JAVA oder C++ ?

Fritz Schmöllebeck

Eine Frage die in letzter Zeit oft gestellt wird. Als sog. „Applets“ in HTML-Seiten kennt man JAVA Programme wohl derzeit am besten, aber auch als Stand-alone-Applikationen sind sie durchaus einsetzbar. Im Umfeld der Elektronik wird derzeit das objektorientierte Softwaredesign auch bei der Entwicklung von Embedded Systems eingehend diskutiert. Im Zusammenhang mit Diskussionen dieser Art scheint es sinvoll einen Blick auf die Designziele von Java zu werfen. So ist in den Whitepapers von SUN Microsystems 1994/95 zu lesen:

The Java programming language and environment is designed to solve a number of problems in modern programming practice. It started as a part of a larger project to develop advanced software for consumer electronics. These devices are small, reliable, portable, distributed, real-time embedded systems.

When we started the project, we intended to use C++, but we encountered a number of problems. Initially these were just compiler technology problems, but as time passed we encountered a set of problems that were best solved by changing the language.

Dieses Statement stellt JAVA primär als Entwicklungssprache von Embedded Systems dar. Die Verwendbarkeit von JAVA für die Entwicklung von Applets welche in Internet-Browsern verwendet werden können ist sozusagen eigentlich ein Abfallprodukt oder zumindest eine nicht von vornherein beabsichtigte und geplante Eigenschaft.

In den folgenden Abschnitten wollen wir sowohl die Analogien als auch Unterschiede der Sprachen C++ und JAVA herausarbeiten.

Einbettung in die Systemumgebung

Bei den Stand-alone-Applikationen müssen wir uns notgedrungen schwerpunktmäßig auf PC bzw. Workstation Architekturen beschränken, da für die in unsrem Elektronikumfeld gängigen Mikrocontroller kaum JAVA-Laufzeitsysteme (Virtuelle JAVA Maschinen) existieren. Im Bereich der Netzwerk-Computer sieht die Sache schon anders aus. Hier wird mit Hochdruck an Betriebssystemen gearbeitet welche auf festplattenlosen (diskless) Rechnern effizient lauffähig sind und für den Netzwerkbetrieb angemessene Sicherheitsfunktionen zur Verfügung stellen. Scheint die hier geführte Diskussion für Embedded Systems im Sinne von Mikrocontrollern vorerst vielleicht akademisch, so ist sie es für Embedded Systems im Sinne von netzwerkfähigen Minimalsystemen ganz und gar nicht.

Vor allem das oft ins Treffen geführte Argument von der Schwerfälligkeit des Diskless Computing sollte man hinterfragen. Unsere derzeitigen Erfahrungen sind geprägt von Betriebssystemen welche primär für Einzelrechner entwickelt und dann netzwerkfähig gemacht wurden. Dies gilt sowohl für den Desktopbereich (Windows 3.11/95/NT, OS/2, MAC-OS,...) als auch für die UNIX-Welt mit ihrem X-Windows System. Eines der wenigen schon sehr früh für Netzwerkcomputing entwickelte Betriebssystem ist VMS der Firma Digital Equipment (DEC). Dieses System hätte vom Kerndesign viele Merkmale welche ein effizientes Clusterkonzept und auch Diskless Computing stark unterstützen würden. Die Nachteile sind aber leider auch hier nicht zu übersehen. VMS ist proprietär und es lief zu der Zeit als DEC am Netzwerksektor noch entsprechenden Vorsprung hatte praktisch ausschließlich auf VAX-Procesoren und diese konnten vor allem aufgrund ihrer Architektur - sie waren CISC-Schulbeispiele - nicht mit der Geschwindigkeit der Konkurrenz mithalten. (Das Clusterkonzept - jetzt von Microsoft als große Neuigkeit verkauft - war übrigens in VMS schon Mitte der 80er Jahre im Grundkonzept implementiert). X-Windows besitzt in gewisser Weise eine Ausnahmestellung. Vom Konzept her sind X-Terminals Diskless Workstations welche vor allem Displayaufgaben übernehmen sonst aber vom Standpunkt der Aplikationsprogramme ein eher zentralistisches Multiuserkonzept verfolgen, welches vom zentralen Hostrechner enorme Resourcen fordert.

Von diesen Szenarios schließt man nun gerne „Das kennen wir alles schon,... das hat nicht funktioniert,... das kann auch mit den NCs nicht funktionieren, ...“ - aber Vorsicht !

In den bisherigen Konzepten waren unsere exekutierbaren Applikationsprogramme (Images, .exe-Dateien) vom Prinzip her für Einzelmaschinen gemacht. Anstatt diese Applikationen von der Festplatte zu laden wurden sie über ein Netzwerk geladen, oftmals in einer Weise welche die Kernfunktionalität des verwendeten Betriebssystems trickreich zumindest teilweise lahmlegte. Und diese Vorgangsweise war und ist - schwerfällig und zum großen Teil ineffizient. Sie ist gleichzeitig aber auch nicht so einfach zu ändern weil es da noch so etwas wie „Kompatibilität“ zu berücksichtigen gilt.

JAVA geht hier einen anderen Weg. Auch dieser ist nicht ganz so neu wie er scheint. Im Umfeld der sog. verteilten Betriebssysteme (z.B. Amoeba, ...) gibt es den Begriff des Microkernels schon relativ lange. Dabei verfolgt man wie auch bei JAVA die Strategie einen effizienten kleinen Kern des Netzwerkbetriebssystems auf jedem System im Netzwerk laufen zu lassen. Dieses Kernsystem wird bei JAVA vor allem mit hoch effizienten Laufzeitservices bestückt. Daraus entsteht dann die virtuelle JAVA Maschine - ein (virtueller) Rechner der anscheinend als Maschinensprache JAVA Byte-Code exekutieren kann. Dieses Konzept sorgt für die außerordentlich hohe Portabilität von JAVA Byte-Code. Auf jedem Rechner auf dem eine virtuelle JAVA-Maschine läuft kann auch JAVA-Code ausgeführt werden. Der Speicherbedarf des JAVA-Microkernels ist außerordentlich niedrig - 40K für den Interpreter und den Support für die Basisklassen, zusätzlich noch 175K für die standard Klassenbibliotheken und den Multithreading - Support ! Das Problem des Portierens von JAVA-Programmen liegt also nicht beim Programmentwickler, sondern bei jenen die die virtuelle JAVA-Maschine portieren. Die Sprache C++ ist Sourcecode-portabel, JAVA ist Byte-Code-portabel. Jeder der schon einmal versucht hat ein im Sourcecode verfügbares Programm z.B. von einem UNIX-System auf ein anderes UNIX-System zu portieren, das C++-Programm „einfach“ auf einer anderen UNIX-Plattform zu übersetzen, der weis wie lange man oft zu kämpfen hat bis die richtigen Biliotheken sowie die Includepfade etc. in den Makefiles passen. Gegen diese Problematik können auch

„Imake“, „configure“, oder wie auch immer die Mechanismen zur Adaption des Sourcecodes an die Eigenheiten des jeweiligen Systems heißen mögen, kaum etwas tun.

Es existieren in der Sprachdefinition von JAVA keine implementationsabhängigen Aspekte.

Nach dieser längeren Einleitung über Systemumgebungen und die Einbettung des Softwareentwicklungsprozesses in die Systemumgebung nun zu den Konzepten der beiden Sprachen selbst.

Syntax

Die Syntax von JAVA ist sehr nahe an jene von C++ angelehnt. Dies geschah bewußt, da viele der derzeit tätigen Softwareentwickler mit C und C++ vertraut sind.

Einfachheit

Um JAVA einfacher benutzbar als C++ zu machen wurden verschiedene Sprachkonzepte von C++ nicht übernommen. Dazu gehören vor allem das Überladen von Operatoren und die Mehrfachvererbung. Das Überladen von Methoden ist in JAVA selbstverständlich möglich.

Dazu ein Zitat von B.Stroustrup dem Entwickler von C++

Weil jeder Vater und Mutter hat... -- comp.lang.c++

Für die meisten Leute war der Mechanismus der Mehrfachvererbung, die Möglichkeit, eine Klasse von mehreren Basisklassen abzuleiten, die neue Errungenschaft der Version 2.0 schlechthin. Ich war damals anderer Meinung, da ich das Gefühl hatte, daß die Verbesserungen im Typsystem von weitaus größerer praktischer Bedeutung waren.

Ich halte es sogar fast für einen Fehler, die Mehrfachvererbung schon mit der Version 2.0 vorgestellt zu haben.

Um das Memory-Management von C und C++ zu vereinfachen enthält JAVA einen automatischen „garbage collection“-Mechanismus. Das besonders fehleranfällige Anfordern und Freigeben von Speicherbereichen in C oder C++ (malloc(),...) entfällt. Das Konstrukt der Interfaces wurde aus Objective C in JAVA übernommen. Interfaces existieren in C++ nicht. So kann in der Praxis über Interfaces auch in JAVA ein zur C++ Mehrfachvererbung analoges Konzept benutzt werden.

Der freie Zugriff auf Datenbereiche über Zeiger ist in JAVA nicht implementiert. Dies führt zu einem sehr strikt implementierten Konzept des Information-Hiding. Im Umfeld des hardwarenahen Softwareentwurfs ist diese Spracheigenschaft restriktiv und vielfach hinderlich. Gibt es für eine Zielhardware keinen entsprechenden JAVA-Microkernel welcher alle benötigten Eigenschaften beinhaltet so muß man wohl oder übel eigene Methoden implementieren um spezielle Hardwarefeatures zu benutzen (Dies kann auch in anderen Implementationssprachen -z.B. in C geschehen).

Zuverlässigkeit und Qualitätsaspekte

Die Zuverlässigkeit von Programmen hängt entscheidend von einer frühen Erkennung und Beseitigung von Programmierfehlern ab. Die Daumenregel wonach die Beseitigung eines im momentanen Entwicklungsschritt nicht entdeckten Fehlers im nächsten Entwicklungsschritt das zehnfache kostet, gilt nahezu unabhängig von der verwendeten Entwicklungsumgebung und der Aufgabe des entwickelten Programms. So ist es nur allzu verständlich Fehler früh erkennen und beheben zu wollen.

Jede Sprache mit einem strengen Typsystem erlaubt weitreichende Tests der Datentypen und ihrer Verträglichkeit zur Übersetzungszeit. C++ übernimmt von C einige Lücken im Typsystem. Besonders die Überprüfung von Methoden- bzw. Funktionsaufrufen ist in C/C++ nicht sehr streng, wodurch sich Fehler fortpflanzen und zu Laufzeitproblemen führen.

Durch das in JAVA nicht implementierte offene Zeigerkonzept und die Fähigkeit des JAVA-Linkers viele jener Checks zu wiederholen die der Compiler schon durchgeführt hat können in JAVA auch so tiefliegende Probleme wie geänderte Methodenaktivierung in aufeinanderfolgenden Bibliotheksversionen etc. zur Übersetzungszeit bzw. zur Zeit des Linkens erkannt werden und führen erst gar nicht zu Laufzeitproblemen.

Verteilte Systeme

Die Unterstützung für die Entwicklung von Verteilten Systemen in einer Netzwrkumgebung ist in JAVA weit fortgeschritten. Verschiedene Protokolle der TCP/IP Familie sind fixe Bestandteile des JAVA API. JAVA Applikationen benutzen Protokolle wie HTTP und FTP, öffnen URLs wie lokale Dateien und besitzen einfachste Mechanismen um mit Datagrammen und IP-Sockets zu arbeiten.

Diese Eigenschaften sind in C++ nur über Klassenbibliotheken zu realisieren. Entwickelt man eine Applikation für verschiedene Plattformen so muß man natürlich auch die Klassenbibliotheken für sämtliche Plattformen zur Verfügung haben (kaufen).

Eine in JAVA auf einer Plattform entwickelte Netzwerkapplikation funktioniert mit sehr hoher Wahrscheinlichkeit ohne jede Änderung auf anderen Plattformen. Selbst Netzwerkapplikationen sind in JAVA architekturneutral, dies wäre in C++ wohl kaum zu schaffen.

Sicherheitsaspekte

Wo verteilte Systeme und offene Netzwerke im Spiel sind müssen Sicherheitsaspekte diskutiert werden. Es gibt eine starke Abhängigkeit zwischen Zuverlässigkeit und Sicherheit einer Implementationssprache. Ein Beispiel ist die Änderung in der Semantik der Zeigerbehandlung in JAVA. In den offenen Zeigerkonzepten von C und C++ ist es relativ einfach Zugriff auf fremde Datenstrukturen zu erlangen, man muß nur wissen wo im Speicher (an welcher Adresse) die Daten stehen und schon kann man sie verändern. Diese Tatsache ist für die Fortpflanzung vieler Computerviren von „vitaler“ Bedeutung. SUN Microsystems liefert dazu ein treffendes Beispiel einer fiktiven Firma - Archimedes Software Inc.

_ Someone wrote an interesting ``pat-ch'' to the PC version of the Archimedes system. They posted this patch to one of the major bulletin boards. Since it was easily available and added some interesting features to the system, lots of people downloaded it. It hadn't been checked out by the folks at Archimedes, but it seemed to work. Until the next April 1st, when thousands of folks discovered rude pictures popping up in their children's lessons. Needless to say, even though they were in no way responsible for the incident, the folks at Archimedes still had a lot of damage to control.

Eine wichtige Unterscheidung in den Sicherheitskonzepten ist jene zwische JAVA-Applets und Standalone-Applikatio-

1. von NetScape über das Netz geladene Applets

2. von NetScape lokal geladene Applets

3. vom Applet--Viewer über das Netz geladene Applets

4. vom Applet--Viewer lokal geladene Applets

5. mit dem Interpreter ausgeführte Applikationen

Operation	1	2	3	4	5
Lesen einer Datei in /mein/ordner (acl.read=/mein/ordner)	--	—	+	+	+
Lesen einer Datei in /mein/ordner (acl.read=null)	--	—	--	+	+
Schreiben einer Datei in /tmp (acl.write=/tmp)	--	—	+	+	+
Schreiben einer Datei in /tmp (acl.write=null)	--	—	--	+	+
Lesen der Dateiinformation (acl.read=/home/me acl.write=/tmp)	--	—	+	+	+
Lesen der Dateiinformation (acl.read=null acl.write=null)	--	—	--	+	+
Löschen einer Datei mit exec /uer/bin/rm	--	—	--	+	+
Löschen einer Datei mit File. delete ( )	--	—	-	—	+
Lesen des Feldes user. name	--	+	--	+	+
Netzverbindung zum WWW--Server der geladenen Seite	+	+	+	+	+
Netzverbindung zu einem Port am eigenen Rechner	--	+	--	+	+
Netzverbindung zu einem Port eines dritten Rechners	--	+	--	+	+
Laden einer Bibliothek	--	+	--	+	+
Programmbeendigung mit exit(--1)	--	—	--	+	+
Öffnen eines Popup--Fensters ohne Warnung	--	+	--	+	+

nen. Eine Standalone-Applikation hat im Vergleich zu einem Applet relativ wenige Restriktionen. Die folgende Tabelle zeigt eine Übersicht

Restriktionen für JAVA Applikationen

Die Restriktionen welche einem JAVA-Applet auferlegt sind, werden von einem Softwaremodul - dem Security-Manager - kontrolliert. Dieses Modul ist Teil eines Browsers und kann auch modifiziert bzw. abgeschaltet werden. An sich muß diese Modifikation falls sie von der BenutzerIn gewünscht wird an der lokalen Maschine im vollen Bewußtsein der daraus eventuell resultierenden Gefahren vorgenommen werden. Aber ... was ist schon lokal in unseren allgegenwärtigen Netzwerken ?

Tasks und Threads

JAVA beinhaltet Multithreading-Support im Standard API. Dazu gehören natürlich auch verlässliche Synchronisationsmechanismen. Diese beruhen auf dem Konzept von Überwachungs und Bedingungsvariablen von C.A.R. Hoare. Wichtig ist : Threads sind Teil des Standard API und somit standardisiert.

Damit wird die Entwicklung von hoch interaktiven Anwendungen wesentlich erleichtert. In C++ ist eine Unterstützung von Parallelverarbeitung direkt nicht vorgesehen

B.Stroustrup meint dazu:

Die Unterstützung von Parallelverarbeitung war seit jeher eine reiche Quelle für Erweiterungen und Bibliotheken. Einer der Gründe hierfür war die feststehende Gelehrtenmeinung, daß sich Mehrprozessorsysteme bereits in Kürze durchsetzen würden. So weit ich das beurteilen kann, herrscht dieser Glaube seit mindestens 20 Jahren.

...

Solch eine Spracheigenschaft, etwa in Analogie zu den Tasks von Ada, wäre fast allen Programmierern lästig.

Die Erstellung von Bibliotheken, die an die Bequemlichkeit und Leistungsfähigkeit sprachinterner Unterstützung heranreichen, ist durchaus möglich. Bibliotheken können eine Vielzahl von Modellen für die Parallelverarbeitung unterstützen. Somit ist dem Programmierer, der diese Vielfalt benötigt, besser gedient, als wenn er nur auf ein einziges integriertes Modell für die Parallelverarbeitung zugreifen könnte. Ich gehe davon aus, daß dieser Weg von den meisten Leuten eingeschlagen wird und die durch die Bibliothekenvielfalt entstehenden Probleme der Portierbarkeit über eine dünne Schnittstellen--Schicht gelöst werden können.

Beispiele für Bibliotheken zur Unterstützung von Parallelverarbeitung finden sich in [Stroustrup, 1980b], [Shopiro, 1987], [Faust, 1990] und [Perrington, 1990]...

Performance

Was die „Performance“ von JAVA anlangt sind zwei Fragestellungen interessant:

1. Wie schnell kann JAVA Bytecode interpretiert werden ?

2. Wie lange sind die Ladezeiten von Applets über ein Netzwerk ?

Der Bytecode wurde entwickelt um sehr einfach in Maschinencode der heute üblichen Prozessoren in Echtzeit übersetzt werden zu können. In den allermeisten Fällen ist das Laufzeitverhalten mehr als ausreichend. Bei Verwendung von Multithreading sind gewisse Limits durch das unter der virtuellen JAVA-Maschine liegende Betriebssystem gegeben. Auf einer SPARCStation 10 von SUN Microsystems sind laut Firmenangaben 300.000 Methodenaufrufe pro Sekunde bei interpretiertem Bytecode möglich.

Die Ladezeiten der Applets hängen stark von ihrer Größe ab. Bytecode ist extrem kompakt, wodurch sich oftmals erheblich kürzere Ladezeiten als bei compilierten C++ Applikationen ergeben.

Dazu ein kleines Beispiel:

Eine Variation on K&Rs „Hello World“ wurde in C++ und in JAVA als Applet codiert. Die C++ Variante wurde mit Borland C++ 4.52 übersetzt die JAVA Variante mit javac aus dem JAVA-Development Kit 1.0.

```
// Hello.cpp
#include
      H
main()
{
cout < „Hello World“ < endl;
}
hello.java
import java.applet.Applet;
import java.awt.Graphics;

public class hello extends java.applet.Applet
{
      public void paint( Graphics g )
      {
            g.drawString( „Hello, World!“, 50, 25 );
      }
}
```

Das aus dem C++ Sourcecode entstandene hello.exe ist ca. 91Kbyte groß, das JAVA-Applet hello.class ist 411Byte groß ! Zugegeben dies ist ein extremes Beispiel und der Overhead an Bibliotheksroutinen die statisch gelinkt werden müssen ist hier besonders groß aber die Tendenz ist eindeutig. Ein Blick in die im JAVA Development Kit enthaltenen Demos zeigt deutlich die Kompaktheit des Bytecode. Kleinere Beispiele mit ca. 300 bis 400 Zeilen Sourcecode (ohne Kommentare) ergeben in den meisten Fällen weniger als 10Kbyte Bytecode.

JAVA als Anfängersprache

Zum Abschluß noch einige Worte zur Diskussion über eine günstige Einsteigersprache. In der Abteilung Elektronik am TGM wird derzeit C bzw. C++ als Einstiegssprache unterrichtet. Die Schüler der Abteilung gewöhnen sich relativ rasch an die Eigenheiten dieser Sprachen und erreichen in den höheren Jahrgängen ein Niveau auf dem sie in der Lage sind auch maschinennahe Hochsprachenprogrammierung zu betreiben. Die üblichen Probleme mit dem Verständnis von Zeigern sollen hier keinesfalls verschwiegen werden. Am Fachhochschul-Studiengang Elektronik am TGM wird C++ konsequent als Einstiegssprache eingesetzt. Die Studenten kämpfen mit ähnlichen Problemen wie die Schüler an der HTL. Meines Erachtens haben die HTL-Schüler einen Vorteil durch ihre jugendliche Unbekümmertheit, die ihnen hilft im trial and error - Verfahren die dunklen Abgründe von C++ zu erforschen.

Am Kolleg Multimedia am TGM wurde erstmals versucht JAVA als Einstiegsprogrammiersprache einzusetzen. Die Zeit zum Erlernen der grundlegenden Begriffe ist - wie immer in Kollegs - stark begrenzt. So wurde versucht eher spielerisch anhand von Beispielen an die Sache heranzugehen und im Laufe dieses Sommersemesters zeichnen sich erste Erfolge ab. Im Anschluß an diesen Artikel findet sich ein Biorythmus-Applet von Michael Pranger, einem Studenten des Kollegs. Einige Studentengruppen des Kollegs basteln derzeit an einem internetfähigen Geschicklichkeitsspiel. Wir werden am Semesterende über die Erfolge damit berichten.

Grundvokabeln zur Benutzung eines Computers

1 Bit *bekanntes Pils aus der Eifel*

1 Byte *8 dieser Biere*

1 Kilobyte *etwa 27 Hektoliter Bit*

3 1/2 Zoll *drei Zollbeamte + ein kleinwüchsiger, strafversetzter Schupo*

5 1/2 Zoll *fünf Zollbeamte + ein Drogenhund*

BILDschirm *Regenschutz mit Springerreklame*

Bus *öffenttliches Nahverkehrsmittel*

CD *Körperpflegeserie*

Chip *Spielcasino-Geld oder Knabberei aus Kartoffeln*

Commodore *Offizier der Luftwaffe*

Controller *Eltern, Lehrer, usw.*

Datei *Ei mit aufgedrucktem Legedatum*

Directory *englisch für Direction*

EDV *Ende der Vernunft (Abk.)*

File *Werkzeug zur Bearbeitung von Fingernägeln*

Freezer *englisch für Gefriertruhe*

Hardware *Granit, Diamant, 8-Minuten-Ei*

Hacker *Arbeiter mit Axt*

Interface *Fahndungsfotakartei von Interpol*

Lichtstift *Elektrolehrling*

Lightpen *Leichter Schlaf*

Mailbox *Schlägerei zwischen Postbeamten*

Maus *bissiges, hochgefährliches Raubtier*

MS-DOS *Motorschiff mit Namen DOS*

MSX *Motorschiff, inkognito unterwegs*

Mikroprozessor *sehr kleiner Staatsanwalt*

Monitor *politisches Fernsehmagazin*

Port *Portwein (Kurzform)*

RAM *Milchprodukt*

ROM *Hauptstadt Italiens*

Schnittstelle *Wursttheke, Friseur, Verletzung*

Software *Gummibärchen, Softeis, 3-Minuten-Ei*

Space-Bar *Weltraumkneipe*

SYSOP *Säuft Yankeewhiskey Ständig Ohne Peilstab (Abk.)*