K. Berg, B. Marek, J. Sametinger

Java in der praktischen Anwendung

In diesem Artikel werden Stärken und Schwächen von Java aufgezeigt und beurteilt. Dadurch soll eine Evaluierung von Java im Hinblick auf die praktische Einsetzbarkeit in der Softwareentwicklung ermöglicht werden. Wir geben einen kurzen Überblick über Java und diskutieren Stärken und Schwächen. Zusätzlich erläutern wir anhand einer Checkliste verschiedene Punkte, die vor einem Einsatz von Java berücksichtigt werden sollen. Die wichtigsten Ergebnisse sind am Ende des Artikels zusammengefaßt.

1. Einleitung

Im Jahre 1995 wurde von Sun Microsystems die objektorientierte Programmiersprache Java der Öffentlichkeit vorgestellt [Sun97]. Die Sprache war Ergebnis mehrjähriger Forschungsarbeiten bei Sun und zeichnet sich insbesondere durch ihre Unterstützung der Programmierung von Internet-Anwendungen aus. Zu diesem Zeitpunkt verzeichnete die praktische Bedeutung des Internet, insbesondere wegen des *World Wide Web* (WWW), einen raschen Anstieg. Gleichzeitig wurde *HotJava* vorgestellt, ein in Java implementierter Web-Browser mit integriertem Java-Interpreter. Damit war es möglich, Programme in WWW-Seiten einzubinden und automatisch über das Internet zu laden und auszuführen. Solche Programme werden *Applets* genannt. Die Bandbreite möglicher Anwendungen des WWW vergrößerte sich sprunghaft. Zusätzlich gelang es den Entwicklern von Java durch die Anlehnung an C und C++, sowie der Ausmerzung von Unzulänglichkeiten dieser Sprachen, eine hohe Akzeptanz zu erzielen. Java erfuhr somit über das Internet weltweit eine einzigartig rasche Verbreitung und Etablierung.

Der Erfolg von Java basiert nicht zuletzt darauf, daß durch die Definition einer *virtuellen Maschine* Java-Programme plattformunabhängig erstellt werden können. Jedes Java-Programm wird vom Compiler nicht wie sonst üblich in Maschinencode (*object code*), sondern in einen maschinenunabhängigen Bytecode übersetzt, der dann auf verschiedenen Plattformen, auf denen eine virtuelle Java-Maschine verfügbar ist, ausgeführt, d.h. interpretiert werden kann. Um Effizienzeinbußen durch die Interpretation zu verringern, wird der Bytecode oft unmittelbar vor der Ausführung, also zur Laufzeit, in echten Maschinencode übersetzt. Man spricht in diesem Fall von einem *Just-in-Time*-Compiler (JIT). Klassenbibliotheken, z.B. für graphische Benutzerschnittstellen, werden ebenfalls auf allen Plattformen mit identischen Schnittstellen angeboten, um die Portabilität von Java-Programmen zu gewährleisten.

Im folgenden werden wir verschiedene Aspekte von Java unter die Lupe nehmen. Dazu zählen die Sprache, Klassenbibliotheken, Entwicklungsumgebungen, Standardisierung, Kosten, Performance, Ressourcenverbrauch und die Anbindung von fremdem Code. Dabei wird versucht, Stärken und Schwächen in diesen Bereichen aufzuzeigen.

2. Die Sprache

Java ist eine objektorientierte Programmiersprache, die sich konzeptionell nicht wesentlich von anderen Sprachen dieser Kategorie unterscheidet. Dies hat zunächst den Vorteil, daß die Sprache leicht von Leuten mit Erfahrung in objektorientierter Programmierung erlernt werden kann. Die Anlehnung an die Sprachen C und C++ erleichtert überdies den Umstieg für Programmierer, die bereits diese Sprachen verwenden. Eine Umschulung beschränkt sich dann in erster Linie auf die verfügbaren Bibliotheken. Dadurch können Umschulungskosten niedriger gehalten werden als dies sonst der Fall wäre.

Die automatische Speicherfreigabe sowie der explizite Ausschluß von Zeigern bzw. Adressen und der Zeigerarithmetik tragen viel zur Robustheit von Java-Applikationen bei. Mechanismen wie die automatische Speicherfreigabe oder die Ausnahmebehandlung haben sich schon in anderen Programmiersprachen bewährt. Der Vorteil von Java ist, daß die Sprache mit einer klaren

Zielsetzung konzipiert wurde und daß fehlerträchtige Konzepte bewußt weggelassen wurden. Aus softwaretechnischer Sicht ist dies als Fortschritt zu bezeichnen.

3. Klassenbibliotheken

Die Mächtigkeit von Programmiersprachen wird zunehmend weniger wichtig, da mit Bibliotheken flexibler auf verschiedene Bedürfnisse reagiert werden kann. Für Java gibt es eine Klassenbibliothek, die in mehrere Pakete unterteilt ist [Fla96]. Man unterscheidet zwischen den Kernpaketen (*standard* oder *core APIs*), die schon relativ gut ausgereift sind, und erweiterten Paketen (*extended APIs*), die noch umfangreicheren Änderungen unterliegen könnten. Diese Pakete haben plattformunabhängige Schnittstellen, aber plattformabhängige Implementierungen. Wenn ein Java-Programm von einer virtuellen Maschine interpretiert wird, dann werden die jeweiligen Pakete der entsprechenden Implementierung für diese Plattform dazu verwendet. Neben den vordefinierten Paketen gibt es auch Pakete von anderen Herstellern, die nicht notwendigerweise auf allen Plattformen verfügbar sein müssen, wenn sie nicht ausschließlich in Java implementiert sind.

Die Kernpakte unterstützen die Ein-/Ausgabe, Netzzugriffe, graphische Benutzerschnittstellen, die Erstellung von Applets und bieten sonstige nützliche Klassen, z.B. Zeichenketten, Container-Klassen. Der Reifegrad bzw. der Umfang der Pakete ist noch unterschiedlich. Nach unseren Erfahrungen war beispielsweise das AWT in der Version 1.0.2 noch enttäuschend, weil es zum einen noch zuwenig Funktionalität bot und zum anderen auf verschiedenen Plattformen zum Teil verschiedenartiges Verhalten nach sich zog. Diese Mängel sind in der Folgeversion zum Teil behoben, eine Erweiterung der Funktionalität ist aber durchaus noch wünschenswert. Von unterschiedlichen Herstellern werden mittlerweile auch Bibliotheken für verschiedene Aufgabenbereiche zur Verfügung gestellt, z.B. für numerische Berechnungen oder für Animationen [Jav97].

4. Standardisierung

Durch die Lizenzierungspolitik von Sun entstand quasi ein de-facto-Standard, da sich alle Lizenznehmer an Vorgaben von Sun halten müssen. Dieser de-facto-Standard ist allerdings an eine einzige Firma gebunden, obwohl Sun versucht, im Einklang mit anderen Firmen Weiterentwicklungen vorzunehmen. Nunmehr zeichnet sich ab, daß für Java auch ein internationaler Standard etabliert werden soll. Sun strebt eine Standardisierung bei ISO (*International Organization for Standardization*) an. Es hat sich jedoch Widerstand gegen dieses Vorgehen ergeben, da man Sun unterstellt, von der Standardisierung nur profitieren, aber keine Rechte abgeben zu wollen. Zu den massiven Kritikern von Suns Bestrebungen gehört insbesondere auch Microsoft. Zur Zeit ist noch keine endgültige Entscheidung gefallen, aber es ist mehr als ungewiß, daß der von Sun derzeit verfolgte Weg zum gewünschten Ergebnis führen wird.

In diesem Zusammenhang ist die Strategie von Microsoft interessant. In der von Microsoft angebotenen Bibliothek werden die von Sun geforderten Schnittstellen zur Verfügung gestellt. Darüber hinaus werden aber viele zusätzliche, für die Programmierung von Windows-Programmen nützliche Klassen angeboten. Die Verwendung dieser zusätzlichen Klassen mindert die Portabilität der damit entwickelten Systeme. Entscheidend wird sein, ob Entwickler Plattformunabhängigkeit oder bestmögliche Unterstützung einer weit verbreiteten Plattform als wichtiger ansehen. Microsoft könnte bei entsprechend großer Beliebtheit der Microsoft-Klassen eventuell vom Java-Zug abspringen und (unter anderem Namen, z.B. J++) eigene Entwicklungen verfolgen. Wie die juristische Sachlage für diesen Fall genau aussieht entzieht sich unserer Kenntnis. Wir gehen aber davon aus, daß Microsoft bei Bedarf Mittel und Wege finden wird, um eigene Wege zu gehen.

5. Entwicklungsumgebungen und Werkzeuge

Es gibt bereits eine Fülle von Entwicklungsumgebungen auf allen gängigen Hardwareplattformen, die unterschiedlich ausgereift und komfortabel sind. Der Werkzeugmarkt ist sehr dynamisch. Fast ständig gibt es neue Anbieter oder neue Versionen von bestehenden Werkzeugen, so daß es schwierig ist, ein aktuelles Bild von längerer Gültigkeit zu zeichnen. Beispiele für

Entwicklungsumgebungen finden sich in [Pie96, Pie97]. Die angeführten Produkte haben unterschiedliche Funktionalität. Eine entsprechende Evaluierung ist vor dem Einsatz von Java unumgänglich.

6. Kosten

Für den Einsatz von Java zur Softwareentwicklung ist die Anschaffung einer Programmierumgebung, einer Java-Klassenbibliothek (nicht unbedingt notwendig, aber sinnvoll), sowie einer virtuellen Java-Maschine notwendig. Zur Entwicklung von Applets benötigt man zusätzlich ein System, das die Anzeige solcher Applets erlaubt, z.B. diverse WWW-Browser. Typischerweise umfaßt der Kauf eines Java-Pakets eine Programmierumgebung mit Editor, Compiler und Debugger, sowie Werkzeugunterstützung zur Erstellung von graphischen Benutzerschnittstellen und Klassenbibliotheken. Um Java-Applikationen ausführen zu können, braucht man für seine Plattform eine virtuelle Maschine und die Kernpakete der Bibliothek. Diese werden üblicherweise gemeinsam angeboten. Derzeit werden virtuelle Maschinen und die Kernpakete kostenlos von verschiedenen Herstellern abgegeben. Es ist auch zu erwarten, daß diese in zukünftigen Betriebssystemversionen schon integriert sein werden. Als Anbieter von Java-Applikationen bzw. Applets kann man davon ausgehen, daß der Benutzer die notwendige Infrastruktur bereits verfügbar hat. Daß in diesem Zusammenhang in Zukunft Kosten anfallen ist unwahrscheinlich, kann aber nicht ausgeschlossen werden.

7. Sicherheit

Die Verteilung von Programmen über das Internet stellt besondere Anforderungen an die Sicherheit solcher Programme. Mögliche Attacken von über das Netz geladenen Programmen betreffen die Integrität, die Verfügbarkeit, die Vertraulichkeit oder diverse Belästigungen [Sur96]. Es gibt unterschiedliche Möglichkeiten, solche Attacken durchzuführen, z.B. mit trojanischen Pferden. Dabei führt ein Applet eine offizielle und eine versteckte Operation durch, z.B. sendet ein Applet unbemerkt Dateien über das Netz, während es eine Animation auf dem Bildschirm anzeigt oder mit dem Benutzer Schach spielt. Vom Java-Compiler und von der *Java Virtual Machine* werden verschiedene Sicherheitsstufen realisiert, um solche Attacken zu verhindern. Es stehen mehrere Mechanismen zur Verfügung, um fehlerhaften und/oder inkorrekten Code zu erkennen und dessen Ausführung zu verhindern [GM95].

Das Sicherheitspaket von Java definiert Schnittstellen, zu denen verschiedene Implementierungen verwendet werden können, z.B. die von Sun. So hat der Benutzer die Freiheit, in diesem kritischen Bereich nach eigenem Belieben unterschiedliche Algorithmen und/oder Implementierungen (sofern verfügbar) auszuwählen. In einer Java-Implementierung wird das Sicherheitspaket daher separat installiert und kann auch dynamisch verändert werden (jedoch nicht von *remote applets*). Der Begriff *Sandbox Model* widerspiegelt die Tatsache, daß Applets durch diese Maßnahmen in ihren Aktivitäten eingeschränkt sind und praktisch nur in einem geschützten und abgeschlossenem Raum (Sandbox) operieren und außerhalb dieses Raumes keine Aktivitäten durchführen können. Da die Herkunft von signiertem Javacode eruierbar ist, können Applets mit unterschiedlichen Rechten ausgestattet werden. So kann lokalen Applets oder solchen von vertrauenswürdiger Herkunft der Zugriff auf Dateien gestattet werden, während man allen anderen diese kritische Aktion verbietet. Grundsätzlich können auch Applikationen mit eingeschränkten Rechten ausgestattet werden. Damit kann beispielsweise verhindert werden, daß sich dynamisch nachgeladener Code, der von unsicheren Quellen stammt, bösartig verhält. Auch der Benutzer kann eine Applikation in ihren Rechten nachträglich einschränken.

Trotz der getroffenen Sicherheitsvorkehrungen gibt es nach wie vor kleine Sicherheitslücken. Das liegt in erster Linie an Fehlern in den Implementierungen. Es ist damit zu rechnen, daß immer wieder Sicherheitslücken entdeckt und in nachfolgenden Versionen behoben werden. Die Neuheit der Probleme trägt ebenfalls dazu bei, daß nur sukzessive Verbesserungen möglich sein werden. Vorhandene Sicherheitslücken können aber nicht der Sprache Java als solcher, sondern allenfalls den verwendeten Browsern bzw. deren *Virtual Machines* angelastet werden. Andere Programmiersprachen haben diese Sicherheitsprobleme auch deswegen nicht, weil mit ihnen ein Austausch von Code über das Netz auf diese Art und Weise gar nicht möglich ist.

8. Effizienz und Ressourcenverbrauch

Grundsätzlich bewirkt die Verwendung einer plattformunabhängigen virtuellen Maschine mit Interpretation eines Bytecodes wesentlich langsamere Ausführungszeiten als die Ausführung von reinem Maschinencode (um einen Faktor bis zu 20). Durch den Einsatz von Übersetzern während der Laufzeit (Just-In-Time-Compiler) können jedoch Effizienzeinbußen stark verringert werden. Eine Verlangsamung von Javacode um einen Faktor bis zu 2 im Unterschied zu C ist realistisch, da in Java aus Sicherheitsgründen viele Überprüfungen durchgeführt werden, die bei C nicht üblich sind. Wenn mehrere Applets in einem WWW-Browser angezeigt bzw. ausgeführt werden, dann geschieht dies üblicherweise mit nur einer virtuellen Maschine. Die Applets laufen in verschiedenen *Threads*. Gelangen mehrere Applikationen gleichzeitig zur Ausführung, dann hat jede davon eine eigene virtuelle Maschine, was einen hohen Ressourcenverbrauch verursachen kann. Durch die Konzepte von Java ist es aber auch möglich, Applikationen in verschiedenen *Threads* mit nur einer virtuellen Maschine auszuführen. *Benchmarks* von verschiedenen Java-Implementierungen gibt es in [Pen97].

9. Anbindung von C-Code

Viele Vorteile von Java kommen nur zur Geltung, wenn ein System ausschließlich in Java implementiert wurde, z.B. Plattformunabhängigkeit. Dieses Ziel wird nicht immer realisierbar sein, da beispielsweise eine Applikation speziell für eine Plattform entwickelt wird und/oder bereits in anderen Sprachen entwickelter Code wiederverwendet werden soll. Zur Anbindung von nicht in Java implementiertem Code über eine offene Schnittstelle wird das *Java Native Interface* (JNI) vorgegeben [Sun96]. Der Programmierer kann eine einzige Schnittstelle für die Anbindung von Fremdcode auf allen Plattformen verwenden. Bei zeitkritischem und plattformabhängigem Code kann aber auch, sofern vorhanden, auf plattformabhängige Schnittstellen zurückgegriffen werden. Mit JNI sind Aufrufe in beide Richtungen möglich, d.h. man kann aus Java C-Code aufrufen und auch von C aus Methoden von Java benutzen. Damit die automatische Speicherbereinigung keine Objekte aufräumt, auf die in Nicht-Java-Code noch verwiesen wird, wird zwischen lokalen und globalen Referenzen auf solche Objekte unterschieden. Es liegt in der Verantwortung des Programmierers, daß auf keine schon aufgeräumten Objekte zugegriffen wird.

10. Entwicklungszeiten

Konkrete Aussagen bzw. Vergleiche von Entwicklungszeiten (Einarbeitung, Übersetzungszeiten, etc.) liegen derzeit nicht vor. Wir gehen aber davon aus, daß nach erfolgter Einarbeitung die Produktivität, beispielsweise gegenüber C++, um einiges höher sein wird, da viele Fehlerquellen von vornherein ausgeschlossen sind, und so das Suchen von daraus resultierenden Fehlern wegfällt. Produktivitätssteigerungen um einen Faktor 2 bis 4 verbunden mit der Erstellung von Code höherer Qualität scheinen durchaus realistisch zu sein [Orc97].

11. Check-Liste für den Einsatz von Java

Die Weiterentwicklung von Java schreitet rasant voran, so daß jedes Dokument darüber in wenigen Wochen veraltete Informationen enthalten kann. Ein ständiges Beobachten der unterschiedlichen Entwicklungen ist deshalb ratsam. Für den Einsatz von Java in der Produktentwicklung ist wichtig, daß eine gewisse Stabilität bei Sprachdefinition, virtueller Maschine, Klassenbibliothek, etc. eintritt, daß vernünftige Werkzeuge zur Softwareentwicklung auf der gewünschten Plattform zur Verfügung stehen, daß eine Weiterentwicklung und Betreuung von Sprache und Werkzeugen gegeben ist, daß die Lizenzkosten gering sind und daß keine Abhängigkeiten zu anderen Firmen entstehen. Einige dieser Punkte sind schon relativ klar, während andere vor einem tatsächlichen Einsatz von Java noch konkret geprüft werden sollten. Im folgenden geben wir eine Liste mit kurzen, prägnanten Fragen an. Die Beantwortung dieser Fragen soll helfen, einen möglichen Einsatz von Java besser abzuschätzen.

- ***Applet/Applikation***: Ist für mein konkretes Problem eine Implementierung als Applet oder Applikation vorzuziehen?
- ***Beans***: Ist die Komponententechnologie mit JavaBeans bereits ausgereift genug, um eine Entwicklung damit in Angriff zu nehmen? Derzeit ist dies noch nicht der Fall.

- ***Bibliotheken***: Werden am Markt Bibliotheken angeboten, die für die geplante Entwicklung eingesetzt werden können? Sind diese plattform(un)abhängig?
- ***JIT-Compiler***: Ist ein JIT-Compiler notwendig für eine ausreichende Performance der konkreten Anwendung oder ist eine Interpretation des Bytecodes ausreichend?
- ***JVM***: Lohnt es sich bzw. ist es notwendig, eine eigene *Java Virtual Machine* mit den Kernpaketen für die Zielmaschine(n) zu entwickeln?
- ***Lizenz***: Wie hoch sind die entstehenden Lizenzkosten?
- ***Mängel***: Gibt es Teile von Java, die benötigt werden aber noch zu unausgereift sind, um damit ein Produkt zu entwickeln? (z.B. Mängel in AWT, Anbindung von Datenbanken)
- ***Microsoft***: Ist Microsoft noch auf der Java-Schiene, oder werden dort bereits eigene Pläne verfolgt? Falls Microsoft eigene Wege geht: Ist es besser Microsoft zu folgen? (Dies könnte sein, wenn z.B. die Zielplattform ausschließlich Windows ist.)
- ***Netzwerk***: Ist das Netzwerk der Anwender stabil genug für eine Appletlösung?
- ***Performance***: Ist die Performance ausreichend für die geplante Anwendung und Zielmaschine(n)?
- ***Pilotprojekt***: Gab es bereits ein Pilotprojekt, in dem erste Erfahrungen gesammelt und Java-Know-How aufgebaut werden konnte?
- ***Plattform***: Gibt es für die Zielplattform meiner Anwendung eine Java-Implementierung (Java Virtual Machine) und die notwendigen Kernpakete?
- ***Realtime-Anwendungen***: Sind die Voraussetzungen zur Entwicklung von Realtime-Anwendungen erfüllt? (Derzeit ist dies noch nicht der Fall.)
- ***Referenzen***: Gibt es bereits vergleichbare, in Java implementierte Produkte (Domäne, Größe)?
- ***Ressourcen***: Ist der Ressourcenverbrauch hinreichend für die geplante Anwendung und Zielmaschine(n)?
- ***Sand Box Model***: Sind die Rechte von Applets ausreichend für eine Appletlösung?
- ***Sicherheit***: Sind die Sicherheitsvorkehrungen ausreichend/relevant für das geplante Produkt?
- ***Speicherbereinigung***: Stellt die automatische Speicherbereinigung für die geplante Anwendung ein Problem dar?
- ***Standardisierung***: Ist eine internationale Standardisierung bereits erfolgt, bzw. in Sichtweite? Wird sie für die geplante Entwicklung als notwendig erachtet?
- ***Systemprogrammierung***: Sind systemnahe Teile zu entwickeln, die in Java Probleme verursachen können und in einer anderen Sprache implementiert werden sollten?
- ***Umschulung***: Können die Mitarbeiter zeitgerecht in der neuen Technologie geschult werden?
- ***Werkzeugunterstützung***: Gibt es ausreichend Werkzeugunterstützung für eine kommerzielle Softwareentwicklung?

12. Zusammenfassung

Die Programmiersprache Java besitzt eine Fülle von Eigenschaften, die sie auch für einen kommerziellen Einsatz als Alternative zu anderen Sprachen in den Vordergrund rückt. Java besticht durch Eigenschaften, die in anderen Sprachen oft fehlen, die aber aus softwaretechnischer Betrachtung eigentlich unverzichtbar sind. Dazu zählen Einfachheit, leichte Erlernbarkeit, Sicherheit, Robustheit und insbesondere auch Plattformunabhängigkeit. Obwohl Java von nur einer Firma entwickelt wurde, hat es in vielen führenden Firmen der Softwarebranche Akzeptanz gefunden. Bei Weiterentwicklungen sind mittlerweile viele Unternehmen in Entscheidungsprozesse eingebunden, und es werden von unterschiedlichen Herstellern zahlreiche Werkzeuge für Java angeboten.

Neben Applets und typischen Applikationen sind mit Java auch Anwendungen auf spezieller Hardware möglich, z.B. Unterhaltungsgeräte. Verteilung spielt für Java ebenfalls eine essentielle Rolle und wird besonders unterstützt. Lediglich bei Realtime-Anwendungen gibt es Defizite. In diesem Bereich ist Java noch keine Alternative zu anderen Programmiersprachen. In Umfragen geben viele der großen Softwarefirmen bekannt, daß sie in Zukunft verstärkt Java einsetzen

wollen. Aus softwaretechnischer Sicht bedeutet Java einen deutlichen Schritt vorwärts. Zusätzlich läßt sich ein verstärkter Trend zu komponentenorientierter Softwareentwicklung erkennen, der von den Java-Entwicklern erkannt und verstärkt unterstützt wird (Stichwort *JavaBeans*).

Der erfolgreiche Einsatz von Java für die Entwicklung kommerzieller Produkte hängt auch von der Verfügbarkeit komfortabler Werkzeuge für die gewünschte Plattform ab. Eine Evaluierung der Werkzeuge ist vor einem Einsatz von Java notwendig. Außerdem sollte klar sein, wie wichtig eine Standardisierung ist, und ob darauf gewartet werden soll. Einen wichtigen Punkt stellt auch die Verfügbarkeit von Bibliotheken dar. Es muß geklärt werden, ob die derzeit vorhandenen Bibliotheken ausreichend in ihrer Funktionalität und Qualität für den ins Auge gefaßten Anwendungsbereich sind, um die Entwicklung eines Produktes genügend zu unterstützen.

Empfehlung: Wir empfehlen einen sukzessiven Einstieg. Das heißt, mit einem oder mit mehreren kleineren Projekten sollte zunächst Java-Know-How aufgebaut werden. Damit lernen die Projektbeteiligten die Stärken und Schwächen im Detail kennen und können die Entwicklungen von Java mitverfolgen. Ein günstiger Zeitpunkt für einen Umstieg auf Java bei der Produktentwicklung kann dann leichter festgelegt werden. Zunächst sollten solche Erfahrungen mit Applets und kleineren Applikationen gemacht werden. Für größere Anwendungen bzw. Realtime-Applikationen ist der Einsatz von Java noch nicht zu empfehlen.

Referenzen

[Fla96] David Flanagan. *Java in a Nutshell*. O'Reilly & Associates, Inc., 1996.

[GM95] James Gosling, Henry McGilton. *The Java programming language*. OOPSLA '95 Tutorial Notes, Austin, TX, 1995.

[Jav97] JavaWorld. *Developer Tools Guide*. http://www.javaworld.com/javaworld/common/jw-toolstable.html, 1997.

[Orc97] Dave Orchard. *A look at Java's future*, JavaWorld, Vol. 2, Issue 6, June 1997. http://www.javaworld.com/javaworld/jw-06-1997/jw-06-javafuture.html

[Pen97] Pendragon Software Corporation. *Java Performance Report*, April 1997. http://www.webfayre.com/pendragon/jpr/jpr049702.html

[Pie96] Claudia Piemont. *Kaffeeprobe; Java-Entwicklungswerkzeuge im Vergleich.* c't magazin für computer technik, pages 266--277, December 1996.

[Pie97] Claudia Piemont. *Kaffee mit Schuß*. c't magazin für computer technik, pages 306--321, March 1997.

[Sun96] Sun Microsystems. *Java Native Interface Overview*. Part of JDK 1.1 Documentation, December 1996.

[Sun97] Sun Microsystems. *Summary of the Year in Review*. March 10 1997. http://www.javasoft.com:80/features/1996/december/yearnotes.html.

[Sur96] Vijay Sureshkumar. *Java Security*. http://csgrad.cs.vt.edu/~vijay/chap16/index.html, 1996.

Autoren:

Klaus Berg, Dipl.-Ing. (Univ)
Siemens AG, ZT SW 1, Otto-Hahn-Ring 6, D-81730 München
Tel.: +49-89-636-40504, Fax: +49-89-636-40898, e-mail: Klaus.Berg@mchp.siemens.de

Briktius Marek, Dipl.-Inform. (Univ)
Siemens AG, ZT SW 1, Otto-Hahn-Ring 6, D-81730 München
Tel.: +49-89-636-42260, Fax: +49-89-636-40898, e-mail: Brix.Marek@mchp.siemens.de

Johannes Sametinger, Dipl.-Ing. Dr.
Johannes Kepler Universität Linz, CD-Labor für Software Engineering, Altenbergerstr. 69, A-4040 Linz
Tel.: +43-70-2468-9435, Fax: +43-70-2468-9430, e-mail: sametinger@swe.uni-linz.ac.at