into a different class, say Bar, so long as Bar had the same public ctor/get/set semantics as Foo.

New Java GUI Editor (1.4)

The "BeanBox" app lets you draw/edit your GUI. (not yet released)
When you're satisfied, you serialize (dehydrate) down the collection of GUI objects
At run-time, the objects are read in to memory (rehydrated) to re-create the whole GUI and all the listener connections.

New XML Serialization

XML is the serialization format
Objects are serialized based on their bean-like getter/setter interfaces. The "serialized" design is the same as the API design -- there are not two designs to do.

New EventHandler Class

Glue: knows the source (e.g. JButton), the target (e.g. the listener), and the name of the message to send (e.g. foo()).
Set up the EventHandler glue object
It listens for you
When the event happens, it goes to the target, and sends the message chosen.
Note: the message can be whatever the target likes, not some fixed message like actionPerformed(). This all depends hugely on "reflection" (runtime lookup of ivar names, method names, etc.) to work.

Example

In draw program, have a "Delete Shape" button
In canvas object, have a delete() method.
Use a EventHandler object to directly connect the button to send the delete() message to the canvas when clicked -- no listeners syntax/inner class/etc. required.