Java feeling

Komprimiert und übersetzt[1] von Walter Riemer

Generell wurden dem Übernehmen bewährter Eigenschaften anderer Sprachen gegenüber dem Implementieren vielversprechender neuer Ideen der Vorzug gegeben. Java war von Anfang an eine praxisorientierte Sprache und nicht eine im Sinne einer wissenschaftlich fundierten Neuentwicklung, obwohl sehr wohl auch informatikwissenschaftliche Studien betrieben wurden. Folgende Eigenschaften wurden als bewährt von Vorgängersprachen übernommen:

(1) Objektorientierung, wie in C/C++ und Smalltalk; (2) Bindende Standards für numerische Zahlenformate und Rechnen, IEEE 754-Format; (3) Nähe zur Systemprogrammierung wie unter C.

Neu ist jedoch die „verteilte" („distributed") Natur: Es gibt keine Grenzen zwischen den Maschinen. Der eigene Computer begrenzt nicht die Applikation; vielmehr kann man sich mittels Java im ganzen Netzwerk frei bewegen.

Ein wesentliches Ziel war es dementsprechend, Verhalten beschreibende Einheiten zu entwerfen, die überallhin übertragbar waren, aber von den zugehörigen Daten getrennt waren (also ein deutlicher Widerspruch zum Klassenkonzept von C/C++). Wozu muß im Netz jedes übertragene JPEG-Bild seinen eigenen Dekompressor mitschleppen? Allerdings mußten die Daten mit „Tags" versehen sein, welche den Typ bzw. die anwendbaren Routinen eindeutig zuzuordnen erlaubten. Falls die nötige Klarheit dadurch noch nicht geschaffen war, mußte der Client in der Lage sein, die erforderliche Codeimplementierung vom Server anzufordern. So können wesentlich schlankere Systeme entworfen werden, da der Client jederzeit lernfähig ist und sich benötigte Information holen kann; der Kern versteht das zugrundeliegende Prinzip und ist einfach.

Größter Wert wurde darauf gelegt, Java so sauber wie möglich anzulegen und mögliche Programmierer-Tricks zu beschränken. Eine virtuelle Maschine versteht Java-generierten Code als P-Code, der relativ leicht interpretierbar ist; dieser wird erst in Maschinencode übersetzt. Der Compiler führt viele Überprüfungen aus und gibt Fehlermeldungen, aber keine Warnungen aus: Erfahrung hat gezeigt, daß Warnungen so wie ernste Fehler meist zu nicht lauffähigem oder zumindest unzuverlässigem Code führen; daher wurde auf sie verzichtet. Auf Gültigkeitsbereiche von Namen wurde bewußt verzichtet (Namen von Variablen in inneren und äußeren Blöcken dürfen nicht übereinstimmen): Programmierfehler wurden allein dadurch deutlich reduziert!

Großer Wert wurde auf verläßliches „Reinemachen" („garbage collection") gelegt: die Erfahrung zeigt, daß Programmierern immer wieder ins Nirwana oder außerhalb eines Arrays zeigende Zeiger übrigbleiben, ebenso wie angeforderte Speicherbereiche nicht wieder freigegeben werden. Java kümmert sich um all dies verläßlich.

Zeiger sind nicht so freizügig verwendbar wie in C/C++:

```
((int *) Zeiger)[Index],
```

womit praktisch alles angesprochen werden kann, ist nicht erlaubt; in größeren Programmsystemen sind sie oft die Wurzel nicht funktionierender Systeme, ohne daß der Fehler leicht zu finden wäre.

Die Ausnahmebehandlung wurde weitgehend von Modula 3 übernommen und hat sich mitsamt ihrer Strenge bewährt, weil in jeder Echtanwendung unerwartete Vorgänge vorkommen. Zwar zwingt dies den Anwender, über Dinge nachzudenken, die er lieber ignorieren würde, aber dafür wird sein Programm weitaus verläßlicher.

Objektorientierte Erweiterbarkeit war ein wesentliches Entwicklungsziel. Dafür wurde (wie in LISP) „late Binding" (Binden zur Laufzeit) vorgesehen. Unter Beachtung einer gewissen Disziplin können Methoden hinzugefügt und private Variablen entfernt werden, nur beim Entfernen scheinbar unbenützter Methoden sollte man vorsichig sein.

Java wird von vielen Anwendern wie eine einfache Entwurfssprache (im Sinne von „Darauflosschreiben") angewendet, obwohl Java hinsichtlich Typdefinition sehr streng ist; dafür werden aber viele Fehler vermieden oder angezeigt, fast wie in PASCAL. Das dynamische Linken („late binding") erleichtert Wartung und Erweiterung der Software, auch hinsichtlich Handhabung neuer Datentypen. Viele Entscheidungen (insbesondere hinsichtlich Layouts von Objekten) werden in die Laufzeit verlagert. Benötigte Datentypen können schon existieren (und durch „Lookup" aufgefunden werden) oder auch neu erzeugt werden: für ersteres steht ein „factory"-Zugang (statische Methode) zur Verfügung (Type.new), für letzteres ein Konstruktor-Zugang wie in C++ (new Type). All dies trägt zur Portabilität bei.

Ein Entwicklungsziel war auch eine mit C vergleichbar gute Leistung („Performance"). Die strenge Typprüfung hilft dabei gewaltig und vermeidet zusätzlich viele Programmfehler. Heute ist C diesbezüglich fast eingeholt: eine Skriptsprache hat die Leistungsfähigkeit einer guten klassischen Sprache. Weitere Leistungssteigerungen sind noch möglich durch Assembler-geschriebenen Interpreter und echten Maschinencode erzeugende Compiler. Darüber hinaus hat Java eine reichhaltige Klassenbibliothek, während die Sprache selbst recht einfach ist.

Wie also fühlt sich Java an? Verspielt und flexibel; man kann flexible Sachen damit erzeugen, trotzdem ist alles deterministisch: man bekommt, was man will und ist nicht von Eigenleben abhängig. Wenn man etwas ausprobiert, erhält man rasch entsprechende Antworten. Dank der großen Klassenbibliotheken ist Java leistungsfähig. Alles in allem, man hat das Gefühl, man setzt sich hin und schreibt Code.

1 Original: "The Feel of Java", Autor Gosling, James; Erschienen in (IEEE) 6/1997. Der Autor ist der Chefentwickler für Java bei Sun Microsystems. Das Projekt wurde 1991 mit dem Ziel begonnen, verteilte Steuerungen von Geräten der Unterhaltungselektronik zu studieren, wobei auch intensive Kontakte mit Personen stattfanden, die TV- und VCR-Geräte entwickelten. Deren Prioritäten waren völlig andere als jene der Computer-Leute, nämlich insbesondere sichere Netzportabilität und geringe Kosten gegenüber Kompatibilität. Die Notwendigkeit ergab sich, Software plattformunabhängig entwickeln zu können; auch die recht gute Portabilität von C/C++ reichte nicht mehr aus, zum Beispiel weil gewisse Vorgaben noch freibleibend waren (wie etwa int-Format und float-Format).