DIPLOMARBEIT

zur Erlangung des Grades Dipl.-Kfm.

EVALUIERUNG VON JAVA-TECHNOLOGIEN FÜR MOBILE ENDGERÄTE

Betreuer: Prof. Dr. Michael Amberg

vorgelegt an der
Rheinisch-Westfälischen Technischen Hochschule Aachen
Lehr- und Forschungsgebiet Wirtschaftsinformatik

von: Karsten Thomas
Landstraße 82a
41516 Grevenbroich
Tel. 0 21 82 / 82 54 43
e-Mail: K.Thomas@hemmerden.de
Matr.-Nr. 21 66 83

Abgabetermin: 13.9.2001

Inhalt

Abbildungsverzeichnis

Abkürzungen

3GPP	3rd Generation Partnership Project
API	Application Programming Interface
AWT	Abstract Windowing Toolkit
CBS	Cell Broadcast Service
CDC	Connected Device Configuration
CLDC	Connected Limited Device Configuration
CORBA	Common Object Request Broker Architecture
CPU	Central Processing Unit
d.h.	das heißt
DAC	Dynamic Adaptive Compilation
etc.	et cetera
evtl.	eventuell
ETSI	European Telecommunications Standards Institute
EVM	Embedded Virtual Machine
Fa.	Firma
GB	Gigabyte (1024 MB)
GHz	Gigahertz (1000 MHz)
GIF	Graphic Interchange Format
GPRS	General Packet Radio Service
GSM	Global System for Mobile Communication (Mobilfunknetz)
GUI	Graphical User Interface
HDK	Hardware Development Kit
HSCSD	High Speed Circuit Switched Data
IDE	Integrated Development Environment
J2EE	Java 2 Enterprise Edition
J2ME	Java 2 Micro Edition
J2SE	Java 2 Standard Edition
JAD	Java Application Descriptor
JAM	Java Application Manager
JAR	Java Archive Files
JCP	Java Community Process
JDK	Java Development Kit
JNI	Java Native Interface
JRE	Java Runtime Environment

JSR	Java Specification Request
JVM	Java Virtual Machine
KB	Kilobyte (1024 byte)
MB	Megabyte (1024 KB)
MIME	Multipurpose Internet Mail Extensions
MHz	Megahertz
MExE	Mobile Execution Environment
MID	Mobile Information Device
MIDP	Mobile Information Device Profile
OS	Operating System
OTA	Over-The-Air-Protocol
PC	Personal Computer
PDA	Personal Digital Assistent
PJAE	PersonalJava Application Environment
PJEE	PersonalJava Emulation Environment
PNG	Portable Network Graphics
POSE	Palm OS Emulator
RAM	Random Access Memory (“flüchtiger” Speicher)
RIM	Research In Motion, Inc.
RMI	Remote Method Invocation
RMS	Record Management System
ROM	Read Only Memory (“nichtflüchtiger” Speicher)
RTOS	Real-Time Operating System
SCSL	Sun Community Source Licensing
SDK	Software Development Kit
SMS	Short Message Service
TCP/IP	Transmission Control Protocol / Internet Protocol
UI	User Interface (Benutzerschnittstelle)
UMTS	Universal Mobile Telecommunications System
USSD	Unstructured Supplementary Service Data
usw.	und so weiter
u.U.	unter Umständen
uvm.	und viele(s) mehr
VM	Virtual Machine
W3C	World Wide Web Consortium
WAP	Wireless Application Protocol
WML	Wireless Markup Language

WTCA	Wireless Telephony Communication APIs
WWW	World Wide Web
XML	Extensible Markup Language
z.B.	zum Beispiel

Einleitung

Java ist eine von der Firma Sun Microsystems entwickelte Programmiersprache, welche sich bis heute vor allem im Bereich der Desktop-PCs etabliert hat. Eine steigende Bedeutung hat Java darüber hinaus durch das Internet erfahren: auf den Seiten des World Wide Web werden häufig Java-Applets eingesetzt. Dort bieten sie umfangreiche Möglichkeiten der Darstellung und Programmausführung, welche weit über die Fähigkeiten von HTML hinausgehen.

Nun findet Java auch in der Sub-PC-Welt Anwendung. Schon jetzt gibt es mobile Organizer (PDAs) und Mobiltelefone, welche Java unterstützen. Speziell für diese mobilen Endgeräte hat Sun mit der Java 2 Micro Edition eine Java-Version entwickelt, welche die Besonderheiten und Restriktionen mobiler Endgeräte berücksichtigt. Diese Geräte verfügen meist über wenig Speicherplatz und Rechenleistung, kleine Bildschirme und oft nur über eingeschränkte Eingabemöglichkeiten. Neben Sun gibt es zahlreiche Firmen, welche Java-Technologien für mobile Endgeräte entwickeln.
Gegenstand dieser Diplomarbeit ist die systematische Untersuchung der verschiedenen Java-Technologien, wobei die Schwerpunkte auf der Java 2 Micro Edition sowie im Bereich der „Virtual Machines“, welche den Java-Code auf dem jeweiligen Endgerät ausführen, liegen werden.

Zunächst erfolgt eine Vorstellung der aktuellen Generation mobiler Endgeräte, welche bereits Java unterstützen. Java als Programmiersprache bietet Vor- und Nachteile, deren Kenntnis für das Verständnis der Eignung dieser Programmiersprache für mobile Endgeräte unerläßlich ist.
Im den weiteren Kapiteln dieser Diplomarbeit werde ich detailliert auf die Verbindung von Java mit mobilen Endgeräten eingehen. Für die Evaluierung der Technologien sind bestehende Java-Kenntnisse hilfreich, aber nicht unbedingt Voraussetzung. Entsprechend meiner eigenen (bescheidenen) Vorkenntnisse im Bereich dieser Programmiersprache werde ich die Grundlagen ausführlich genug erläutern, um die Kapitel, in denen die technischen Details einer Virtual Machine (VM) und Java-Quellcodes behandelt werden, verständlicher zu gestalten.
Daran anschließend möchte ich mit Java verwandte Technologien vorstellen („related technologies“), die entweder auf Java basieren oder aber funktionell mit Java verwandt sind. Es folgt eine Übersicht über Entwicklungsumgebungen, die bei der Entwicklung von Java-Applikationen für den mobilen Bereich hilfreich sind.

Schließlich möchte ich die derzeitigen Java-Technologien für mobile Endgeräte zusammenfassend betrachten sowie ein Ausblick auf die kommende Gerätegeneration und deren Software geben.

Eine Schwierigkeit bei der Beobachtung dieser Technologien ist, daß sowohl die Software als auch die Hardware einer sehr schnellen Weiterentwicklung unterliegen. Zukünftige Gerätegenerationen werden sicherlich nicht mehr die Restriktionen der aktuellen Geräte aufweisen, so daß in Zukunft vielleicht etwas weniger Rücksicht auf die Besonderheiten mobiler Endgeräte genommen werden muß. Unter anderem durch die Tatsache, daß es für die Entwicklung von Software-Technologien so gut wie keine koordinierende Stelle gibt, sondern an der Entwicklung viele tausend einzelne Entwickler beteiligt sind, ist auch der Softwarebereich einem schnellen Wandel unterworfen.

Dies war vor allem während der Erstellung dieser Diplomarbeit zu spüren: es vergeht kaum eine Woche, in der nicht neue Technologien, neue Entwicklungswerkzeuge, neue Endgeräte oder geänderte Spezifikationen vorgestellt werden. Daher gibt diese Arbeit einen Überblick über die aktuelle Situation in diesem Bereich und versucht, eine Evaluierung vorzunehmen, welche Technologien derzeit sinnvoll erscheinen bzw. für die Zukunft Erfolg versprechen.
Hierbei ist Literatur, die in gedruckter Form vorliegt, meist nur für den Einstieg empfehlenswert. Wichtig sind in erster Linie die Seiten verschiedener Hersteller im World Wide Web, welche in der Literaturliste aufgeführt sind. Nur dort sind aktuelle Informationen über die neuesten Technologien verfügbar - in gedruckter Form sind sie meist bereits schon bei ihrer Veröffentlichung überholt, sofern sie überhaupt noch in Papierform herausgegeben werden.

1. Mobile Endgeräte

1.1. Verschiedene Kategorien

Was sind eigentlich mobile Endgeräte ? Meist sind hiermit die sogenannten Handheld-Organizer (PDA, z.B. Palm Pilot, Psion Revo, Compaq IPAQ etc.), Pager und Mobiltelefone gemeint. Im englischen Sprachgebrauch gibt es die „Small Computing Devices“, wobei dieser Begriff weiter gefaßt ist.[1] Er umfaßt Computer, welche - verglichen mit einem Desktop-PC - über eine eingeschränkte Prozessorgeschwindigkeit und/oder eingeschränkten Speicher verfügen. Diese Geräte müssen nicht notwendigerweise über eine Benutzerschnittstelle zur Ein- oder Ausgabe, wie z.B. einen Bildschirm, verfügen. Es kann sich auch um sogenannte Embedded Systems handeln, welche ohne die Notwendigkeit eines direkten Benutzereingriffes in andere Geräte eingebaut („eingebettet“) sind. Hier sind beispielsweise Automaten zu nennen, aber auch Steuergeräte im Automobilsektor, Set-Top-Boxen für Fernsehgeräte uvm.
Kurz gesagt handelt es sich regelmäßig dann um ein „Small Computing Device“, wenn man bei der Programmierung auf die limitierten Ressourcen wie Rechenleistung (Geschwindigkeit) und Speicher besondere Rücksicht nehmen muß.
Nicht alle dieser „Small Computing Devices“ sind gleichzeitig auch mobile Endgeräte, denn vielfach erfolgt der Einsatz stationär. Somit bilden die mobilen Endgeräte eine Untergruppe. Die Grenzen dieser Gerätegruppen sind fließend, seit es Notebooks und Kleincomputer in allen möglichen Zwischengrößen gibt. So gibt es auf der einen Seite mobile Geräte, bei denen die genannten Restriktionen nicht zutreffen (z.B. Laptops), diese sind jedoch im Zusammenhang mit den in dieser Arbeit berücksichtigten Produkten nicht gemeint.
Auf der anderen Seite lassen sich jedoch viele in dieser Arbeit beschriebenen Technologien mit den stationär eingesetzten Geräten der Embedded Systems nutzen, obwohl diese nicht direkt zu den „mobilen Endgeräten“ zählen.
Mehr hierzu in Kapitel 3.3., in welchem die Java-Konfigurationen CDC (Connected Device Configuration) und CLDC (Connected Limited Device Configuration) behandelt werden.

[1] Vgl. Giguère, Eric, „Java 2 Micro Edition“, Professional Developer´s Guide Series, John Wiley & Sons, Inc., New York, Toronto, Weinheim, 1999, S. 4

Konkret wird Java derzeit von den folgenden Geräten unterstützt:

- alle Geräte mit Palm OS ab Version 3.01
- alle Geräte mit Windows CE (z.B. Compaq iPAQ)
- EPOC-Geräte (z.B. Psion Serie 5)
- Siemens-Mobiltelefone (z.B. SL45i, ab Herbst 2001)
- Nokia-Geräte (z.B. Communicator 9210)
- Motorola-Geräte (z.B. i3000, Accompli 008)
- RIM/Compaq Blackberry (Pager, Verbreitung in USA)

Diese Aufzählung erhebt keinen Anspruch auf Vollständigkeit. Es wird jedoch deutlich, daß die größten Hersteller alle Java unterstützen und kaum ein weit verbreitetes Gerät existiert, für das nicht zumindest ein Nachfolger mit Java-Unterstützung geplant ist.[2]

Aufgrund der Vielzahl unterschiedlichster Geräte (Palm, Windows CE, Mobiltelefone) wird der Hauptgrund für die Verwendung einer Programmiersprache wie Java deutlich. Die Betriebssysteme der Geräte sind untereinander oft nicht kompatibel, so daß die Notwendigkeit besteht, Programme für jeden einzelnen Gerätetyp zu modifizieren und neu zu compilieren. Dies ist bei Java-Programmen nicht notwendig, da sie im jeweiligen Endgerät durch eine Virtuelle Maschine ausgeführt werden. Hier muß nur die Virtuelle Maschine einmalig an das Endgerät angepaßt und installiert werden, die Java-Programme sind anschließend - in Grenzen - ohne weitere Änderungen auf allen Plattformen lauffähig.
Dieser Grundgedanke ist in der Regel allerdings nicht vollständig zu realisieren, da bezüglich der Hardware meist zu große Unterschiede vorliegen (Displaygröße, Tastatur etc.).

[2] Eine umfangreiche Übersicht Java-fähiger Endgeräte findet sich bei JavaMobiles, http://www.javamobiles.com/

1.2. Die technische Ausstattung

Wie im vorigen Kapitel erläutert, verfügen mobile Endgeräte über eine eingeschränkte Kapazität an Speicher und Rechenleistung sowie über kleine Bildschirme und Tastaturen (sofern überhaupt vorhanden). Weshalb diese Restriktionen so bedeutsam sind, wird im folgenden deutlich.[3]

1.2.1. Speicherkapazität

Moderne Desktop-PCs besitzen heutzutage eine Speicherkapazität von 128, 256 MB oder mehr Hauptspeicher (RAM). Im Vergleich mit diesen sind mobile Endgeräte mit 4 oder 8 MB schon auf den ersten Blick benachteiligt. Doch es kommen Geräte auf den Markt, welche über bis zu 64 MB verfügen - und damit nicht mehr weit entfernt von den Kapazitäten im Desktop-Bereich liegen. Diese erlauben auch die Speicherung und Ausführung größerer Programme. Ein wesentlicher Unterschied besteht jedoch, wenn man die Gesamt-Speicherkapazität der Geräte betrachtet. Bei Desktop-PCs besteht die Möglichkeit, in großem Umfang Daten auf andere Speichermedien auszulagern, beispielsweise auf Festplatten mit einer Kapazität von 60 GB oder mehr. Es existieren zwar Lösungen für Organizer in Form von externen Speicherkarten, doch diese bieten nicht die große Kapazität von Desktop-Festplatten. Hinzu kommt, daß solche Zusatzkarten meist durch ihre Bauform und den erhöhten Stromverbrauch die Mobilität des Organizers einschränken.
So liegt die Gesamt-Speicherkapazität mobiler Endgeräte weit unter der von Desktop-Geräten.

1.2.2. Prozessorleistung

Mobile Endgeräte sind batteriebetrieben. Um eine möglichst lange Funktionsdauer zu erreichen, ohne die Akkus aufladen zu müssen, ist ein stromsparendes Design erforderlich. Notebook-Computer funktionieren meist nur wenige Stunden ohne eine Verbindung zum Stromnetz, von modernen Mobiltelefonen und PDAs wird dagegen eine Funktionsdauer von zumindest mehreren Tagen ohne Ladevorgang erwartet.

[3] Vgl. Giguère, Eric, „Java 2 Micro Edition“, S. 5 ff.

Neben dem Bildschirm, welcher in der Regel den meisten Strom benötigt, ist die Geschwindigkeit des verwendeten Prozessors in hohem Maße für den Stromverbrauch verantwortlich. So finden in mobilen Endgeräten speziell auf diese Funktion optimierte Prozessoren sowie Prozessoren mit geringer Geschwindigkeit Verwendung. Sie arbeiten meist mit etwa 20 MHz, es gibt wenige Geräte mit bis zu 200 MHz. Im Vergleich zu einem Prozessor für Desktop-Rechner mit zum Teil über 1 GHz (1000 MHz) ist dies sehr gering; die Programmausführung erfolgt wesentlich langsamer.

1.2.3. Ein- und Ausgabemethoden

Für die Verwendung von Java-Technologien weniger bedeutsam als Speicherausstattung und Prozessor sind die Benutzerschnittstellen zur Ein- und Ausgabe. Allerdings gibt es einige Besonderheiten im Vergleich zu Desktop-Systemen.

Zur Dateneingabe steht bei mobilen Endgeräten regelmäßig keine Standard-Tastatur zur Verfügung (wobei diese für manche Geräte als optionales Zubehör erhältlich ist). Aufgrund des platzsparenden Designs, um kompakte Außenmaße zu erreichen, sind die numerischen Tasten des Mobiltelefons meist mehrfach belegt und dienen durch mehrfachen Tastendruck auch der Eingabe von Buchstaben. Es gibt Technologien wie z.B. „T9“, um die Dateneingabe zu vereinfachen, indem aufgrund eines integrierten Wörterbuches versucht wird, das einzugebende Wort nach jedem Tastendruck zu „erraten“. Palm Inc. verwendet bei den Palm-Organizern eine eigene Schrift namens „Graffiti“, welche bei der Eingabe mit dem Zeichenstift zu einer höheren Erkennungsrate führt als Handschriften oder normale Druckbuchstaben. Es sind meist (wenige) Tasten oder Roll-Räder („Wheels“) vorhanden, um sich beispielsweise in Menüs auf und ab zu bewegen oder eine Funktionsauswahl innerhalb eines Programms zu treffen.
Problematisch für Java-Programme ist an dieser Stelle, daß diese Tasten nicht einheitlich vorhanden und von Endgerät zu Endgerät verschieden angeordnet sind. Hier ist eine gerätespezifische Schnittstelle erforderlich, welche die Ansteuerung dieser Funktionen übernimmt, damit die Bedienung verschiedener Java-Programme für den Benutzer einheitlich erfolgen kann. Auch Funktionen wie T9 oder Graffiti müssen vom Endgerät zur Verfügung gestellt werden, denn diese müssen unabhängig vom verwendeten (Java-)Programm funktionieren.

Ist diese Technologie bei Mobiltelefonen bisher fast nicht vorhanden, so finden bei PDAs in der Regel Touch-Screens Verwendung, druckempfindliche Bildschirme, welche neben der Datenanzeige gleichzeitig auch der Dateneingabe dienen. So läßt sich ein Form-Faktor realisieren, welcher auf engstem Raum viele Funktionen zuläßt.
Aufgrund des Form-Faktors und der hohen Bedeutung des Stromverbrauches sind viele Bildschirme sehr klein gehalten und bieten meist nur eine Schwarz-/Weiß-Darstellung. Auch dies ist bei der Entwicklung von Anwendungen für mobile Endgeräte zu berücksichtigen.

Zur Datenkommunikation steht bei mobilen Endgeräten oft eine Netzwerkverbindung, meist zum Internet, zur Verfügung. Um die Ansteuerung dieser Funktionen zu vereinheitlichen, hat Sun innerhalb der Java 2 Micro Edition (J2ME) gerätegruppenspezifische Konfigurationen entwickelt, CLDC und CDC (vgl. Kapitel 3.3.). Eine Verbindung zum Internet muß demnach zwar nicht permanent aufgebaut, aber jederzeit verfügbar sein, so daß sie bei Bedarf aufgebaut werden kann. Eine standardisierte Java-Konfiguration für Geräte ohne Netzanbindung steht bisher nicht zur Verfügung.

1.3. Die heutige Bedeutung mobiler Endgeräte

Waren noch vor einiger Zeit Laptops die einzigen Geräte, die eine mobile Nutzung zuließen, gibt es mittlerweile eine Vielzahl dieser Geräte, welche bei immer geringeren Außenmaßen eine immer höhere Leistung besitzen (PDAs, Personal Desktop Assistent). Mit steigender Ausstattung an Rechenleistung und Speicherkapazität ermöglichen diese Geräte heute die Ausführung komplexer Programme. Die jüngste Entwicklung verbindet die Vorteile von Mobiltelefonen mit denen der PDAs. In diesem Jahr kommen die ersten Mobiltelefone auf den Markt, welche neben der integrierten Organizer-Funktionalität auch die Ausführung von Java-Programmen ermöglichen.
Der Hauptvorteil der Verbindung von PDA und Telefon ist sicherlich darin zu sehen, daß diesen Programmen eine vollständige Netzanbindung über die Mobilfunknetze zur Verfügung steht, beispielsweise an das Internet oder firmeninterne Intranets. So ist ein Zugriff auf entfernte Daten möglich, aber auch das Herunterladen von neuen Programmen in das Mobiltelefon.

Durch diese Funktionalität haben die mobilen Endgeräte in letzter Zeit enorm an Bedeutung hinzugewonnen. Vor einiger Zeit dienten sie als Datenbank, mit der man maximal einige hundert Telefonnummern bei sich tragen konnte, ein Abgleich dieser Daten mit dem Desktop-PC war meist nicht möglich. Nur wenige Geräte waren per seriellem Kabel an den PC anschließbar. Meist begrenzte der limitierte Speicher von wenigen Kilobyte die Nutzung für weitergehende Applikationen, selbst wenn das Gerät an sich programmierbar war (z.B. die erste Generation der Psion-Organizer).
Heute verfügen PDAs meist über RAM-Speicher zwischen 2 und 64 MB, was den Weg für eine umfangreichere Nutzung eröffnet.
Pager wie z.B. Cityruf, Scall u.a. sind heute dank Technologien wie SMS und WAP sowie neuen Abrechnungskonzepten wie Prepaid-Karten fast vollständig durch Mobiltelefone ersetzt und spielen nur noch eine untergeordnete Rolle. Selbst in Automaten (Embedded Systems) kommen GSM-Module zum Einsatz, welche für die Datenübertragung sorgen. Fast die einzige Ausnahme bilden die Blackberry-Geräte[4] von Research In Motion (RIM) / Compaq, welche in den USA weit verbreitet sind und nicht nur Nachrichten empfangen, sondern auch versenden können.
Mobiltelefone sind neben reinen PDAs die weitere große Gruppe der mobilen Endgeräte. Verfügten sie bis vor kurzem nur über rudimentäre Organizer-Funktionen wie z.B. einen Kalender und ein Telefonbuch, so bieten sie heute teilweise schon dieselben Funktionen wie PDAs. Zukünftig wird der Unterschied mehr und mehr verschwimmen, da alle Funktionen wie Telefonie, Internet-Zugang und die klassischen Organizer-Funktionen in einem einzigen Gerät integriert sein werden.

Aufgrund dieser Vorteile, den immer kleineren Abmessungen und den mit höheren Stückzahlen sinkenden Preisen werden diese mobilen Endgeräte für eine immer größere Zielgruppe attraktiv. Mittlerweile sind sie durch fast intuitiv zu bedienende Benutzerschnittstellen (User Interfaces, UI) auch professionellen Anforderungen gewachsen. Aufgrund des sinkenden Preisniveaus werden Mobiltelefone zunehmend mit den Funktionen eines PDAs ausgestattet, so daß die Grenze zwischen diesen Gerätegruppen verschwimmt und auch Privatnutzer über diese Geräte verfügen werden.
An dieser Stelle ist eine hohe Durchdringung des Marktes mit mobilen Endgeräten erreicht, so daß diese aus dem täglichen Leben nicht mehr wegzudenken sind.

Allen Geräten gemeinsam ist, daß sie Software benötigen. Ob Java dafür die geeignete Plattform ist, wird in den folgenden Kapiteln erörtert.

[4] Compaq, iPAQ Blackberry, http://www.compaq.com/products/handhelds/blackberry/index.html

2. Java als Programmiersprache für mobile Endgeräte

2.1. Die Rolle von Sun Microsystems

Die Entwicklung von Java als Programmiersprache begann 1990, als die Firma Sun Microsystems eine objektorientierte Programmiersprache für ein Projekt zur Softwaresteuerung von Kommunikationsgeräten benötigte. Ursprünglich war geplant, C++ einzusetzen. Da diese Sprache sich jedoch als nicht vollständig geeignet für diese Aufgabenstellung erwies, entwickelte James Gosling zu diesem Zweck Java.[5] Somit wurde Java von vornherein nicht für High-End-PCs, sondern für Embedded-Geräte entwickelt. Bedingung war, daß Java „(...) klein, effizient und leicht auf eine ganze Reihe von Geräten portierbar sein[6]“ mußte. Im Laufe der folgenden Jahre vergab Sun eine Reihe von Lizenzen an andere Firmen, darunter Netscape und Microsoft, welche Java-Technologien in ihren Produkten einsetzen.

Heute erfolgt die Weiterentwicklung in Form des „Java Community Process“ (JCP)[7], einem offenen Zusammenschluß von Sun Microsystems und über 500 Entwicklern, welcher von Sun Microsystems moderiert wird.

Zugleich tritt Sun als profitorientierter Lösungsanbieter für Java-Technologien und -Produkte auf - eine Funktion, welche sich nicht immer mit der Rolle als „Gralshüter Javas[8]“ verträgt.

Einerseits ist Sun als Moderator des Java Community Process mit verantwortlich für die Definition von Java-Standards, auf der anderen Seite steht das Unternehmen unter dem Druck, profitabel kommerzielle Produkte zu entwickeln und zu vermarkten. Es dürfte nicht immer leicht fallen, diese Bereiche klar voneinander zu trennen und dabei andere Konkurrenten am Markt nicht zu benachteiligen.

[5] Vgl. Lemay, Laura, Cadenhead, Roger, „JAVA 2”, Markt + Technik Verlag, München, 2001, S. 32

[6] ebenda, S. 32

[7] The Java Community Process (JCP), http://jcp.org/

[8] Java Magazin, Software & Support Verlag, Frankfurt, Heft 5/2001, S. 3

2.2. Vor- und Nachteile von Java

Java wird von vielen Programmierern belächelt, weil es im Vergleich mit anderen Programmiersprachen (wie zum Beispiel C++) über einige Nachteile verfügt. Die Ausführungsgeschwindigkeit liegt unter dem Durchschnitt, was allerdings stark von der verwendeten Virtual Machine abhängt. Einige Dinge sind mit Java nur sehr umständlich umzusetzen, was in anderen Programmiersprachen direkter und einfacher möglich ist. Manches war bisher gar nicht oder nur rudimentär implementiert.
Allerdings hat sich durch die neue Version von Java 2 einiges verbessert.[9] Dank der HotSpot-Technologie (vgl. Kapitel 4 über Virtual Machines) laufen Programme schneller und verbrauchen weniger Speicher. Auch darüber hinaus wurde die Performance stark verbessert. Mit JavaSound werden nun auch die Soundwiedergabe und -Aufnahme weit besser unterstützt.
Manche der vermeintlichen Nachteile sind außerdem nur auf den ersten Blick nachteilig und stellen sich bei näherer Betrachtung als Vorteil dar.
Java ist objektorientiert, leicht zu erlernen und plattformübergreifend. Damit sind sicherlich einige Restriktionen verbunden, dennoch bietet Java so eine Flexibilität, welche bei anderen Programmiersprachen in dieser Form nicht zu finden ist. Speziell die Tatsache, daß Java plattformübergreifend ist, spielt im Bereich der mobilen Endgeräte eine tragende Rolle.

- Java ist objektorientiert

Dies erlaubt es, Programmteile zu erstellen, welche mühelos in anderen Programmen wiederverwendet werden können. Aus Sicht der Programmierer ist dies ein Vorteil, weil auf bestehende Routinen zurückgegriffen werden kann, ohne diese neu programmieren zu müssen. Auch - und gerade - im mobilen Bereich, wo Speicherplatz und Übertragungskapazität einen Engpaß darstellen, ist die Objektorientierung ein nicht zu unterschätzender Vorteil.

- Java ist leicht zu erlernen

Java wurde zwar als Alternative zu C++ entwickelt, ist aber stark an diese Sprache angelehnt und übernimmt vieles von deren Syntax.[10] Dies ermöglicht einem großen Kreis von Programmierern die schnelle Einarbeitung in die Programmiersprache. Im

[9] Vgl. Lemay, Laura, Cadenhead, Roger, „JAVA 2“, S. 34 ff.
[10] ebenda, S. 36

Gegensatz zu C++ verzichtet Java auf Zeiger, Zeigerarithmetik und andere fehlerträchtige Aspekte. Es gibt keine Mehrfachvererbung, und das Speichermanagement mit Garbage Collection erfolgt automatisch. Somit gibt es sicherlich einige Dinge, die erfahrene Programmierer vermissen, anderen aber wird der Einstieg in die Programmiersprache hierdurch stark erleichtert.

- Java ist plattformunabhängig

Ein Grundgedanke bei der Entwicklung von Java war es, daß die Programme auf jeder Plattform laufen sollten, unabhängig davon, wofür sie entwickelt wurden. Dies war im Gegensatz zu anderen bisherigen Programmiersprachen eine wesentliche Neuheit.
Wenn das Ziel auch nicht vollständig erreicht wurde, so ist es dank Java doch wesentlich einfacher geworden, ein Programm auf eine andere Plattform zu portieren. Besonders im Hinblick auf den Einsatz im mobilen Sektor ist dies ein wichtiger Aspekt. Unterstützt ein Gerät eine bestimmte Virtual Machine (VM), so ist bei Einsatz des entsprechenden MIDP-Profils fast sichergestellt, daß die Anwendung auch auf anderen Endgeräten mit dieser Spezifikation funktioniert.

Mit zunehmender Rechenleistung, welche die Endgeräte zur Verfügung stellen, und mit zunehmender Optimierung der Virtual Machines stellt die eher langsame Ausführungsgeschwindigkeit bereits heute kaum noch einen Nachteil dar. Hinzu kommt, daß es sich bei den mobilen Anwendungen selten um derart zeitkritische Applikationen handelt, daß dieser Nachteil stark ins Gewicht fiele. Hier gibt es andere Dinge, welche die Geschwindigkeit stärker belasten, z.B. die in heutigen GSM-Netzen noch vergleichsweise langsame Datenübertragung.
Durch die Java 2 Micro Edition (J2ME) und die Klassen, welche durch CLDC und MIDP sowie die herstellereigenen Software Development Kits (SDK) zur Verfügung stehen, lassen sich nun auch genau auf mobile Endgeräte zugeschnittene Applikationen entwickeln.
Die Vorteile von Java liegen sicherlich nicht in der Geschwindigkeit, sondern eher darin, daß die Sprache plattformübergreifend funktioniert. In den Bereichen, wo genau diese Gesichtspunkte gefragt sind, bietet Java verglichen mit anderen Programmiersprachen einige Vorteile. Genau dies ist im Bereich der mobilen Endgeräte der Fall.

2.3. Verschiedene Java-Technologien für mobile Endgeräte

Zur Zeit existieren einige Technologien, welche eine Java-Unterstützung für mobile Endgeräte bieten. Java 2 Micro Edition (J2ME) ist das Produkt, welches heute am meisten beachtet wird und in dieser Arbeit den Schwerpunkt bilden soll. Dieses wird in Kapitel 2.4. ausführlich besprochen. Es gibt jedoch noch andere weit verbreitete und ausgereifte Technologien. Die wichtigsten werden im folgenden vorgestellt.[11]

2.3.1. PersonalJava

PersonalJava existierte schon lange, bevor Sun die Java 2 Micro Edition (J2ME) vorgestellt hat. Es wurde im Jahre 1998 für den Einsatz in „Konsumgütern" konzipiert (im englischen Sprachgebrauch besser: die Gruppe der „consumer electronic devices[12]"). Ziel sind hauptsächlich Geräte mit Netzwerkfunktionen und einer grafischen Benutzeroberfläche, also PDAs, Mobiltelefone und Set-Top-Boxen für Fernsehgeräte.
PersonalJava liegt heute in der Version 1.2a vor[13] und setzt eine fast vollständige Java Virtual Machine (JVM) voraus, wobei es sich um eine reduzierte Version der JVM 1.1 handelt. Dies ist der wichtigste Unterschied gegenüber der J2ME, welche hauptsächlich auf einer speziell auf mobile Endgeräte zugeschnittenen Virtual Machine, der KVM, aufbaut. PersonalJava stellt einen Teil der Kern-Java-APIs zur Verfügung, speziell eine Teilmenge des JDK 1.1 API. Gegenüber der Java 2 Standard Edition (J2SE) weist PersonalJava eine Reihe von Optimierungen auf, die für den Einsatz auf mobilen Endgeräten erforderlich sind. So ist aufgrund der speziellen Klassen der Speicherbedarf wesentlich geringer als bei Einsatz der J2SE, außerdem gibt es Klassen für die grafische Darstellung auf kleinen Bildschirmen.

[11] Vgl. Ericsson, „Mobile Applications with J2ME, A White Paper", http://www.ericsson.de/upload/download/white_paper_j2me.pdf (Stand: 19.08.2001)

[12] Moreira, Paulo, Micro Java Network, "From PersonalJava to J2ME: Some Introductory Ideas", http://www.kvmworld.com/news/perspective/personaljava_j2me?content_id=1440 (Stand: 13.07.2001)

[13] Vgl. Sun Microsystems, "PersonalJava Specification 1.2", http://java.sun.com/products/personaljava (Stand: 20.08.2001)

Das API von PersonalJava ist unterteilt in obligatorische und optionale Pakete:

- In den obligatorischen Paketen sind Klassen aus `java.applet`, `java.awt` und `java.net` enthalten. Dies hat zur Folge, daß einerseits jede PersonalJava-Implementierung diese Pakete umfassen muß, andererseits aber auch jedes Endgerät, welches PersonalJava ausführen soll, über die entsprechenden Funktionen verfügen muß, also in jedem Fall eine Netzwerkanbindung und eine grafische Benutzerschnittstelle.
  Die grafischen Funktionen von Swing werden nicht unterstützt, dies ist aufgrund der eingeschränkten Ressourcen allerdings verständlich.
- In den optionalen Paketen werden weitere Funktionen bereitgestellt, welche über die Standard-Funktionen hinausgehen, also beispielsweise `java.rmi` oder die Datenbankanbindung über `java.sql`.
  Welche der optionalen Pakete schließlich eingebunden werden können, hängt davon ab, ob sie vom Hersteller unterstützt werden. Hier ergibt sich das Problem, daß der Entwickler einer PersonalJava-Anwendung bei der Verwendung optionaler Klassen nicht davon ausgehen kann, daß diese auch in jedem Fall vom verwendeten Endgerät unterstützt werden.

Von Sun Microsystems ist ein Emulator namens „PersonalJava Emulation Environment“ (PJEE) erhältlich,[14] welcher die Ausführung von PersonalJava-Programmen auf Windows- und Solaris-Plattformen erlaubt.
Des weiteren ist mit JavaCheck ein Werkzeug erhältlich,[15] mit dem ein Quellcode auf seine Lauffähigkeit in bezug auf PersonalJava überprüft werden kann (d.h. ob die verwendeten Klassen auch dort enthalten sind, oder ob „unerlaubte“ nicht in PersonalJava implementierte APIs aufgerufen werden).

Im Vergleich zu PersonalJava unterstützt J2ME eine größere Anzahl von Endgeräten, so daß PersonalJava eine Teilmenge der J2ME ist. Diese Tatsache führt dazu, daß PersonalJava in Zukunft kein eigenständiges Produkt mehr sein wird, sondern neu definiert und als Profil in die J2ME integriert wird. Dieses Profil wird kompatibel zu bisherigen Versionen von PersonalJava sein (mehr dazu in Kapitel 3.4.5.).

[14] PersonalJava Emulation Environment,
http://java.sun.com/products/personaljava/pj-emulation.html (Stand: 21.08.2001)

[15] JavaCheck, http://java.sun.com/products/personaljava/javacheck.html (Stand: 21.08.2001)

2.3.2. EmbeddedJava

EmbeddedJava[16] ist PersonalJava sehr ähnlich, ist jedoch speziell auf "Embedded-Geräte" zugeschnitten. Sun Microsystems sieht die Zielgruppe sehr breit gestreut: "Today the market for embedded devices spans a wide variety of consumer and business products, including devices such as mobile phones, pagers, PDAs, set-top boxes, process controllers, office printers, and network routers and switches.[17]" Auch wenn EmbeddedJava laut Sun ausdrücklich für PDAs und Mobiltelefone geeignet ist, wird es dort vermutlich kaum zum Einsatz kommen. Hier bieten sich andere Lösungen wie z.B. die J2ME mit entsprechenden Konfigurationen und Profilen an. Drucker, Router, Switches und (Verkaufs-)Automaten jedoch sind Geräte, welche genau eine spezielle Funktion besitzen, für die sie geschaffen wurden. Hier erscheint der Einsatz von EmbeddedJava sinnvoll.
Dies liegt hauptsächlich darin begründet, daß in EmbeddedJava - im Gegensatz zu PersonalJava - sämtliche Klassen und Methoden optional sind und keine Klassen obligatorisch vorhanden sein müssen. Dies erschwert es Fremdherstellern, Software für solche Geräte zu entwickeln. Solange der Hersteller der Geräte nicht öffentlich dokumentiert, welche Funktionen tatsächlich im Endgerät zur Verfügung stehen, ist eine Programmierung nicht möglich.[18] In der Praxis werden Geräte mit EmbeddedJava daher nur von den Herstellern selbst programmiert werden. Dies stellt meist kein Problem dar, da viele der in Frage kommenden Geräte ohnehin nur eine Funktion bieten (Automaten, Router, Switches) und eine Fremdsoftware in diesen Geräten nicht sinnvoll erscheint. In dem Fall, daß Fremdsoftware nützlich sein könnte (wie beispielsweise auf PDAs), wird vermutlich von vornherein PersonalJava bzw. J2ME zum Einsatz kommen.

[16] EmbeddedJava, http://java.sun.com/products/embeddedjava/ (Stand: 18.08.2001)

[17] Sun Microsystems, "Technical Overview of EmbeddedJava Technology", http://java.sun.com/products/embeddedjava/overview.html (Stand: 18.08.2001)

[18] Vgl. Ericsson, „Mobile Applications with J2ME, A White Paper", S. 14

2.3.3. Java Card

Für den Einsatz auf Smart Cards stellt Sun die Java Card Technologie bereit.[19] Smart Cards sind Karten in Scheckkartengröße, auf denen elektronische Schaltkreise integriert sind. Ein bekanntes Beispiel sind SIM-Karten für Mobiltelefone. Diese verfügen über einen Chip, auf welchem nicht nur Daten gespeichert, sondern vollständige Programme ausgeführt werden können. Die Speicherkapazität beträgt heute bis zu 32 KB.

Die Schnittstelle zur Kommunikation nach außen (beispielsweise zum Mobiltelefon) ist genormt, während die auf dem Chip ablaufende Software vom Hersteller frei gewählt werden kann. Java Card ist eine solche Software. Sie stellt eine sehr kleine Virtual Machine zur Verfügung, welche auf den Karten fast aller Hersteller lauffähig ist.
Einsatz findet Java Card auch auf den oben genannten SIM-Karten für Mobiltelefone. Hier ist meist ein so genanntes SIM-Toolkit installiert, durch welches Zusatzfunktionen für das Mobilfunknetz eines bestimmten Netzbetreibers zur Verfügung gestellt werden. Durch diese standardisierte Funktion ist es möglich, diese Funktionen nahtlos in das bestehende Menü des Mobiltelefons einzubinden - die Software dazu läuft auf der SIM-Card ab.
Ein wichtiger Vorteil dieser Technologie ist die flexible Programmierbarkeit. Selbst mittels eines Over-The-Air-Protokolls (OTA) läßt sich die SIM-Karte neu programmieren, indem z.B. per SMS neue Software oder neue Daten an die Karte übermittelt werden.[20]
VIAG Interkom beispielsweise setzt diese Technologie zur Steuerung der Informationen über die „Home Zone“[21] ein. Per SMS werden Informationen über die genaue Lage des Mittelpunktes der Home Zone an die SIM-Karte übertragen. Auf diese Weise wird dem Mobiltelefon dynamisch mitgeteilt, in welchem Bereich (an welchen Sendemasten) preiswerte Telefongespräche geführt werden können.

---

[19] Sun Microsystems, Java Card Technology, http://java.sun.com/products/javacard/ (Stand: 19.08.2001)

[20] Vgl. Ericsson, „Mobile Applications with J2ME, A White Paper“, S. 15

[21] Die „Home Zone“ ist ein geographischer Bereich, in dem im Rahmen eines GENION-Mobilfunkvertrages von VIAG Interkom Telefongespräche zu ermäßigten Tarifen geführt werden können.

Da Java Card keine Unterstützung für Grafikausgabe, Benutzereingriffe, Multi-Threading oder Speichermanagement bietet und nur sehr wenige Java-Klassen unterstützt, ist der Einsatz dieser Technologie auf den Bereich der Smart Cards beschränkt. Für andere Endgeräte bieten sich andere hier genannte Technologien an.

2.3.4. MExE

MExE steht für „Mobile Execution Environment“. Hierbei handelt es sich um eine Spezifikation, welche vom European Telecommunications Standards Institute (ETSI) aufgestellt wurde und nun im „3rd Generation Partnership Project“ (3GPP) weiterentwickelt wird (deren Mitglied das ETSI ist).
MExE definiert einen Standard für eine Ausführungsumgebung für mobile Endgeräte[22] und umfaßt drei „Classmarks“:

- MExE Classmark 1: WAP
  Geräte, welche diese Spezifikation unterstützen, basieren auf dem WAP-Standard
- MExE Classmark 2: PersonalJava
  Classmark 2 umfaßt PersonalJava, welches auf dem Endgerät verfügbar sein muß, sowie als Erweiterung das JavaPhone API. Hierbei handelt es sich um ein API, das im Wesentlichen Funktionen zur Ansteuerung der Telefoniefunktionen und zum Nachrichtenversand bereitstellt.
- MExE Classmark 3: Java 2 Micro Edition CLDC
  Geräte dieser Spezifikation unterstützen J2ME in Verbindung mit der CLDC-Konfiguration und dem MID-Profil (MIDP), welche in Kapitel 3 beschrieben werden.

Somit stellt MExE im Grunde keine eigene neue Technologie dar, sondern definiert Anforderungen für Endgeräte, die auf bewährten Standards basieren.
MExE-Geräte müssen mindestens ein „Classmark“ unterstützen, optional auch mehrere. In der Regel werden Classmark 1 und zusätzlich entweder 2 oder 3 unterstützt, so daß in jedem Fall das WAP-Protokoll implementiert ist.
Microsoft bemüht sich derzeit, ein Classmark 4 einzuführen, welches Microsoft .NET-Technologien beinhalten soll.

[22] Vgl. Ericsson, „Mobile Applications with J2ME, A White Paper“, S. 15

2.3.5. Microsoft .NET

Bei Microsoft .NET handelt es sich um eine Plattform der Firma Microsoft für XML[23]-basierte Dienste. Microsoft plant, in sämtlichen Produkten der kommenden Generation (einschließlich Betriebssystemen und Servern) .NET und damit die Verteilung von Software mittels XML zu unterstützen. Derzeit ist die Haupt-Entwicklungsumgebung Visual Studio .NET, welche die Sprachen Visual Basic, C++ und C# unterstützt, darüber hinaus aber auch für die Entwicklung von Java-Programmen geeignet ist.
In welche Richtung sich Microsoft .NET entwickeln wird, ist derzeit noch unklar. Es handelt sich um mehr als nur eine Technologie - vielmehr beschreibt es eine zukünftige Strategie der Firma Microsoft.

2.3.6. BREW

BREW ist ein Akronym für „Binary Runtime Environment for Wireless“, eine von der Firma Qualcomm[24] entwickelte Technologie, welche als Middleware zwischen der Hardware eines Mobiltelefons und der darauf ablaufenden Software agiert. BREW stellt der Anwendung die Telefonie-Funktionen standardisiert zur Verfügung, so daß Anwendungsentwickler keine Kenntnisse über die Details der verwendeten Hardware (Chipsatz etc.) besitzen müssen. Die BREW-Programme heißen Applets, sind aber nicht in Java entwickelt (wie man vom Namen her erwarten könnte), sondern in C++ oder in C. BREW stellt APIs zur Verfügung, welche den Applikationen beispielsweise Zugriff auf die Benutzerschnittstelle (einschließlich des Bildschirms), die Netzwerkverbindung oder auch das Speichermanagement des Gerätes gewähren. BREW-Applets lassen sich per OTA (Over The Air-Protokoll) an das Endgerät übermitteln.
Insgesamt scheint BREW eine ernstzunehmende Alternative zu Java zu sein, welche große Ähnlichkeit zu den derzeitigen Java-Technologien bietet. Allerdings ist nicht zu übersehen, daß BREW bisher nur von sehr wenigen Geräteherstellern unterstützt wird (u.a. NTTDoCoMo) und - trotz eines guten Konzeptes - zur Zeit nur einen Alleingang der Firma Qualcomm darstellt. Betrachtet man die große Gemeinde der Java-Entwickler mit allen damit zusammenhängenden Firmen vom Softwareentwickler bis zum Hardwarehersteller, so erscheint es sehr zweifelhaft, ob BREW tatsächlich Fuß

[23] E**x**tensible **M**arkup **L**anguage, ein Standard für Datenaustausch, entwickelt vom World Wide Web Consortium (W3C), http://www.w3c.org/

[24] Qualcomm Inc., http://www.qualcomm.com/

fassen und eine Alternative zu den weit verbreiteten etablierten Java-Technologien werden kann. Dies wird vermutlich nur gelingen, wenn eine ausreichende Zahl einflußreicher Partner diese Technologie einsetzt und mehrere Hersteller BREW in ihren Geräten implementieren.

2.3.7. i-Mode (NTT DoCoMo)

i-Mode ist ein Service des japanischen Providers NTT DoCoMo, der den Benutzern seit 1999 mobilen Zugang zum Internet verschafft. Seit Anfang 2001 wird ergänzend der Service „iAppli“ angeboten, welcher das Herunterladen von Java-Applikationen auf Java-fähige Endgeräte unterstützt.[25] iAppli basiert auf der CLDC-Konfiguration für mobile Java-Endgeräte mit Netzanbindung, entspricht dabei allerdings nicht dem vom Java Community Process entwickelten Profil MIDP (vgl. Kapitel 3.4.1.).
Im Vergleich zu MIDP verfügt iAppli über die folgenden Erweiterungen:[26]

- komplexere Benutzerschnittstelle
- GIF-Unterstützung (MIDP unterstützt lediglich PNG)
- zusätzliche Multimedia-Funktionen (Sound und Animation)
- sichere Verbindungen via HTTPS (!)
- japanische Sprachunterstützung (S-JIS)
- spezieller Zeichensatz für DoCoMo-Anwendungen

Mit Hilfe dieser Erweiterungen war auch ohne die Verfügbarkeit eines MID-Profils der Einsatz von Java auf mobilen Endgeräten realisierbar. Zum ersten Mal war es möglich, Anwendungen wie beispielsweise Spiele mit Netzwerkfunktionen (Server mit Highscore-Listen, Nachladen neuer Levels über das Internet) oder interaktive dynamische Stadtpläne anzubieten.
Diese Technologie verschaffte NTT DoCoMo bzw. i-Mode einen großen Vorsprung: bis heute hat es kein Mobilfunkanbieter geschafft, in einem solchen kurzen Zeitraum einen so großen Kundenkreis zu erschließen, wie es NTT DoCoMo mit diesen Technologien gelang.
An dieser Stelle wurde erstmals deutlich, welches Potential in der Verwendung von Java im Bereich mobiler Endgeräte liegt.

---

[25] Vgl. wap.de, “NTT DoCoMo mit mobilen Internet-Service auf Java-Basis“, http://www.wap.de/News/Archiv/2001/01/N010131ntt,version=3.html (Stand: 13.07.2001)

[26] Vgl. Ericsson, „Mobile Applications with J2ME, A White Paper“, S. 12

3. Java 2 Micro Edition

3.1. Terminologie

Die Namensgebung der Java-Komponenten seitens Sun Microsystems war nicht immer einheitlich und leicht durchschaubar. Zum besseren Verständnis möchte ich dies hier kurz erläutern.
Mit SDK ist das Java 2 Software Development Kit gemeint, welches früher unter dem Namen JDK (Java Development Kit) bekannt war. Allerdings war damals nicht klar, ob sich die Bezeichnung „Java 1.x“ auf die Version der Programmiersprache oder auf das JDK bezog, oder ob diese Angabe die Version der Virtual Machine (VM) bezeichnete. Das SDK hätte eigentlich die Version 1.2 haben müssen, aus Marketinggründen wurde jedoch für die neue Version mit umfangreichen Änderungen die Bezeichnung „Java 2 Software Development Kit“ gewählt. „Java 2“ ist keine spezielle Versionsnummer mehr, sondern eine Bezeichnung für die Technologie - die derzeit aktuelle Version ist die Version 1.3 (wobei die Versionsnummern von SDK und Runtime Environment (JRE), welches aus dem Paket ausgelagert wurde, wiederum unabhängig voneinander gezählt werden).

3.2. Java 2 - Editions

In früheren Zeiten beinhaltete Java alle verfügbaren Klassen in einer einzigen Version. Dieses Konzept ließ sich auf Dauer nicht beibehalten, denn in der Version 1.1.7 umfaßte das JDK über 1600 Klassen in einer Datei mit einer Größe von 8,34 MB - zuviel für viele Endgeräte.[27]
Sun entschied sich, das Java-Paket in drei „Editionen“ aufzuteilen, die Java 2 Standard Edition, die Java 2 Enterprise Edition und die Java 2 Micro Edition[28] (vgl. Abbildung 1). Auf diese Weise können drei Produkte angeboten werden, welche auf bestimmte Zielgruppen zugeschnitten sind und jeweils die Klassen für eine Gruppe von Geräten beinhalten. Der Kern des Runtime Environments enthält nur noch wenige Klassen, die in allen drei Editionen zur Verfügung stehen müssen. Außerdem ist es auf diese Weise

[27] Vgl. Giguère, Eric, „Java 2 Micro Edition“, S. 48

[28] ebenda, S. 46 ff.

möglich, Lizenzen für weitergehende Funktionen zu verkaufen, die über die Basisfunktionalität hinausgehen - ein wichtiger Gesichtspunkt, denn Sun stellt die meisten Java-Produkte kostenlos zur Verfügung und hat so die Möglichkeit, zusätzliche Einnahmen zu erzielen und einen Teil der Entwicklungskosten zurück zu gewinnen.
In dieser Arbeit über mobile Endgeräte ist die Java 2 Micro Edition von zentralem Interesse, die J2SE und J2EE werden im folgenden nur kurz erwähnt.

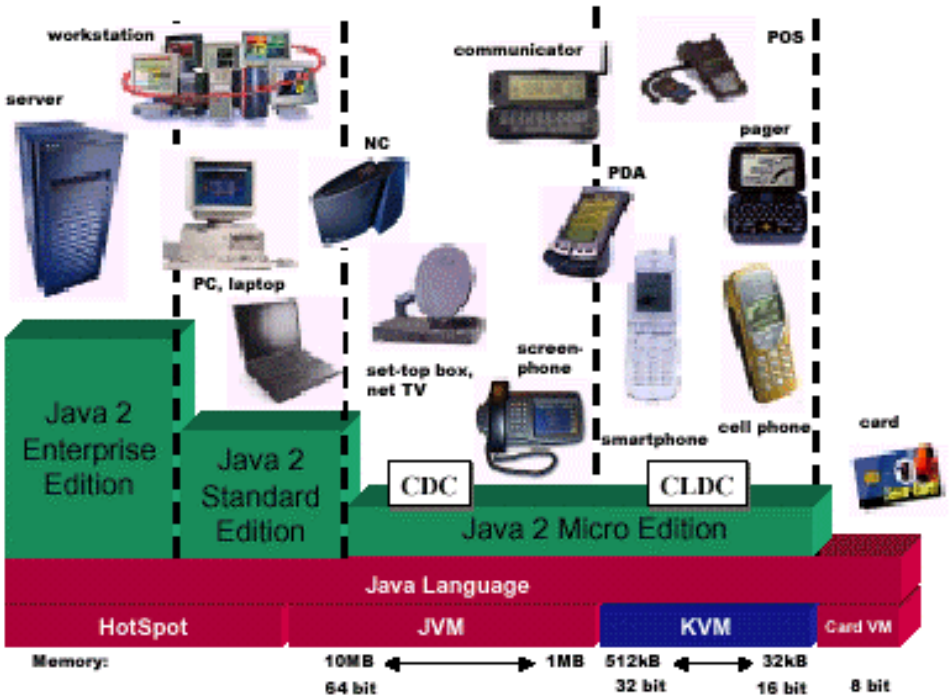

Abbildung 1: Plattformen und Zielmärkte von Java
Quelle: Mahmoud, Qusay, „The J2ME Platform - Which APIs Come from the J2SE Platform ?“, http://developer.java.sun.com/developer/technicalArticles/wireless/midpapi/ (Stand: 23.08.2001)

3.2.1. Java 2 Standard Edition

Die Java 2 Standard Edition (J2SE)[29] ist das Java-Hauptprodukt von Sun für alle konventionellen Java-Applikationen. J2SE ist damit der Nachfolger der Vorversionen (Java 1.2 und früher). J2SE stellt alle Klassen bereit, welche für die Entwicklung für PCs und kleinere Server benötigt werden und eignet sich sowohl für Einzelapplikationen als auch für Applets in Web-Browsern.

[29] Sun Microsystems, Java 2 Standard Edition, http://java.sun.com/j2se/

3.2.2. Java 2 Enterprise Edition

Mit der Java 2 Enterprise Edition (J2EE)[30] bietet Sun eine Version für Unternehmen zur Programmierung von Serveranwendungen. Zwar ist dies auch mit J2SE möglich, J2EE bietet aber zusätzlich eine Unterstützung von größeren serverbasierten Programmen, welche tausende Clients unterstützen und mit Nicht-Java-Applikationen interagieren müssen (CORBA, XML).

3.2.3. Java 2 Micro Edition

Die Java 2 Micro Edition (J2ME)[31] ist das zentrale Thema dieser Arbeit. Diese Version wendet sich an Entwickler für „Small Computing Devices“, also kleinere Rechner mit eingeschränkter Kapazität, beispielsweise mobile Endgeräte (vgl. Kapitel 1.1.).

Wie auch bei der J2SE und J2EE handelt es sich weder um eine genaue Spezifikation, noch um ein konkretes Produkt, also keine Software, die man herunterladen könnte. Vielmehr ist J2ME ein Marketing- bzw. Lizenz-Begriff, unter dem verschiedene Technologien für mobile Endgeräte zusammengefaßt sind.

J2ME wurde erstmals 1999 auf der JavaOne-Entwicklerkonferenz vorgestellt. Neu war einerseits die auf mobile Endgeräte abgestimmte Entwicklungsumgebung, eine außerordentlich wichtige Komponente ist aber die KVM, eine neue Virtual Machine (VM) für kleine mobile Endgeräte (im Sinne von Kapitel 1.1.). Auf die KVM werde ich in Kapitel 4.3.2. näher eingehen.

Mit J2ME entwickelte Applikationen lassen sich auf J2SE- und J2EE-Systemen ausführen, sofern APIs verwendet wurden, welche in allen drei Editionen enthalten sind. Allerdings ist diese plattformübergreifende Funktionalität eher selten erforderlich. Vielmehr kommt es darauf an, daß mit J2ME entwickelte Programme auf einer bestimmten Gruppe von Endgeräten ablaufen. Dies wird durch die flexible Gestaltung erreicht, denn die J2ME ist skalierbar, modular aufgebaut und läßt sich durch die Auswahl von entsprechenden Konfigurationen und Profilen genau auf das Endgerät abstimmen, für welches entwickelt wird. Wie aus Abb. 2 ersichtlich, setzen diese Konfigurationen (hier: CLDC) und Profile auf der Virtual Machine auf:[32]

---

[30] Sun Microsystems, Java 2 Enterprise Edition, http://java.sun.com/j2ee/

[31] Sun Microsystems, Java 2 Micro Edition, http://java.sun.com/j2me/

[32] Vgl. Sun Microsystems, „Java 2 Micro Edition (J2ME) Technology for Creating Mobile Devices, White Paper”, http://java.sun.com/products/cldc/wp/KVMwp.pdf (Stand: 23.08.2001)

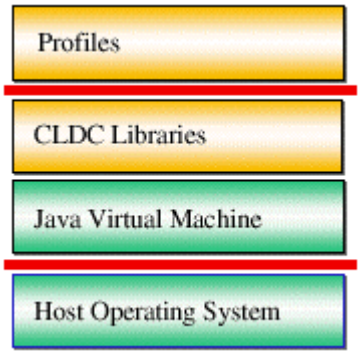

Abbildung 2: J2ME Software Layer Stack
Quelle: Sun Microsystems, „Java 2 Micro Edition (J2ME) Technology for Creating Mobile Devices, White Paper", S. 6
http://java.sun.com/products/cldc/wp/KVMwp.pdf (Stand: 23.08.2001)

Auf der untersten Schicht in der Stapelarchitektur, dem Betriebssystem des Endgerätes, basiert die Java Virtual Machine, die auf die Hardware des jeweiligen Gerätes zugeschnitten ist. Darauf baut die Konfigurationsschicht auf, welche die Minimalanforderungen der verfügbaren Java-Technologie beschreibt. Schließlich folgen ein oder mehrere Profile, welche diese Minimalanforderungen durch optional verfügbare Komponenten ergänzen.

Über die Frage, was genau die J2ME ist, gibt es einige weit verbreitete Irrtümer. Die wichtigsten seien an dieser Stelle klar gestellt:[33]

1. **Java-Programme funktionieren generell auf J2ME**
   Man kann nicht in jedem Fall davon ausgehen, daß alle Java-Programme ohne Änderungen auch mit J2ME ablaufen. Es gibt einige technische Restriktionen, welche beachtet werden müssen: CLDC beispielsweise unterstützt keine Fließkomma-Operationen, bei der J2ME fehlen einige aus der J2SE bekannte Klassen, des weiteren sind einige Klassen auf mobile Endgeräte abgestimmt bzw. zu diesem Zweck zusätzlich eingeführt. Programme müssen außerdem auf die Speichernutzung und Performance hin optimiert werden.

[33] Vgl. Giguère, Eric, „Fallacies About Java 2 Micro Edition",
http://www.ericgiguere.com/microjava/fallacies.html (Stand: 15.08.2001)

2. **KVM = J2ME**

   Oft werden die Begriffe KVM und J2ME synonym gebraucht, obwohl es unterschiedliche Dinge sind. Bei der KVM handelt es sich um eine Virtual Machine, die für die Nutzung auf mobilen Endgeräten optimiert ist. J2ME dagegen ist eine Reihe von Spezifikationen. Manche Konfigurationen der J2ME nutzen die KVM als Virtuelle Maschine, andere wiederum nicht - so setzt z.B. die CDC eine Classic VM voraus.

3. **KVM = CLDC**

   Auch die Begriffe KVM und CLDC werden oft gleich gesetzt. Wenn CLDC auch die KVM als Virtuelle Maschine erfordert, so handelt es sich doch bei Konfigurationen und Virtuellen Maschinen um verschiedene Dinge. Die in der CLDC vorgesehene VM ist zwar die KVM, doch könnte auch eine andere VM an diese Anforderungen angepaßt werden.

4. **J2ME wird WAP ersetzen**

   WAP ist eine Technologie für drahtlose Kommunikation, welche WML als Seitenbeschreibungssprache benutzt. Damit ist es etwas grundsätzlich anderes als eine Java-Technologie, auch wenn die Einsatzmöglichkeiten sich unter Umständen ähneln können. Allerdings müssen sich diese Technologien nicht gegenseitig ausschließen, es gibt einige Argumente, beide zu kombinieren. Mehr dazu in Kapitel 5.2.

5. **Um Java zu erlernen, beginne mit J2ME**

   Bei J2ME mag es sich zwar in gewisser Weise um eine eingeschränkte Java-Version handeln, doch ist J2ME daher nicht unbedingt einfacher zu erlernen. J2ME verfügt über Konfigurationen, Profile und andere Besonderheiten, die in der J2SE nicht existieren. Mit Hilfe der J2SE ist es leichter, Java zu erlernen, da diese Besonderheiten dort nicht zu berücksichtigen sind.

3.3. Konfigurationen

Mobile Endgeräte wie Mobiltelefone und PDAs unterscheiden sich in ihren Funktionen, Ausstattung und ihrer äußeren Form. Aus diesem Grund unterstützt die J2ME-Plattform mit ihrer kleinen Virtual Machine nur eine kleine Anzahl von APIs, welche für alle Endgeräte von Bedeutung sind.[34]
Mit Hilfe von Konfigurationen wie der Connected Limited Device Configuration (CLDC) und der Connected Device Configuration (CDC) werden weitere APIs zur Verfügung gestellt, die jeweils eine Produktfamilie mit ähnlichen Anforderungen hinsichtlich des Speichers und der Prozessorgeschwindigkeit unterstützen („*horizontales* Marktsegment"). Konfigurationen legen die unterstützten Funktionen der Java-Programmiersprache, die erforderlichen Funktionen der Virtual Machine (vgl. Kapitel 4) sowie die Java-Bibliotheken und APIs fest, welche auf jedem Gerät in dieser Gerätefamilie *mindestens* zur Verfügung stehen müssen. Somit bildet eine Konfiguration den „kleinsten gemeinsamen Nenner" der Java-Funktionen und Bibliotheken, auf deren Vorhandensein auf dem Endgerät sich der Programmierer verlassen kann. Hier festgelegte Spezifikationen sind grundsätzlich bindend - optionale Komponenten können lediglich in Profilen festgelegt werden (vgl. Kapitel 3.4.).

Derzeit gibt es für die J2ME zwei Konfigurationen:

- **Connected Limited Device Configuration (CLDC)**
  - für Mobiltelefone, Pager, Organizer etc.
  - benötigt die KVM als Virtual Machine
- **Connected Device Configuration (CDC)**
  - für Set-Top-Boxen, Fahrzeug-Navigationssysteme etc.
  - arbeitet mit der Classic VM

Beide Konfigurationen dienen der Programmierung von Endgeräten mit Netzwerkanbindung, wobei die CLDC Geräte mit eingeschränkten Ressourcen anspricht, zu denen in der Regel mobile Endgeräte wie Mobiltelefone und PDAs gehören.

[34] Vgl. Sun Microsystems, „The J2ME Platform - Which APIs Come from the J2ME Platform ?", http://developer.java.sun.com/developer/technicalArticles/wireless/midpapi/ (Stand: 23.08.2001)

Mit steigender Prozessorleistung und Speicherkapazität wird in der Zukunft vielleicht auch die CDC für mobile Endgeräte geeignet sein. Die Entscheidung, welche Konfiguration ein Produkt unterstützt, obliegt dem Hersteller des Gerätes.

Die Konfigurationen ermöglichen u.a. auch eine „dynamic application delivery", also die dynamische Verteilung von Applikationen.[35] Hiermit ist die Möglichkeit gemeint, neue Anwendungen per Netzwerkverbindung in das Gerät herunterzuladen - ähnlich einem Applet in einem Webbrowser. Dies ist eine wichtige Voraussetzung für Dritthersteller von Software. Ist die Entwicklung zusätzlicher Software durch Dritthersteller nicht erwünscht, bietet sich statt der J2ME die Verwendung von EmbeddedJava an (vgl. Kapitel 2.3.2.).

Welche Konfiguration auf dem jeweiligen Endgerät eingesetzt wird, läßt sich aus Programmierersicht mit Hilfe der Methode `System.getProperty` feststellen, welche in der Variable `microedition.configuration` den entsprechenden Wert ausgibt, beispielsweise `CLDC-1.0`.

3.3.1. Connected Limited Device Configuration (CLDC)

Connected Limited Device Configuration (CLDC)[36] ist die eine der beiden momentan existierenden Konfigurationen für J2ME. Sie liegt in der Version 1.0.2 vor und wurde in dieser Fassung im Mai 2000 vom JCP verabschiedet.[37] Diese Konfiguration bietet Unterstützung für mobile Endgeräte, die über eine Netzverbindung, aber eingeschränkte Ressourcen verfügen. Sie enthält Spezifikationen, welche Anforderungen eine Virtuelle Maschine erfüllen muß, damit die auf CLDC basierenden Profile auf den entsprechenden Endgeräten funktionieren.
Mittlerweile sind einige andere Virtual Machines - die den eigentlichen Kern der J2ME darstellen - auf dem Markt, welche diesen Anforderungen entsprechen, doch die bisher einzige Referenz-Implementation einer solchen VM von Sun Microsystems ist die „KVM". Näheres zu den Virtual Machines folgt in Kapitel 4.

---

[35] Vgl. Giguère, Eric, „Java 2 Micro Edition", S. 91

[36] Sun Microsystems, "CLDC and the K Virtual Machine (KVM)", http://java.sun.com/products/cldc/ (Stand: 29.08.2001)

[37] Java Community Process, JSR-30, http://jcp.org/jsr/detail/30.jsp (Stand: 30.07.2001)

Ziele der CLDC

Die CLDC dient dem Zweck, einen Standard für eine Java-Plattform für mobile Endgeräte zu definieren. Sie soll gleichzeitig die dynamische Verteilung von Software über die Netzverbindung sicherstellen (siehe oben) und außerdem Drittherstellern die Entwicklung von Software für mobile Endgeräte ermöglichen.[38]

Die CLDC soll Minimalanforderungen an die verwendete Hardware definieren, um so - mit Hilfe der Profile, welche auf sie aufsetzen - eine möglichst große Portabilität zwischen den Geräten zu gewährleisten, für die sie konzipiert ist. Dabei soll die CLDC mit ihrer Virtual Machine und den benötigten Klassenbibliotheken nicht mehr als 128 KB Speicher benötigen, außerdem dürfen die Java-Applikationen nicht mehr als 32 KB Stapelspeicher (Heap) belegen.

Spezifikationen

Um diese Ziele zu erreichen, wurden Spezifikationen in den folgenden Bereichen festgelegt:[39]

- Java-Sprache und Ausstattung der Virtual Machine:
  Es werden keine Fließkomma-Datentypen wie `float` und `double` unterstützt, die Methode `Objects.finalize()` existiert nicht, es gibt nur eine eingeschränkte Fehlerbehandlung, die meisten Klassen aus `java.lang.error` sind nicht implementiert. Fehler werden auf andere Weise abgefangen.
- Inhalt der Java-Kernbibliotheken `java.lang.*`, `java.util.*`
- Ein- und Ausgabefunktionen
- Netzwerkfunktionen
- Sicherheit des Systems:
  Neue heruntergeladene Klassen müssen eine Überprüfung durchlaufen, Applikationen sind voneinander mittels „Sandbox“-Technologie getrennt, Klassen des Systems können nicht von externen Applikationen überschrieben werden.
- Internationalisierung

[38] Sun Microsystems, „Java 2 Micro Edition (J2ME) Technology for Creating Mobile Devices, White Paper”, S. 19 ff.

[39] ebenda, S. 20

Es gibt weitere wichtige Spezifikationen, die jedoch *nicht* in den Konfigurationen, sondern in den darauf aufbauenden Profilen festgelegt werden:

- Lebenszyklus einer Applikation (application life-cycle management: Installation, Ablauf, Deinstallation)
- Benutzerschnittstelle (user interface)
- Behandlung von Ereignissen (event handling)
- Interaktion zwischen Benutzer und Applikation (high-level application model)

Es ist nicht einfach, die Spezifikationen, welche die CLDC betreffen, von denen der Virtual Machine (vgl. Kapitel 4) zu trennen. Es wurden Aufgaben, die ansonsten bei Java der VM zufallen, in die Spezifikation der CLDC verlagert.
So ist es beispielsweise erforderlich, daß eine Java-VM in der Lage ist, ungültige bzw. unerlaubte Klassen zu erkennen und von der Ausführung auszuschließen. Da diese Aufgabe aufgrund des stark minimierten Umfanges der VMs und der mangelnden Ressourcen auf mobilen Endgeräten kaum zu bewältigen ist, erfolgt dies nun entgegen der bisherigen Java-VM-Spezifikation gemäß CLDC auf andere Weise: jede Java-Klassendatei enthält ein Attribut „`stackmap`“. Dieses Attribut wird bereits auf dem Server oder PC vor dem Herunterladen auf das mobile Endgerät von einem Programm erzeugt, welches jede Methode der Klassendatei analysiert („pre-verification“). Auf diese Weise vergrößert sich zwar das Classfile um ca. 5 %, doch die Prüfung auf dem Endgerät kann wesentlich schneller erfolgen, da die ressourcenintensiven Teile der Prüfung auf den Server oder PC verlagert und dort durchgeführt wurden.

Das Format der Klassendateien ist zwingend auf Java Archive Files (JAR) festgelegt. Hierbei handelt es sich um komprimierte Archivdateien, welche alle Klassen in einer einzigen Datei beinhalten. So wird die Distribution der Applikationen über die Mobilfunknetze erleichtert, da auf diese Weise nur eine einzige Datei übertragen werden muß.

Klassenbibliotheken
In der CLDC ist eine Teilmenge der Klassen aus der J2SE und J2EE enthalten - hierbei handelt es sich aus Kompatibilitätsgründen um dieselben Klassen wie auch in den anderen Java-Versionen. Allerdings sind einerseits aus naheliegenden Gründen (Ressourcenmangel) nicht alle Klassen vollständig übernommen worden, sondern nur diejenigen, die sich für den Einsatz auf mobilen Endgeräten eignen. Auf der anderen Seite

sind CLDC-spezifische Klassen hinzugefügt worden, welche bisher in den anderen Java-Editionen nicht benötigt wurden (vgl. Abbildung 3).

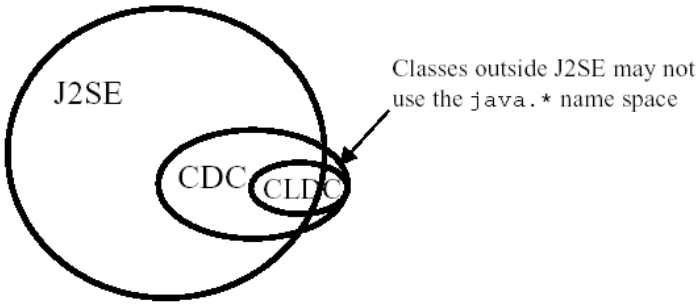

Abbildung 3: Die Beziehung der J2ME-Konfigurationen zur J2SE
Quelle: Sun Microsystems, „Java 2 Micro Edition (J2ME) Technology for Creating Mobile Devices, White Paper", S. 16

Klassen der CLDC, die aus der J2SE übernommen wurden:[40]

- Systemklassen aus `java.lang`:
  `Object, Class, Runtime, System, Thread, Runnable, String, StringBuffer, Throwable`
- Datentyp-Klassen aus `java.lang`:
  `Boolean, Byte, Short, Integer, Long, Character`
- Collection-Klassen aus `java.util`:
  `Vector, Stack, Hashtable, Enumeration`
- I/O-Klassen aus `java.io`:
  `InputStream, OutputStream, ByteArrayInputStream, ByteArray-OutputStream, DataInput, DataOutput, DataInputStream, DataOutputStream, Reader, Writer, InputStreamReader, Output-StreamWriter, PrintStream`
- Kalender- und Zeit-Klassen aus `java.util`:
  `Calendar, Date, TimeZone`
- zusätzliche Hilfsklassen:
  `java.util.Random, java.lang.Math`

[40] ebenda, S. 24 ff.

- "Exception"-Klassen:
  - aus `java.lang`:
    `Exception, ClassNotFoundException, IllegalAccessException, InstantiationException, InterruptedException, RuntimeException, ArithmeticException, ArrayStoreException, ClassCastException, IllegalArgumentException, IllegalThreadStateException, NumberFormatException, IllegalMonitorStateException, IndexOutOfBoundsException, ArrayIndexOutOfBoundsException, StringIndexOutOfBoundsException, NegativeArraySizeException, NullPointerException, SecurityException`
  - aus `java.util`:
    `EmptyStackException, NoSuchElementException`
  - aus `java.io`:
    `EOFException, IOException, InterruptedException, UnsupportedEncodingException, UTFDataFormatException`
- Fehlerbehandlungsklassen aus `java.lang`:
  `Error, VirtualMachineError, OutOfMemoryError`

Es existieren Einschränkungen bezüglich des Unicode-Zeichensatzes, außerdem werden die Klassen aus `java.util.Properties` der Java 2 Standard Edition nicht unterstützt. Es lassen sich einige Einstellungsvariablen mit dem Namen `microedition` über die Methode `System.getProperty(String key)` abfragen.

CLDC-spezifische Klassen:

Die Bibliotheken der J2SE und J2EE bieten in Form der Klassen `java.io` und `java.net` umfangreiche Möglichkeiten zur Daten-Ein- und -Ausgabe. Diese sind jedoch aufgrund ihrer Größe nicht vollständig im kleinen Speicher mobiler Endgeräte abzulegen, so daß entsprechend angepaßte Klassen verwendet werden. Dieses sogenannte „Generic Connection Framework" befindet sich im Paket `javax.microedition.io` und stellt die folgenden Klassen zur Verfügung:[41]

[41] Vgl. Giguère, Eric, „Java 2 Micro Edition", S. 99 ff.

- `javax.microedition.io`:
  `Connection, ConnectionNotFoundException, Connector, ContentConnection, Datagram, DatagramConnection, InputConnection, OutputConnection, StreamConnection, StreamConnectionNotifier`

Mit Hilfe dieser Klassen kann die gesamte I/O- und Netzwerk-Kommunikation abgewikkelt werden. Wenn auch keine konkreten Protokolle in der CLDC implementiert sind, so ist es mittels des Generic Connection Frameworks doch möglich, Protokolle wie z.B. HTTP, Sockets oder Files unter Verwendung einer einheitlichen Syntax anzusprechen. Da es sich um eine Konfiguration für vernetzte Endgeräte handelt, wird erwartet, daß von Seiten des Herstellers zumindest eines dieser Protokolle, vorzugsweise HTTP, implementiert wird.

3.3.2. Connected Device Configuration (CDC)

Ziele der CDC

Gegenüber der CLDC spricht die Connected Device Configuration (CDC) ebenfalls vernetzte Endgeräte an, jedoch solche, die nicht oder nicht wesentlich in ihren Energie- und Speicher-Ressourcen oder in der Netzwerkbandbreite eingeschränkt sind. Daher ist es auch nicht erforderlich, eine modifizierte Virtual Machine wie die KVM einzusetzen - die CDC erfordert eine gewöhnliche Classic VM, wie sie auch bei J2SE zum Einsatz kommt.

„The CDC is perhaps best described as a stripped-down J2SE to which the CLDC classes were added.[42]" - Bei der CDC handelt es sich um eine eingeschränkte J2SE, in welche die Klassen der CLDC integriert sind. Somit funktionieren alle Programme der CLDC-Spezifikation auch mit der CDC.

Die Connected Device Configuration wendet sich an Endgeräte, die über mindestens 512 KB nichtflüchtigen Hauptspeicher und mindestens 256 KB flüchtigen Speicher verfügen sowie hinsichtlich der Stromversorgung nicht eingeschränkt sind.

[42] ebenda, S. 101

Spezifikationen

Die Spezifikationen der CDC wurden im März 2001 in der JSR-36 durch den JCP festgelegt.[43] Sie umfassen die folgenden Regelungen:

- volle Unterstützung der Java-Sprache gemäß der Java Language Specification (JLS)[44]
- volle Unterstützung der Java Virtual Machines gemäß der Java Virtual Machine Specification
- alle Klassen der CLDC werden unterstützt, insbesondere das Paket `javax.microedition`
- alle anderen Klassen entsprechen denen der J2SE 1.3

Klassenbibliotheken

Neben den oben genannten Klassen der CLDC sind alle Klassen der J2SE verfügbar, beispielsweise aus den folgenden Paketen:

- `java.lang`
- `java.lang.reflect`
- `java.io`
- `java.net`
- `java.security`
- `java.lang.security.cert`
- `java.lang.security.spec`
- `java.text`
- `java.text.resources`
- `java.util`
- `java.util.jar`
- `java.util.zip`

Es fehlen - wie auch bei der CLDC - die Klassen zur Ansteuerung der Benutzerschnittstelle (user interface); diese sind in den Profilen enthalten, welche auf der Konfiguration aufbauen.

[43] Java Community Process, JSR-36, http://jcp.org/jsr/detail/036.jsp (Stand: 30.07.2001)

[44] Java Community Process, JSR-901, http://jcp.org/jsr/detail/901.jsp (Stand: 02.08.2001)

3.4. Profile

Die Konfigurationen, wie sie im vorigen Kapitel vorgestellt wurden, bilden die Basis für die Programmierung auf mobilen Endgeräten. Doch in den Konfigurationen fehlen wichtige Klassen, die eine konkrete Unterstützung für die Hardware des Gerätes bieten. Diese werden erst durch Profile ergänzt, welche auf den Konfigurationen CLDC oder CDC aufsetzen (vgl. Abbildung 4).

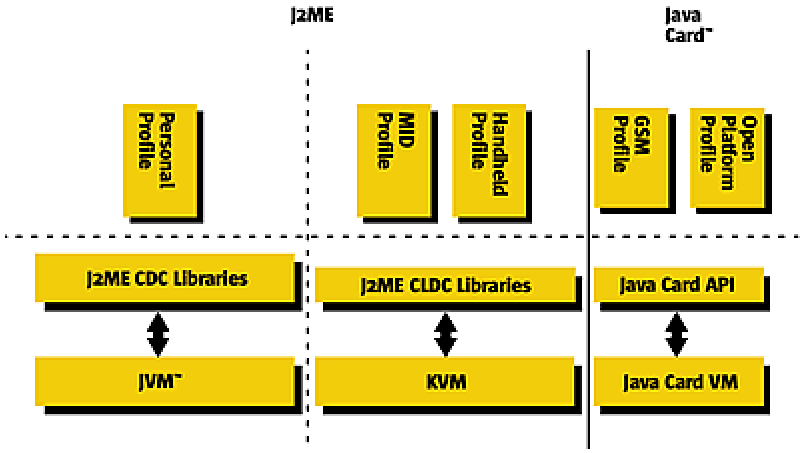

Abbildung 4: J2ME und ihre Profile im Überblick
Quelle: Glahn, Kay, Meyen, Sebastian, „Die Straßen von SF“, Java Magazin, Heft 8/2001, http://www.javamagazin.de/ausgaben/2001/8/artikel/3/online.shtml (Stand: 24.07.2001)

Ein Profil stellt dabei APIs zur Verfügung, welche auf die Hardware einer bestimmten Gerätefamilie mit ähnlichen Anwendungsmöglichkeiten zugeschnitten sind („*vertikales* Marktsegment“), beispielsweise Palm-Organizer.
Applikationen, die für ein bestimmtes Profil geschrieben wurden, sind auf allen anderen Geräten lauffähig, die ebenfalls dieses Profil unterstützen. Dies ist ein wesentlicher Bestandteil der mit der J2ME verfolgten Philosophie, denn nur so kann eine Portabilität der Anwendungen zwischen ähnlichen Endgeräten gewährleistet werden (beispielsweise innerhalb der Gruppe der Mobiltelefone, von PDA zu PDA oder zwischen Haushaltsgeräten wie z.B. Waschmaschinen[45]).
Diese durch Profile erreichte vertikale Portabilität ist in der Regel für den Endbenutzer von wesentlich höherer Bedeutung als die horizontale Portabilität zwischen Endgeräten mit ähnlicher Hardwareausstattung. Es ist schließlich wenig sinnvoll, eine Anwendung

[45] Sun Microsystems, „Java 2 Micro Edition (J2ME) Technology for Creating Mobile Devices, White Paper”, S. 13

des Mobiltelefons zum Mobile Banking[46] gegebenenfalls auch in einer Waschmaschine ausführen zu können, nicht aber auf dem Mobiltelefon eines anderen Herstellers.
Allerdings sind nicht nur gerätespezifische Profile denkbar, sondern auch applikationsspezifische. Ein Endgerät kann ein oder mehrere Profile unterstützen. Dies erlaubt es, nicht nur vollständige Profile zu erstellen, welche die Entwicklung komplexer Anwendungen für eine bestimmte Gerätekategorie (wie z.B. Pager, Mobiltelefone, Waschmaschinen oder interaktive elektronische Spielzeuge) ermöglichen, sondern auch „Teilprofile“ zu erstellen, welche die für die Ausführung einer bestimmten Applikation erforderlichen Klassen enthalten, um dann auf den unterschiedlichsten Endgeräten eingesetzt zu werden. Ist ein Profil darauf ausgerichtet, die Voraussetzungen für eine bestimmte Mobile-Banking-Anwendung zu schaffen, so ist diese sowohl auf PDAs als auch auf Mobiltelefonen lauffähig, sofern diese das Profil unterstützen.
Welche Profile in einem Gerät verfügbar sind, läßt sich analog den Konfigurationen über die Methode `System.getProperty` herausfinden, sie liefert in der Variable `microedition.profiles` den entsprechenden Wert zurück, beispielsweise „`MIDP-1.0`“, ggf. auch mehrere Profile durch Leerzeichen getrennt.

Zur Zeit befinden sich mehrere Profile in der Entwicklungsphase des Java Community Process. Fertiggestellt sind zum jetzigen Zeitpunkt zwei Profile, das Mobile Information Device Profile (MIDP)[47] und das Java Foundation Profile[48].

3.4.1. Mobile Information Device Profile (MIDP)

Die Grundlagen des MIDP
Das MIDP ist das erste vom JCP verabschiedete Profil für die CLDC, es wurde als JSR-37 bereits im September 2000 in seiner finalen Version vorgestellt. Aus diesem Grund ist es annähernd das einzige Profil, welches sowohl in der Literatur häufig aufgegriffen wird, als auch bereits von Geräteherstellern in der Praxis implementiert worden ist. An der Entwicklung beteiligt waren unter der Federführung von Motorola 21 Firmen, darunter alle führenden Hersteller von Mobiltelefonen wie z.B. Siemens, Nokia

[46] Eine überzeugende Anwendung im Bereich Mobile Banking hat die schweizer Ergon AG für die Credit Suisse entwickelt: „youtrade“ ist eine Broking-Anwendung für Palm-PDAs, http://www.credit-suisse.ch/de/direktzugriff/pcbank/youtrade/index.html (Stand: 10.09.2001)

[47] Java Community Process, JSR-37, http://jcp.org/jsr/detail/37.jsp (Stand: 14.08.2001)

[48] Java Community Process, JSR-46, http://jcp.org/jsr/detail/46.jsp (Stand: 14.08.2001)

und Ericsson. Allein schon aufgrund dieser Tatsache läßt sich absehen, daß MIDP in Zukunft ein weit verbreiteter Standard sein wird.
Daher wird das MIDP im folgenden detaillierter besprochen; andere Profile, welche sich noch in der Entwicklungsphase befinden, folgen im Anschluß in kürzerer Form.
Sun Microsystems stellt eine Referenzimplementation für Microsoft Windows[49] sowie eine erste „EarlyAccess"-Implementation für PalmOS-Geräte[50] zur Verfügung (siehe unten).

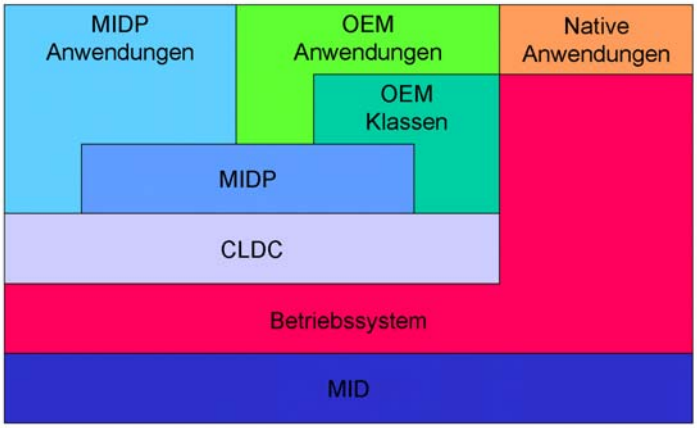

Abbildung 5: Der Aufbau der MIDP-Architektur
Quelle: Glahn, Kay, „Java macht mobil", Java Magazin, Heft 3/2001

Das MIDP baut auf der CLDC auf (vgl. Abbildung 5) und stellt APIs aus den folgenden Bereichen zur Verfügung:[51]

- **Definition und Verwaltung/Kontrolle von Applikationen**
  Das MIDlet wird analog den Java-Applets mit Methoden gesteuert, um die Anwendung zu starten, anzuhalten oder vollständig zu beenden (siehe Beispiel).
- **Anzeige von Text und Grafik**
  MIDP bietet Klassen für die Benutzerschnittstelle (user interface).

[49] Sun Microsystems, MIDP Reference Implementation for Microsoft Windows, http://javashoplm.sun.com/ECom/docs/Welcome.jsp?StoreId=5&PartDetailId=MIDP-1.0-WIN-G-CS&TransactionId=communitySource (Stand: 30.08.2001)

[50] Sun Microsystems, Mobile Information Device Profile (MIDP) for Palm OS, http://java.sun.com/products/midp/palmOS.htm (Stand: 30.08.2001)

[51] Vgl. Giguère, Eric, „Java 2 Micro Edition", S. 109

- **Speicherung in einfachen Datenbanken**
  MIDP bietet die Möglichkeit, mit einfachen Datenbank-Mechanismen Daten bis zum nächsten Aufruf der Applikation zu konservieren.
- **Netzwerkanbindung via HTTP**
  Über die Klasse `HttpConnection` wird das Generic Connection Framework der CLDC (siehe oben) erweitert.

- Definition und Verwaltung/Kontrolle von Applikationen

Bei Aktivierung einer Funktion auf dem Endgerät (Start / Stop / Pause) wird die entsprechende Methode des MIDlets aufgerufen (siehe Beispiel), die weitere Behandlung erfolgt im Prinzip wie in entsprechenden Java-Programmen der J2SE.

Als Beispiel ein einfaches MIDlet zur Ausgabe einer Text-Zeile:[52]

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

public class HalloWeltMIDlet extends MIDlet implements CommandListener {
   private Command exitCommand; // Befehl zum Beenden
   private Display display;     // Display definieren

   public HalloWeltMIDlet() {
      display = Display.getDisplay(this);
      exitCommand = new Command("Beenden", Command.SCREEN, 2);
                // Initialisierung
   }

   public void startApp() {
                // Das MIDlet wurde aktiviert.
                // Datenbanken oeffnen, Benutzerschnittstelle aktivieren etc.
      TextBox t = new TextBox("Hallo Welt ...",
                              "nur ein kleiner MIDlet-Test !", 256, 0);
      t.addCommand(exitCommand);
      t.setListener(this);
      display.setCurrent(t);
   }

   public void pauseApp() {}
                // Das MIDlet wurde voruebergehend deaktiviert.
                // So viele Ressourcen freigeben wie möglich !
```

[52] Eine Sammlung kostenlos erhältlicher MIDlets findet sich bei JavaMobiles, http://www.javamobiles.com/midlets/index.html (Stand: 06.08.2001)

```
    public void destroyApp(boolean unconditional) {}
                // Das MIDlet wurde beendet.
                // Datenbanken schließen, Verbindungen beenden etc.

    public void commandAction(Command c, Displayable s) {
                // Aktion: Befehl wurde aktiviert !
       if (c == exitCommand) {
          destroyApp(false);
          notifyDestroyed();
       }
    }
}
```

- Anzeige von Text und Grafik

Im Bereich der Benutzerschnittstelle stellt das MIDP APIs zur Verfügung, welche nicht auf den bekannten AWT-Klassen der J2SE basieren. Diese wären zu ressourcenintensiv für mobile Endgeräte, so daß speziell für den Einsatz auf diesen Geräten eigene APIs definiert werden mußten.[53]

MIDP bietet zweierlei Klassen, das sogenannte High-Level API und das Low-Level-API. Mit Hilfe des eher aufgabenorientierten High-Level-APIs läßt sich festlegen, welche Objekte am Bildschirm angezeigt werden (z.B. Meldungen, Auswahllisten oder Textboxen), während die Klassen des hardwarenahen Low-Level-APIs eine direkte Ansteuerung des Bildschirmes etc. zulassen. Darüber hinaus können je nach MIDP-Implementation eigene Klassen des Herstellers enthalten sein, um die Benutzerschnittstelle zu steuern. Allerdings führt die Verwendung von Low-Level-APIs und proprietärer Klassen zu einer Einschränkung der Portabilität, da sich die MIDP-Endgeräte technisch unterscheiden (beispielsweise in der Bildschirmgröße) und so eine korrekte Wiedergabe auf anderen Endgeräten u.U. nicht immer gewährleistet ist. Dafür bieten sie die Möglichkeit, die Hardware sehr genau anzusteuern, um auf dem entsprechenden Endgerät eine sehr präzise Darstellung zu erhalten, wie es z.B. für Spiele sinnvoll ist.

- Speicherung in einfachen Datenbanken

Um eine Möglichkeit zu bieten, Daten auf dem Endgerät abzulegen, die trotz Ausschaltens bis zum nächsten Aufruf der Anwendung gespeichert bleiben, definiert das MIDP eine Reihe von Klassen im Paket `javax.microedition.rms`. Hierbei handelt es sich um das Record Management System (RMS). Dieses legt die Daten in einfachen Datenbanken ab (ähnlich dem PalmOS-Datenbankformat). Auf diese Datenbank können alle MIDlets zugreifen, die aus dem gleichen JAR-Archiv stammen, wie die Appli-

[53] ebenda, S. 115 ff.

kation, welche die Datenbank angelegt hat. Die Daten werden automatisch gelöscht, wenn das MIDlet vom Endgerät entfernt wird.[54]
Die Details des verwendeten Formates sind für den Programmierer nicht sichtbar, d.h. in Programmen können nur die im RMS implementierten Methoden benutzt werden, auch wenn das tatsächlich verwendete Format evtl. einem vom Endgerät unterstützten Format entspricht. Diese Einschränkung ist darin begründet, daß nur so eine vollständige Portabilität erhalten bleiben kann - schließlich wäre es möglich, daß andere MIDP-Endgeräte andere Datenbankformate benutzen und der Zugriff dort nicht mehr funktionieren würde. Es ist zwar für Gerätehersteller möglich, weitere Datenbankroutinen im MIDP zu integrieren, doch wird die Portabilität der Applikation deutlich eingeschränkt, wenn sie diese proprietären Methoden verwendet.

- Netzwerkanbindung via HTTP

Das MIDP ergänzt das Generic Connection Framework um einen Kommunikationstreiber für HTTP.[55] Er bietet die Funktionalität der GET-, POST- und HEAD-Methoden des HTTP-1.1-Protokolls.
Zu beachten ist, daß mittels der Methode `setRequestProperty` die Variablen `User-Agent` und `Content-Language` gesetzt werden müssen, um die Software (z.B. „`Profile/MIDP-1.0 Configuration/CLDC-1.0`“) und Sprachversion (z.B. „`en-US`“) des Endgerätes an den Server zu übermitteln.
Die Verbindung muß gemäß HTTP-Spezifikation nicht zwingend über TCP/IP aufgebaut werden, es ist unter Verwendung eines Gateways auch ein anderes Trägerprotokoll wie z.B. WAP oder i-Mode denkbar.[56]
Das MIDP sieht neben HTTP kein weiteres Protokoll vor. Es ist jedoch dem Gerätehersteller freigestellt, ein weiteres Protokoll zu implementieren.

Anforderungen an das Endgerät

Wie der Name schon sagt, zielt das MIDP auf Mobile Information Devices, also mobile Endgeräte. Darunter fallen Mobiltelefone, Pager und PDAs mit Netzanbindung. Die Geräte müssen über die folgenden Voraussetzungen verfügen:[57]

[54] ebenda, S. 120 ff.

[55] ebenda, S. 123 ff.

[56] Vgl. Glahn, Kay, „Java macht mobil“, Java Magazin, Heft 3/2001

[57] ebenda

Anzeige	- Größe: mindestens 96 x 54 Pixel - Farbtiefe: mindestens 1 bit - Seitenverhältnis: ca. 1:1
Eingabegeräte (alternativ)	- Einhandtastatur: ITU-T-Telefontastatur (1-9, *, #) - Zweihandtastatur: QWERTY-Tastatur - Touchscreen
Speicher (zusätzlich zum Speicherbedarf der CLDC)	- mind. 128 KB nichtflüchtiger Speicher für MIDP-Komponenten - mind. 8 KB nichtflüchtiger Speicher für persistente Daten der Anwendungen - mind. 32 KB flüchtiger Speicher für die Java Runtime (Heap)
Netzwerkanbindung	- duplex (two-way) - Funknetzwerk - keine ständige Verbindung erforderlich - eingeschränkte Bandbreite

Das Betriebssystem des Endgerätes muß ebenfalls bestimmte Aufgaben erfüllen:
Es ist ein einfacher Kernel erforderlich, um eine Java VM auszuführen. Lese- und Schreibvorgänge auf den nichtflüchtigen Speicher sind erforderlich, ebenso Lese- und Schreibzugriff auf das Funk-Netzwerk. Des weiteren muß die Möglichkeit bestehen, ein grafisches Display zu beschreiben sowie Benutzereingaben von mindestens einem der drei möglichen Eingabegeräte (siehe oben) entgegenzunehmen.

Application Manager

Java-Anwendungen im Rahmen des MID-Profils heißen „MIDlets“, von denen mehrere im nichtflüchtigen Speicher des Endgerätes abgelegt werden können. Es gibt zwei Typen von MIDlets: permanente MIDlets, welche vom Anwender ausgeführt werden können und System-MIDlets, welche für gerätespezifische Aufgaben zuständig sind, eine bevorzugte Ausführung genießen und mitunter für den Anwender nicht sichtbar sind.
Das JAR-Archiv, in dem alle zu einer Anwendung gehörenden MIDlets, Klassen und Ressourcen enthalten sind, heißt „MIDlet Suite“. Hierdurch wird das Herunterladen und

die Installation einer Anwendung wesentlich erleichtert, da nur eine Datei übertragen werden muß und nicht alle Klassenbibliotheken einzeln.
Zu diesem Zweck muß ein „Java Application Manager“ (JAM) auf dem Endgerät enthalten sein, welcher ein Herunterladen und Verwalten der MIDP-Anwendungen erlaubt. Dies ist in diesem Fall eine Aufgabe des Betriebssystems, der Application Manager wird nicht grundsätzlich vom MIDP zur Verfügung gestellt. Sun Microsystems bietet jedoch eine Beispiel-Implementation eines solchen Managers an, welche als Grundlage für eigene Entwicklungen genutzt werden kann.
Der Application Manager ist für die Auswahl des Übertragungsweges zuständig, über ihn kann die Funkverbindung oder andere Wege wie beispielsweise die Kommunikation via Infrarot (IrDA) oder eine serielle Schnittstelle ausgewählt werden. MIDlets werden vom Application Manager aus verwaltet, d.h. von hier aus können Programme von einem Webserver heruntergeladen werden (vgl. Abbildung 6), gestartet oder zur Deinstallation mit allen zugehörigen Daten vom Endgerät entfernt werden.

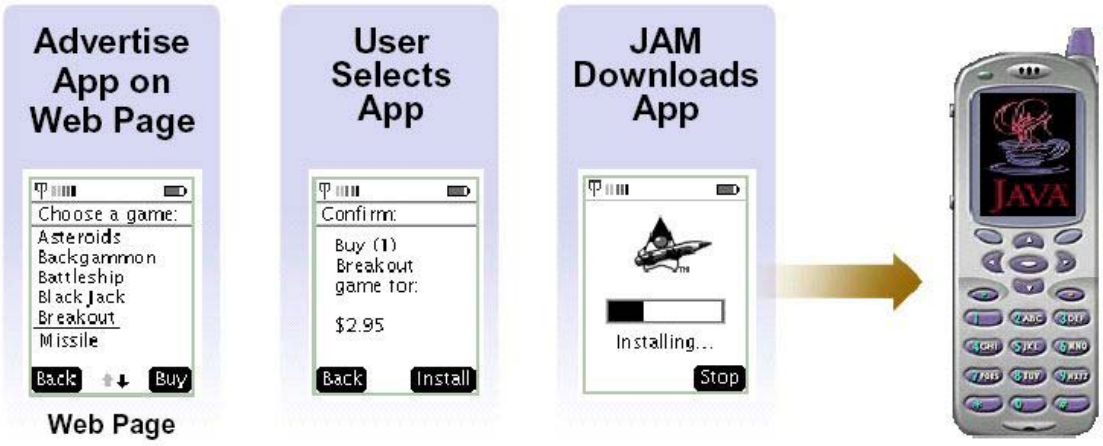

Abbildung 6: Der Java Application Manager
Quelle: Llobregat, Fernando, “Developing Applications Using the MIDP/UI APIs”, JavaOne-Conference 2000 San Francisco,
http://servlet.java.sun.com/javaone/javaone2000/pdfs/TS-1666.pdf (Stand: 13.08.2001)

Der Application Manager führt außerdem eine Versionskontrolle durch, welche verhindert, daß aktuelle Anwendungen durch ältere Versionen überschrieben werden bzw. daß dieselbe Version einer Anwendung versehentlich doppelt heruntergeladen wird. Dies geschieht mit Hilfe eines Application Descriptors.

Application Descriptor

Der Application Manager muß, ohne die Datei selbst herunterzuladen, feststellen können, um welche Art von Anwendung es sich bei einer JAR-Datei handelt, welche Version sie enthält und welche Anforderungen sie an das Endgerät stellt. Zu diesem Zweck gibt es *außerhalb* der JAR-Datei einen „Application Descriptor“ (JAD). Dabei handelt es sich um eine Textdatei, die Informationen zu den in der JAR-Datei enthaltenen MIDlets enthält. Die Verwendung ist optional, doch es wird empfohlen, eine solche Datei zu jeder Anwendung zu erstellen, denn sie erspart dem Benutzer u.U. den Zeitaufwand und die Kosten, die Datei vergeblich herunterzuladen.

Die Application-Descriptor-Datei trägt die Endung `.jad`, welche auf dem Webserver, auf dem die Datei liegt, mit der korrekten MIME-Bezeichnung konfiguriert sein muß (`text/vnd.sun.j2me.app-descriptor`).

Innerhalb der JAD-Datei werden die folgenden Attribute verwendet, um die Anwendung näher zu beschreiben:

`MIDlet-Jar-Size`	obligatorisch	Größe der MIDlet-Suite in Byte
`MIDlet-Jar-URL`	obligatorisch	URL zum Herunterladen der MIDlet-Suite
`MIDlet-Name`	obligatorisch	Name der MIDlet-Suite
`MIDlet-Vendor`	obligatorisch	Hersteller der MIDlet-Suite
`MIDlet-Version`	obligatorisch	Versionsnummer im Format xx.yy[.zz]
`MIDlet-Data-Size`	optional	Minimal erforderlicher Speicherplatz in Byte
`MIDlet-Description`	optional	Beschreibung der MIDlet-Suite
`MIDlet-Info-URL`	optional	URL, unter der eine detaillierte Beschreibung der MIDlet-Suite abgerufen werden kann

Eine JAD-Datei könnte beispielsweise wie folgt aussehen:

```
MIDlet-Name: PromilleRechner
MIDlet-Version: 1.0.2
MIDlet-Vendor: Karsten Thomas
MIDlet-Description: Errechnet die Blutalkoholkonzentration
nach Alkoholgenuss
MIDlet-Info-URL: http://www.hemmerden.de/diplomarbeit/
MIDlet-Data-Size: 300
MIDlet-Jar-Size: 28933
MIDlet-Jar-URL: http://www.tk-expert.de/java/promille.jar
```

Auf diese Weise läßt sich schon vor dem Herunterladen einer MIDP-Anwendung feststellen, ob es sich um die gewünschte Version handelt und ob die Ressourcen des Endgerätes ausreichen, um die Applikation auszuführen.

Manifest

Detaillierte Informationen über die Anwendung werden in einem „Manifest" festgehalten. Dies ist eine dem Application Descriptor ähnliche Datei, welche jedoch zusätzliche Informationen enthält und sich *innerhalb* des JAR-Archives befindet. Diese Datei ist im Gegensatz zur JAD-Datei nicht optional.

Hier kann neben den in der JAR-Datei enthaltenen Attribute ein Icon definiert werden, das im PNG-Format im Archiv enthalten ist. Des weiteren müssen die Attribute `MicroEdition-Configuration` und `MicroEdition-Profile` vorliegen, welche die für die Anwendung erforderliche Konfiguration und die verwendeten Profile definieren.

Die Attribute der Manifest-Datei können von der Applikation über die Methode `javax.microedition.midlet.MIDlet.getAppProperty` ausgelesen werden.

MIDP für PalmOS

Eine erste wichtige Implementation des MIDP auf einem Endgerät ist das „Mobile Information Device Profile (MIDP) for Palm OS". Es stellt die CLDC und das MIDP für Geräte auf der Basis des PalmOS ab Version 3.5 bereit und wendet sich damit an eine große Zahl von Endgeräten, beispielsweise der Firmen Palm, Handspring und Sony.

Das Paket beinhaltet neben einer Dokumentation und einigen Beispielapplikationen ein Werkzeug, welches auf Desktop-PCs die Umwandlung von MIDlets in das Palm-Format PRC erlaubt.

Aus dem MIDP 1.0 werden insbesondere unterstützt:[58]

- Low-Level Grafik-API (Canvas)
- High-Level Grafik-API (LCDUI)
- Benutzerschnittstelle (Abstract Commands and Canvas Input)
- Datenbank-Zugriff über RMS
- Netzwerkanbindung via HTTP

[58] Sun Microsystems, Mobile Information Device Profile (MIDP) for Palm OS, http://java.sun.com/products/midp/palmOS.html (Stand: 30.08.2001)

Darüber hinaus ist der Zugriff auf die „Eigenschaften"-Einstellungen des Palm-Endgerätes möglich. Der Datenaustausch mit PCs via HotSync wird ebenso unterstützt, wie die Datenübertragung von MIDlets per Infrarot („Beaming").

Nicht integriert sind Palm-spezifische APIs, welche auf Betriebssystemebene auf die Palm-Hardware zugreifen, z.B. um einen direkten Zugriff auf das Palm-Datenbankformat zu erlauben. Auch der Zugriff auf die Palm-internen Datenbanken wie Adreßbuch und Kalender steht damit nicht zur Verfügung. Ebenfalls nicht möglich ist mit dieser MIDP-Implementation die Ansteuerung der Infrarot-Schnittstelle zur Datenübertragung innerhalb von MIDlets.
Solche Funktionen lassen sich über eigene APIs nachrüsten. Sie wurden jedoch absichtlich nicht in das Profil integriert, da dies den Standards des JCP widerspricht. Eine Integration dieser Klassen würde eine Portabilität der MIDP-Anwendungen auf andere Endgeräte wie z.B. die Blackberry-Pager von RIM verhindern. Werden nur die offiziellen Klassen des MID-Profils verwendet, sind alle Applikationen ebenfalls auf den Endgeräten anderer Hersteller lauffähig, sofern für diese eine MIDP-Implementation erhältlich ist.

Mit Hilfe der Implementation für PalmOS ist es recht einfach geworden, Java-Applikationen für Palm-Geräte zu entwickeln. Programme können am PC unter Microsoft Windows mittels eines Palm-Emulators[59] getestet und korrigiert werden (es sind umfangreiche Debugging-Funktionen integriert).
Es gibt mehrere Entwicklungsumgebungen, welche in Kapitel 6 näher beschrieben werden.

3.4.2. Mobile Information Device Profile Next Generation (MIDP_NG)

Seit April 2001 existiert ein weiterer Java Specification Request (JSR-118)[60], in welchem eine Expertengruppe mit über 50 Teilnehmern eine Weiterentwicklung des bestehenden MIDP erarbeiten wird.
Dieses Profil wird abwärtskompatibel zum MIDP 1.0 sein und sich weiterhin an mobile Endgeräte wie Mobiltelefone richten. Ziel ist es, das bestehende Profil weiterzuentwikkeln und zu ergänzen. Hierbei wird ein besonderes Augenmerk darauf gerichtet, die

[59] Palm Inc., PalmOS-Emulator, http://www.palmos.com/dev/tech/tools/emulator/

[60] Java Community Process, JSR-118, http://jcp.org/jsr/detail/118.jsp (Stand: 05.09.2001)

Größe der APIs und damit den Speicherbedarf des Profils nicht übermäßig ansteigen zu lassen. Die aus den Fehlern des ersten Profils MIDP 1.0 gezogenen Erkenntnisse sollen integriert werden und Funktionen implementiert werden, die für alle Endgeräte und Applikationen von Bedeutung sind. Eine besondere Beachtung soll dabei dem Mobile Commerce (m-Commerce) zuteil werden.

Insbesondere die folgenden Funktionsbereiche sollen in diesem Prozeß untersucht werden:

- das Domain Security Model, einschließlich Signatur von Applikationen und Verifizierung von Zertifikaten
- HTTPS und Secure Networking
- Netzwerkverbindungen via Sockets und Datagrammen
- OTA Provisioning
- Integration von Push-Technologien[61]
- Erweiterungen der Benutzerschnittstelle (user interface), vor allem Erweiterungen der Low-Level-APIs zur besseren Layout-Kontrolle
- ein kleiner, effizienter XML-Parser zur Unterstützung eines plattformunabhängigen Datenaustauschs
- API zur Sound-Unterstützung

Das MID-Protocol Next Generation wird im ersten Halbjahr 2002 im JCP diskutiert und im Laufe des Jahres 2002 veröffentlicht werden.

3.4.3. Foundation Profile

Das Foundation Profile (JSR-46)[62] existiert seit März 2001 in seiner endgültigen Fassung. Es handelt sich um ein Profil, welches nicht auf der CLDC aufbaut, sondern auf der Connected Device Configuration (CDC). Es dient als Basis für andere auf diese Konfiguration aufbauenden Profile. Bei Einsatz des Foundation Profiles werden neben einer Netzwerkanbindung - im Gegensatz zur CDC-Spezifikation, bei der geringere

[61] Vgl. Thomas, Karsten, „Push-Technologien“, http://www.hemmerden.de/mcommerce/ (Stand: 12.09.2001)

[62] Java Community Process, JSR-46, http://jcp.org/jsr/detail/46.jsp (Stand: 02.09.2001)

Werte ausreichen - mindestens 1 MB nichtflüchtiger und 512 KB flüchtiger Speicher benötigt.
Das Foundation Profile stellt eine Reihe von APIs der J2SE zur Verfügung, darunter allerdings keine Funktionalität einer Benutzerschnittstelle. Diese ist in vielen Endgeräten der CDC ohnehin nicht erforderlich und kann bei Bedarf durch auf das Foundation Profile aufbauende Profile hinzugefügt werden.

3.4.4. RMI Profile

Das RMI-Profil (Remote Method Invocation Profile) liegt seit Januar 2001 in einer fast endgültigen Fassung vor (JSR-66).[63] Es setzt die CDC sowie das Foundation Profile als Basis voraus. Mit Hilfe dieses Profils werden der J2ME die RMI-Klassen aus der J2SE[64] zur Verfügung gestellt, um eine Kompatibilität mit RMI-basierten Applikationen der J2SE zu erreichen.

3.4.5. Personal Profile

Wie bereits in Kapitel 2.3.1. erörtert, werden zum Teil bisher eigenständige Java-Technologien in die Java 2 Micro Edition integriert. Dies ist bei PersonalJava derzeit der Fall. Im JSR-62[65] wird das Personal Profile erarbeitet, welches auf die CDC und das Foundation Profile aufsetzt und der J2ME die Klassen von PersonalJava hinzufügt. Dies ist vor allem im Hinblick auf die Klassen des User Interface, der Benutzerschnittstelle, interessant, welche durch das Personal Profile nun zur Verfügung stehen.

Wenn auch die Mindestanforderungen an das Endgerät in hohem Maße steigen (es sind nun 2,5 MB ROM und mindestens 1 MB RAM erforderlich), ist dies ein wesentlicher Schritt für die J2ME, da mit Hilfe dieses Profils eine große Zahl der PersonalJava-Applikationen nun auch auf Endgeräten der J2ME implementiert werden kann.

---

[63] Java Community Process, JSR-66, http://jcp.org/jsr/detail/66.jsp (Stand: 02.09.2001)

[64] Sun Microsystems, “Java Remote Method Invocation Specification”, http://java.sun.com/products/jdk/1.2/docs/guide/rmi/spec/rmiTOC.doc.html (Stand: 03.09.2001)

[65] Java Community Process, JSR-62, http://jcp.org/jsr/detail/62.jsp (Stand: 02.09.2001)

3.4.6. PDA Profile

Das PDA-Profil befindet sich ebenfalls noch im Entwicklungsstadium.[66] Es wird auf der CLDC aufbauen und APIs für die Benutzerschnittstelle sowie Datenbankzugriffe auf PDAs bereitstellen.
Gemäß der CLDC-Spezifikation ist dieses Profil für batteriebetriebene mobile Endgeräte mit einer Speicherkapazität zwischen 512 KB und 16 MB gedacht. Voraussetzung werden ein Bildschirm mit einer Auflösung von 20.000 Pixeln, ein Zeigegerät und die Möglichkeit der Zeicheneingabe sein.
Ein Teil der Klassen wird sich auf die Unterstützung von einfachen Datenbankzugriffen beziehen, die wichtigste Komponente wird allerdings die Unterstützung der AWT-Klassen sein. Das Abstract Windowing Toolkit (AWT) dient der grafischen Steuerung der Benutzerschnittstelle unter J2SE und steht bisher nicht für mobile Endgeräte unter J2ME zur Verfügung.
Es gibt eine Entwicklung namens kAWT[67] (k steht für kilo, in Anlehnung an KVM, also eine Entwicklung für in ihren Ressourcen eingeschränkte mobile Endgeräte). Diese bietet (schon jetzt) einen Teil der AWT-Funktionalität im Bereich der J2ME. Die Programmierer des kAWT gehören der Expertengruppe des JSR-75 an, welche die Spezifikationen des PDA-Profils erarbeitet. Mit Hilfe dieses Profils sollen nun die kAWT-Klassen Teil der offiziellen Spezifikation werden und in das offizielle AWT integriert werden.
Damit wird für PDA-Geräte eine normierte Schnittstelle zur Ansteuerung des User Interface zur Verfügung stehen.

Sicherlich wird es nicht möglich sein, eine vollständige Funktionalität der AWT auf mobilen Endgeräten zu erreichen, jedoch werden in umgekehrter Richtung alle für das PDA-Profil geschriebenen Applikationen auch im Bereich der J2SE mit AWT lauffähig sein.

[66] Java Community Process, JSR-75, PDA Profile for the J2ME Platform, http://jcp.org/jsr/detail/75.jsp (Stand: 04.09.2001)

[67] kAWT, http://www.kawt.de/

3.5. Unabhängige APIs

Es gibt über die Profile hinaus eine Reihe von „unabhängigen Klassen“, welche auf die Konfigurationen der J2ME (CLDC und CDC) aufsetzen und diese um bestimmte Funktionen erweitern. Diese APIs müssen nicht notwendigerweise zu einem bestimmten Profil gehören.
Exemplarisch seien an dieser Stelle zwei vielversprechende Entwicklungen aufgeführt.

3.5.1. JavaPhone API

Das JavaPhone API[68] stellt eine Erweiterung für PersonalJava dar und bietet Java-Applikationen den Zugriff auf die internen Funktionen des verwendeten Mobiltelefons, beispielsweise die Telefoniefunktionen selbst oder das integrierte Telefonbuch.

Verschiedene Hersteller, darunter Nokia, Ericsson, Motorola, Psion und Symbian, unterstützen bereits das JavaPhone-API oder planen dies für zukünftige Endgeräte.

3.5.2. Wireless Telephony Communication APIs (WTCA)

Im Rahmen des JCR-120[69] befindet sich ein API in der Entwicklung, welches basierend auf den Konfigurationen der J2ME Unterstützung für den Zugriff auf die Ressourcen eines Mobiltelefons bieten soll.

Hiermit stehen den Applikationen die folgenden Telefoniefunktionen zur Verfügung:

- Short Message Service (SMS)
- Unstructured Supplementary Service Data (USSD)
- Cell Broadcast Service (CBS)

[68] Sun Microsystems, JavaPhone API, http://java.sun.com/products/javaphone/ (Stand: 30.06.2001)

[69] Java Community Process, JSR-120, http://jcp.org/jsr/detail/120.jsp (Stand: 06.09.2001)

3.5.3. Multimedia API

Der JSR-135 steht für die Entwicklung des „J2ME Multimedia API“.[70] Diese soll eine Lücke im Bereich der Multimedia- und Sound-Unterstützung schließen, da solche APIs bisher in der J2ME noch nicht implementiert sind.
Diese API wird sowohl mit der CLDC als auch der CDC arbeiten und (auf High-Level-Basis, um die Portabilität zu wahren) unter Berücksichtigung der eingeschränkten Speicherkapazität auf mobilen Endgeräten Java-Programmen den Zugriff auf die Multimedia-Funktionen des Endgerätes ermöglichen, sofern solche dort vorhanden sind.
Es ist geplant, diese Funktionen in Form einer Klasse `javax.microedition.media` zu integrieren. Die Spezifikationen sollen in Anlehnung an das oben genannte MIDP_NG erarbeitet werden und voraussichtlich noch im Jahr 2001 zur Verfügung stehen.

[70] Java Community Process, JSR-135, http://jcp.org/jsr/detail/135.jsp (Stand: 06.09.2001)

4. Virtual Machines

4.1. Grundlagen

Die Virtuelle Maschine (VM) ist die zentrale Komponente in der Java-Technologie. Hierbei handelt es sich um einen virtuellen Computer, welcher in der Regel selbst als Software realisiert ist und auf einer realen Hardware-Plattform mit einem Betriebssystem aufsetzt (vgl. Abbildung 7). Die Virtual Machine setzt die plattformunabhängigen Java-Befehle in den Maschinencode der jeweiligen Plattform um.

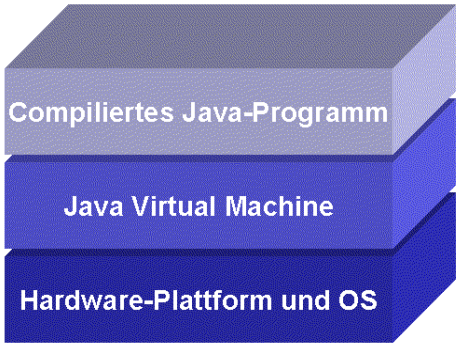

Abbildung 7: Einordnung der Virtual Machine

Der Java-Quelltext wird - wie bei anderen Programmiersprachen auch - zunächst kompiliert, dabei allerdings eben nicht direkt in ausführbaren Maschinencode für die verwendete Hardware umgewandelt, sondern zunächst in einen so genannten „Bytecode“, welcher den Maschinencode der Virtual Machine darstellt.
So ist es möglich, Java-Programme plattformunabhängig zu kompilieren und den so entstandenen Bytecode unverändert auf verschiedenen Plattformen einzusetzen. Voraussetzung ist lediglich, daß für die gewünschte Plattform eine Java Virtual Machine (JVM) erhältlich ist.

Ein Nachteil dieser Vorgehensweise ist, daß der Interpretationsvorgang des Bytecodes Geschwindigkeitsprobleme bereitet. Bei Programmiersprachen, bei welchen der Compiler direkt nativen Maschinencode des jeweiligen Prozessors erstellt, entfällt dieser Vorgang ganz. Daher werden Java-Programme zum Teil um den Faktor zehn langsamer ausgeführt als beispielsweise C++-Programme.
Es gibt verschiedene Ansätze, die Leistungsfähigkeit einer VM zu optimieren. Diese Technologien werden im folgenden Kapitel 4.2. vorgestellt.

Besonderheiten einer VM für mobile Endgeräte

Eine Virtual Machine sollte bestimmte Voraussetzungen erfüllen. Diese betreffen einerseits - aus Programmierersicht - die Unterstützung der verwendeten APIs, auf der anderen Seite - aus Anwendersicht - soll die VM eine möglichst stabile und vor allem zügige Ausführung der Applikation gewährleisten.
Auf mobilen Endgeräten sind diese Anforderungen schwerer zu erfüllen als auf Geräten, welche über ausreichend Prozessorleistung und Speicherplatz verfügen. Eine VM benötigt mehr Speicherplatz, wenn sie eine größere Zahl von Klassen/APIs unterstützt. Außerdem ist es wichtig, bei der Entwicklung einen besonderes Augenmerk auf die Performance der VM zu legen. Mobile Endgeräte sind meist mit weniger leistungsfähigen Prozessoren ausgestattet[71], welche die Programme wesentlich langsamer ausführen als CPUs auf Desktop-Systemen. So machen sich langsame und aufwendige Routinen einer VM auf mobilen Endgeräten um so stärker bemerkbar.
Dies zwingt besonders die Entwickler von solchen VMs, diese in hohem Maße zu optimieren.

Einige Virtual Machines werde ich in Kapitel 4.3. herausgreifen und näher beschreiben. Für die J2ME sind speziell an mobile Endgeräte angepaßte VMs verfügbar. Für das Grundverständnis, inwiefern sich Virtual Machines unterscheiden und - was gerade bei mobilen Endgeräten wichtig ist - welche Möglichkeiten der Optimierung sich bieten, ist das folgende Kapitel hilfreich, auch wenn es nicht speziell auf Virtual Machines der J2ME ausgerichtet ist.

[71] Vgl. Kapitel 1.2.

4.2. Aufbau und Optimierungsmöglichkeiten

4.2.1. Der Aufbau einer VM

Es ist nicht Gegenstand dieser Arbeit, die Interna einer Virtual Machine im Detail aufzugreifen, hierzu findet sich ausreichend Fachliteratur. Für das Verständnis der Unterschiede der Virtual Machines und der Optimierungsmöglichkeiten ist es ist jedoch notwendig, Grundkenntnisse über den Ablauf innerhalb der VM zu besitzen.

Oft wird die Bezeichnung „Virtual Machine“ lediglich als Synonym für einen virtuellen Prozessor benutzt. Zu einer vollen Laufzeitumgebung („Run Time Engine“) gehören jedoch, wie bei realen CPUs auch, einige andere Komponenten (vgl. Abbildung 8).

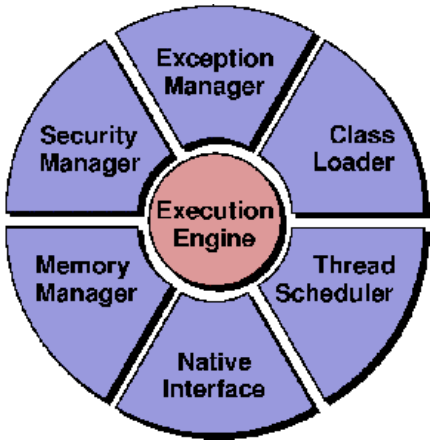

Abbildung 8: Komponenten der Java Virtual Machine
Quelle: Raner, Mirko, „Der Motor: Implementation der Java Virtual Machine“, Java Magazin, Heft 5/1999

Den Kern der Laufzeitumgebung bildet die Execution Engine, der virtuelle Prozessor. Dieser ist eingebettet in andere Komponenten, welche u.a. Schnittstellen zur umgebenden Hard- und Software bereitstellen.[72] Über die interne Struktur der Implementierung auf dem einzelnen Endgerät schreibt die „Java Virtual Machine Specification“[73]

[72] Vgl. Raner, Mirko, „Der Motor: Implementation der Java Virtual Machine“, Java Magazin, Heft 5/1999, http://www.javamagazin.de/ausgaben/1999/5/online_p.html (Stand: 24.07.2001)

[73] Lindholm, Tim, Yellin, Frank, "The Java Virtual Machine Specification, Second Edition”, Addison Wesley, Reading (MA), 1999, http://java.sun.com/docs/books/vmspec/2nd-edition/html/VMSpecTOC.doc.html (Stand: 27.08.2001)

nur weniges vor: es muß gewährleistet sein, daß die VM in der Lage ist, die .class-Dateien zu lesen und den darin enthaltenen Bytecode korrekt auszuführen. Diese Formulierung klingt zunächst sehr ungenau - doch dies bedeutet, daß die Entwickler einer VM einen recht großen Freiraum genießen bei der Entscheidung, welche Technologien zum Einsatz kommen.

4.2.2. Die Interpreter-Lösung

Bei den herkömmlichen JVMs handelt es sich um Interpreter. Diese greifen Stück für Stück den Bytecode auf und setzen ihn in die Maschinensprache des verwendeten Hardware-Prozessors um. Dabei wird jeweils nur ein Befehl („Opcode“) betrachtet und ausgeführt.
Diese Lösung erfordert naturgemäß einen sehr hohen Zeitaufwand und zeichnet sich durch einen hohen Speicherbedarf aus. Allerdings ist ein Interpreter vergleichsweise einfach zu implementieren und bereitet meist keine Schwierigkeiten bei der Portierung auf andere Endgeräte.

4.2.3. Just-In-Time-Compiler

Während bei Compilern anderer Programmiersprachen der Quellcode direkt in den Maschinencode des Prozessors übersetzt wird, erfolgt bei einer Java VM der Umweg über den Bytecode. Doch die Technologie anderer Programmiersprachen läßt sich auch bei Java aufgreifen.
Just-In-Time-Compiler (JIT, nicht zu verwechseln mit dem Compiler, welcher aus den .java-Dateien die .class-Dateien mit Bytecode erzeugt) basieren auf dem Interpreter-Konzept und erweitern dieses. HotSpot ist beispielsweise ein solcher JIT-Compiler. Diese funktionieren zunächst wie ein herkömmlicher Java-Interpreter. Allerdings werden während der Ausführung eines Java-Programmes bestimmte Bytecode-Sequenzen oder auch ganze Methoden direkt in Maschinencode übersetzt, welcher für spätere Ausführungen erhalten bleibt. Bei sich wiederholenden Sequenzen wie beispielsweise Schleifen liegt dann bereits der Maschinencode vor, so daß bei allen weiteren Ausführungen desselben Codes keine Übersetzung mehr erfolgen muß, sondern der Code direkt auf dem Hardware-Prozessor ausgeführt werden kann.
Wie effektiv diese Technologie tatsächlich ist, hängt davon ab, wie gut der JIT-Compiler in der Lage ist, den Bytecode auf sich wiederholende Befehlsfolgen zu

durchsuchen. Er muß während der Laufzeit bewerten können, ob eine direkte Übersetzung lohnt oder nicht. Da diese Bewertung wiederum wertvolle Rechenzeit kostet, läßt sich diese Aufgabe oft nicht optimal bewältigen. Dennoch führt die Verwendung eines JIT-Compilers durchschnittlich zu einer Geschwindigkeitssteigerung um den Faktor 2-5.[74]

Eine weitere Steigerung der Effektivität könnte erreicht werden, wenn der übersetzte Maschinencode nicht nur während der Laufzeit eines Programms im Speicher verbleiben würde, sondern auch darüber hinaus konserviert werden könnte. So stünde er bei der nächsten Programmausführung wieder zur Verfügung. Solche Caching-Technologien, die bei anderen Softwareprodukten regelmäßig vorzufinden sind (beispielsweise bei Internet-Browsern), wurden bisher nicht realisiert. Auf mobilen Endgeräten ist eine solche Idee meist von vornherein ausgeschlossen, da der verfügbare Speicherplatz oft schon nicht ausreicht, um die Daten der Laufzeitumgebung vollständig unterzubringen.

JIT-Compiler sind relativ aufwendig zu implementieren und sind schlecht zu portieren, da sie maschinennah realisiert werden müssen. Aus diesen Gründen sind sie meist im Desktop-Bereich und nur selten in mobilen Endgeräten zu finden.

4.2.4. Recompiler

Gegenüber den Just-In-Time-Compilern gehen die sogenannten Recompiler[75] noch einen Schritt weiter. Sie übersetzen vor Ausführung des Programms den kompletten Bytecode der .class-Datei, so daß er bei der tatsächlichen Ausführung vollständig in Maschinensprache vorliegt. Der Interpreter fällt hier völlig weg.

Durch dieses Vorgehen ist eine Geschwindigkeitssteigerung gegenüber einem Bytecode-Interpreter um den Faktor 3-10 möglich, in Einzelfällen läßt sich die Geschwindigkeit bei einzelnen Methodenaufrufen auch bis zum Faktor 100 steigern.

Ein weiterer Vorteil gegenüber den JIT-Compilern ist der geringere Speicherbedarf während der Ausführung, da zwar der vollständige Maschinencode vorliegen muß, aber kein Interpreter und nur kleine Teile des Compilers zur Laufzeit im Speicher benötigt werden. Ein gravierender Nachteil der Recompiler ist allerdings, daß beim Aufruf des Programms zunächst relativ viel Zeit für die Übersetzung benötigt wird, bevor der

[74] Vgl. Raner, Mirko, „Der Motor: Implementation der Java Virtual Machine“

[75] auch „native compiler“, „one-off compiler”, „monolithic compiler” oder „way-ahead-of-time compiler”

tatsächliche Programmstart erfolgen kann. Des weiteren ist es schwierig, Recompiler auf andere Hardwareplattformen zu portieren.

4.2.5. Vom Quellcode zum Prozessor

Zusammenfassend beschreibt Abbildung 9 den Weg, den ein Programm von der Entwicklung in der Java-Programmiersprache bis zur Ausführung im Prozessor beschreitet. Allen drei soeben beschriebenen Technologien ist gemeinsam, daß zunächst - einmalig - aus dem Java-Quelltext (der Datei „*.java“) ein Bytecode erstellt wird („*.class“). Dieser wird nun zu jeder Programmausführung von der Virtual Machine eingelesen und - je nach Compiler-Technologie - auf unterschiedliche Weise in Maschinencode überführt, welcher auf dem Hardware-Prozessor ausgeführt werden kann.

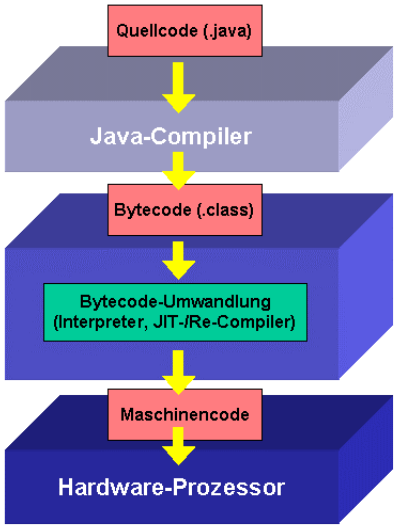

Abbildung 9: Der Weg vom Quellcode zum Prozessor

4.2.6. Java in Silizium

Wenn bisher immer die Rede ist von einer „Virtuellen Maschine“, welche den Java-Code ausführt - ist es dann nicht naheliegend, eine solche als reale Maschine zu bauen? Dieser Frage geht die Industrie bereits seit einiger Zeit nach.
So wurde die Virtuelle Maschine auch schon vollständig in Silizium realisiert. Bei dieser Maschine werden die häufigsten Befehle direkt in der Hardware ausgeführt, während komplexere Befehle als Maschinencode bereitgestellt oder per Software simuliert werden können.
Die Ausführungsgeschwindigkeit liegt bei weitem über der von Virtuellen (Software-)Maschinen, außerdem ist der Speicherbedarf („Memory Footprint“) während der Programmausführung sehr gering.
Nachteile weisen diese Hardware-Prozessoren bei der Flexibilität auf. Es ist ohnehin schon nicht leicht, eine solche Maschine zu implementieren, welche flexibel auf die verschiedensten Situationen bei der Ausführung von Java-Programmen reagieren kann. Doch ein weiterer sehr bedeutender Aspekt ist, daß bei der reinen Implementation der Virtual Machine als Hardware zunächst keine Routinen bereitstehen, über die eine Kommunikation mit der restlichen Hardware erfolgen könnte.[76]

Sun, Zucotto, ARM und einige andere Firmen haben reale Java-Prozessoren entwikkelt.[77]

Sun Microelectronics
Sun bietet mit „picoJava I und II“[78] zwei Prozessoren an, welche Java-Bytecode direkt ausführen können. picoJava II ist zusätzlich in der Lage, mit C oder C++ generierten plattformspezifischen Code auszuführen.
Die Prozessoren eignen sich insbesondere für digitale Set-Top-Boxen, Internet-Screen-Telefone und Kommunikationsgeräte für die Automobilindustrie. Im Vordergrund steht bei der Entwicklung weniger die absolut höchste Ausführungsgeschwindigkeit, diese ist im Vergleich zu Virtuellen Maschinen ohnehin schon um ein Vielfaches

[76] Vgl. Raner, Mirko, „Der Motor: Implementation der Java Virtual Machine“

[77] Vgl. Glahn, Kay, „VM Anywhere - Virtuelle Java Maschinen auf fast jeder Plattform“, Java Magazin, Heft 5/2001, S. 40

[78] Sun Microelectronics, picoJava, http://www.sun.com/microelectronics/picoJava/

höher. Hier ist vor allem wichtig, daß die Geräte zuverlässig funktionieren und dabei so preiswert sind, daß sie sich für den Einbau in Massenprodukte eignen.
Sun vergibt im Rahmen des Sun Community Source Licensing Vertrages (SCSL)[79] Lizenzen dieser Technologie. Die Firmen NEC, Fujitsu, IBM, Rockwell Collins und neuerdings auch LG Semicon haben bereits Lizenzen erworben und entwickeln nun eigene Prozessoren auf der Basis dieser Technologie.

Zucotto Wireless

Die Firma Zucotto Wireless bietet neben Softwaretools für Java auch einen Hardware-Java-Prozessor an. Xpresso soll ein speziell für J2ME geeigneter, preiswerter Java-Prozessor mit hoher Geschwindigkeit und wenig Speicherbedarf sein, der den Java-Bytecode direkt auf der Hardware ausführen kann.
Mit dem „XPRESSOboard HDK“[80] steht sogar ein Hardware Development Kit bereit, mit dem mit Hilfe eines Motherboards und eines „Daughterboards“ mit Flash-Speicher Java-Prozessoren entwickelt werden können.

ARM

Ein weiterer Java-Prozessor stammt von ARM, dem Hersteller des in Organizern wie z.B. dem Compaq iPAQ verwendeten StrongARM-Prozessors. Der Java-Prozessor basiert auf der „Jazelle“-Technologie.[81] ARM bietet bereits 32bit-RISC-Prozessoren an und hat in diese neben zwei bisherigen Befehlssätzen mit dem Java-Bytecode nun einen weiteren integriert.
Auch dieser Prozessor soll sich vor allem für Embedded-Geräte, Set-Top-Boxen und PDAs eignen - nicht zuletzt auch aus dem Grund, daß er nicht nur den Java-Bytecode interpretieren kann, sondern daneben auch über zwei weitere Befehlssätze verfügt.

weitere Entwicklungen

Darüber hinaus sind ähnliche Entwicklungen von weiteren Firmen auf dem Markt, beispielsweise von aJile Systems[82], JStamp/Systronix[83] und PTSC[84].

---

[79] Sun Microelectronics, Sun Community Source Licensing, http://www.sun.com/microelectronics/communitysource/ (Stand: 07.09.2001)

[80] Zucotto Wireless, XPRESSOboard HDK, http://www.zucotto.com/products/xpressoboard_index.html (Stand: 07.09.2001)

[81] ARM, ARM Jazelle Technology, http://www.arm.com/armtech/jazelle?OpenDocument (Stand: 07.09.2001)

[82] aJile Systems, http://www.ajile.com/products.htm

4.2.7. Verteiltes Java

Es gibt dank Remote Method Invocation (RMI) und CORBA auch heute schon Möglichkeiten, Java-Programme in einer verteilten Umgebung ablaufen zu lassen. Doch hierzu sind in der Regel Kenntnisse über das Netzwerk und die Infrastruktur notwendig.[85]
Dies ist mit der „Distributed Java VM“[86] anders. Diese Virtual Machine erlaubt es, ohne genaue Kenntnisse der Infrastruktur Java-Programme auf verschiedene Rechner verteilt auszuführen. Hierbei kann es sich sogar um Rechner unterschiedlicher Plattform handeln, beispielsweise zwei Windows- und drei Linux-Rechner. Dank der verteilten JVM wirkt die Umgebung auf die Java-Anwendung homogen wie ein einziger Rechner. Dies ist im Hinblick auf die große Rechenleistung sinnvoll, die sich durch ein solches Zusammenschalten mehrerer Rechner zu einem Cluster erreichen läßt.

Leider ist diese VM seit 1998 nicht mehr weiterentwickelt worden. Allerdings steht ihr Quellcode frei zur Verfügung, so daß eventuell später an anderer Stelle mit einer Weiterentwicklung zu rechnen ist.

4.3. Verschiedene Virtual Machines für mobile Endgeräte

Während in der Java 2 Standard Edition die Classic VM von Sun eine geeignete Virtual Machine darstellt sowie zahlreiche Optimierungen wie z.B. die HotSpot-VM existieren, basiert die J2ME entweder ebenfalls auf der Classic VM (bei Verwendung der CDC), oder aber auf einer für mobile Endgeräte optimierten Maschine, beispielsweise der KVM (bei Verwendung der CLDC). Der Name „KVM“ steht für „Kuaui VM“[87], einem Codenamen für eine frühe Entwicklungsversion. Heute wird das „K“ im Namen oft als

[83] JStamp, http://jstamp.systronix.com/info.htm

[84] PTSC, http://www.ptsc.com/

[85] Vgl. Glahn, Kay, „VM Anywhere - Virtuelle Java Maschinen auf fast jeder Plattform“, Java Magazin, Heft 5/2001, S. 38

[86] Clark, Rob, „Distributed Java VM”, http://www.cs.hmc.edu/~rclark/dj/index.html (Stand: 07.09.2001)

[87] Vgl. Giguère, Eric, „Java 2 Micro Edition“, S. 78

Abkürzung für „kilo“ betrachtet, da die Endgeräte, für welche die KVM entwickelt ist, oft nur über einige Kilobyte Speicher verfügen.
Auch im Bereich der mobilen Endgeräte gibt es zahlreiche Virtuelle Maschinen, die sich in ihrer Performance, der Hardware-Plattform und den unterstützten Java-Klassen unterscheiden. Die Grenze zwischen VMs der J2SE und der J2ME sind dabei fließend. Es gibt Virtuelle Maschinen, die eigentlich für die J2SE konzipiert sind, aufgrund ihres geringen Ressourcenbedarfs („foot print“) jedoch auch auf manchem Organizer im PDA-Format lauffähig sind. Auch die Classic VM kann im Rahmen der J2ME eingesetzt werden, sofern das Endgerät über ausreichend Speicherplatz verfügt.

Eine vollständige Vorstellung der verfügbaren VMs würde die Möglichkeiten im Rahmen dieser Diplomarbeit bei weitem übersteigen. Daher möchte ich einige wenige Virtuelle Maschinen herausgreifen und auf diese etwas näher eingehen. Dieses Kapitel soll den Überblick des vorigen Kapitels über die verfügbaren VM-Technologien ergänzen und reale Produkte zu den beschriebenen Technologien vorstellen.
Falls tiefergehende Informationen benötigt werden, beispielsweise eine genaue Spezifikation oder eine vollständige Aufstellung der unterstützten APIs, so sind diese leicht unter den angegebenen Quellen im WWW aktuell zu beschaffen. Ebenso ist dort die jeweils neueste Version der besprochenen Software verfügbar.

4.3.1. Übersicht

Name	Plattform	Eigenschaften
KVM	viele Portierungen	Referenz-VM für J2ME, CLDC, MIDP (Palm)
PersonalJava	Windows CE 2.11	bisher nicht für Windows CE 3.x in Zukunft als Profil der J2ME CDC
IBM J9	PalmOS, Windows CE, Win32, Solaris und andere OS	Einsatz in Embedded-Systemen, CDC, CLDC, MIDP
JBed Micro Edition CLDC	PalmOS	vollständig J2ME-kompatibel, CLDC, MIDP Recompiler
xKVM / Color KVM	PalmOS	bietet u.a. Farbunterstützung für KVM CLDC, kAWT
CrEme	Windows CE	PersonalJava, JDK 1.1.8
SavaJe XE OS	StrongARM-Geräte (z.B. Compaq iPAQ)	J2SE (!), RMI, CORBA, Jini Betriebssystem, welches J2SE unterstützt. Ersetzt das OS auf StrongARM-Geräten.

Kaffe OpenVM	viele Portierungen	erweitertes PersonalJava auf vielen Plattformen
Kada VM	PalmOS, Windows CE, PocketPC	PersonalJava, CLDC, CDC
HP Chai VM	Windows CE, Red Hat Linux, Win NT/x86	für Embedded-Systeme, JDK 1.1.8, CLDC, demnächst MIDP
Jeode	Windows CE 2.12 und 3.0 u.a.	PersonalJava, EmbeddedJava, JDK 1.1.8 nicht frei verfügbar (außer für Compaq iPAQ)
microJBlend Kit for CE	bisher PalmOS, demnächst auch Windows CE	CLDC, MIDP
Symbian EPOC JVM	EPOC 5 (Psion-Geräte)	PersonalJava, JDK 1.1.4, AWT mit EPOC-"look-and-feel", Unterstützung für CLDC und MIDP geplant.
Microsoft VM	Windows	JDK 1.1

Exemplarisch seien die genannten Virtuellen Maschinen im folgenden detaillierter vorgestellt.[88]

4.3.2. KVM

Im Bereich der Connected Limited Device Configuration (CLDC) werden die Eigenschaften der verwendeten Virtual Machine durch die Vorgaben der CLDC festgelegt. Welche VM im Einzelfall zum Einsatz kommt, ist dem Gerätehersteller oder sogar dem Endbenutzer überlassen - solange sie die Anforderungen der CLDC erfüllt.
Die KVM ist eine Virtual Machine, welche diesen Spezifikationen entspricht. Sie ist zugleich die Referenz von Sun und daher weit verbreitet - doch an ihrer Stelle könnte jede andere VM mit gleichen Eigenschaften eingesetzt werden.

Die Hauptmerkmale der KVM:
Die KVM besitzt keine dynamischen Optimierungstechniken, wie sie in Kapitel 4.2. vorgestellt wurden. Die meisten mobilen Endgeräte weisen hierfür keine ausreichenden Ressourcen auf.

[88] weitere Informationen zu verfügbaren Virtual Machines finden sich hier (Stand: 08.09.2001):
JavaMobiles.com, „List of JVM for PDA", http://www.javamobiles.com/jvm.html#home
Friedman, David K., Wheeler, David A., „Java Implementations",
http://www.dwheeler.com/java-imp.html
Schmidt, Marc, "List of Java Compilers and Virtual Machines",
http://www.geocities.com/marcoschmidt.geo/java.html

Die KVM ist aus dem „Spotless“-System der Fa. Sun hervorgegangen,[89] ist in C implementiert und aufgrund ihrer guten Dokumentation sowie der Tatsache, daß sie dem Sun Community Source Licensing (SCSL) unterliegt, leicht auf verschiedene Plattformen zu portieren (neben Portierungen auf Windows, Solaris und PalmOS [ab Version 3.01] ist die KVM bisher auf über 25 Plattformen portiert worden).
Sie benötigt 40-80 KB Speicher (auf Palm-PDAs und Win32-Systemen: 60 KB) und erreicht zwischen 30 und 80 % der Geschwindigkeit eines JDK-1.1-Systems (ohne JIT-Compiler).[90]
Die KVM ist den Restriktionen der CLDC unterworfen, so z.B. werden keine Fließkomma-Operationen und nur eine limitierte Zahl von J2SE-Libraries unterstützt (vgl. Kapitel 3.3.1.). Auch ist aufgrund der geringen Ressourcen der Endgeräte das Sicherheitsmodell der J2SE nicht realisierbar.

Weitere Eigenschaften in Kurzform:[91]

- Unterstützung fast aller Java-Sprachelemente:
  - Byte, boolean, char, short, int, long
  - Unicode
  - Strings und StringBuffer
  - Klassen und Schnittstellen
  - Ausnahmen:
  - Fäden (Threads)
  - Garbage Collection
  - ByteCode-Verification und dynamisches Laden von Klassen
- Das vereinfachte Sicherheitsmodell der KVM:
  - kein Überladen oder Ersetzen der Systemklassen
    (geschlossener Satz an Funktionalität, Namensraum `java.*`, `javax.microedition.*`)

[89] Ziel dieses Forschungsprojektes war es, eine vollständige Virtual Machine für Palm-PDAs zu entwickeln. Vgl. http://www.sun.com/research/spotless/ (Stand: 26.07.2001)

[90] Vgl. Yellin, Frank, “Inside the K Virtual Machine”, Sun Microsystems, JavaOne-Conference, http://servlet.java.sun.com/javaone/javaone2000/pdfs/TS-1507.pdf (Stand: 05.09.2001)

[91] Vgl. Merck, Martin, Sun Microsystems GmbH, „Java 2 Micro Edition - Die Basis für Wireless Geräte“ auf der Sun-Konferenz „The Future is Wireless“ im Mai 2001 in Neuss, http://www.sun.de/Downloads/Praesentationen/Wireless/Vortraege/1/Merck.pdf (Stand: 01.08.2001)

  - keine benutzerdefinierbaren Klassenlader (Reihenfolge der Klassensuche)
  - kein Nachladen von Nicht-Java-Methoden (keine öffentliche “native” Schnittstelle)

- Einschränkungen der KVM gegenüber der Classic Virtual Machine (CVM):
  - keine Gleitkommazahlen (kein `float`, `double`)
  - keine Finalisierung (`Object.finalize()` nicht vorhanden)
  - keine Reflection (keine Unterstützung für RMI, Objektserialisierung, Profiling JVMPI oder Debugging JVMDI)
  - eingeschränkte Fehlerbehandlung (es werden nur `java.lang.VirtualMachineError` und `java.lang.OutOfMemoryError` unterstützt)

  Einschränkungen durch das vereinfachte Sicherheitsmodell:
  - keine Unterstüzung des JNI (“native”-Methoden dürfen nur vom Hersteller implementiert weden)
  - keine benutzerdefinierten Klassenlader
  - nur einfache Fäden (keine ThreadGroups oder daemon threads)
  - keine schwachen Referenzen

- Funktionsweise des Verifizierers:
  - Zwei-Phasen-Verifikation
    (Off-Device Pre-Verification, In-Device Verification)

4.3.3. PersonalJava

Das „PersonalJava Application Environment" (PJAE) liegt derzeit in der Version 1.2a vor.[92] Die Virtual Machine basiert auf dem JDK 1.1.8, es wurden gegenüber diesem jedoch einige Zusätze aus dem JDK 1.2 der J2SE eingearbeitet (vgl. Kapitel 2.3.1.).
PersonalJava erfordert ein System mit Windows CE in der Version 2.11. Mit Windows CE 2.0 ist PersonalJava nicht lauffähig, und für die aktuelle Version 3.0 liegt ebenfalls keine Unterstützung vor. Als Prozessor ist im Endgerät ein MIPS- oder SH3-Prozessor erforderlich sowie mindestens 16 MB RAM, wovon etwa 6 MB für die PersonalJava-Laufzeitumgebung benötigt werden.

Wie bereits oben beschrieben, wird das bisher eigenständige Produkt „PersonalJava" in die Form eines auf der CDC basierenden Profils der J2ME überführt werden. Damit wird es bei geeigneter Hardwareausstattung auf einer wesentlich größeren Zahl von Endgeräten lauffähig sein, als es bisher der Fall ist.

4.3.4. IBM J9 VM

Die Virtuelle Maschine „J9" von IBM ist Bestandteil der „Visual Age Micro Edition".[93] Diese stellt eine Entwicklungsumgebung auf Windows- und Linux-Plattformen bereit.
Die VM ist für Endgeräte mit eingeschränkten Ressourcen konzipiert, insbesondere Embedded Systems. Sie unterstützt die Betriebssysteme PalmOS, Windows CE, Hard Hat Linux und QNX Neutrino auf einer Reihe von Prozessoren (darunter Intel x86, Motorola 68xxx, ARM und StrongARM, MIPS, PowerPC und Hitachi SuperH SH-3 und SH-4).

Die unterstützten APIs basieren auf dem JDK 1.2.2, sind jedoch für den Einsatz auf mobilen Endgeräten optimiert. Es bleibt dem Entwickler überlassen, welche dieser Klassen implementiert werden. Es werden je nach Plattform CDC, CLDC und MIDP unterstützt. Mit der Visual Age Micro Edition Personal Configuration stehen außerdem Klassenbibliotheken u.a. für AWT, RMI und Java Beans für Windows bereit.

---

[92] Vgl. Sun Microsystems, "PersonalJava Application Environment", http://java.sun.com/products/personaljava/ (Stand: 20.08.2001)

[93] IBM, J9 VM, http://www.embedded.oti.com/learn/vaesvm.html (Stand: 20.08.2001)

Die PalmOS-Laufzeitumgebung enthält spezielle APIs für den Einsatz auf PalmOS-Geräten.

Die J9 VM ist in der Lage, neben Standard-Class-Dateien und JAR-Dateien sogenannte JXE-Dateien zu lesen. Hierbei handelt es sich um ein optimiertes Format für Klassen und Ressourcen, mit dem diese im ROM des Gerätes abgelegt werden können und somit keinen wertvollen RAM-Speicher belegen.
Des weiteren unterstützt J9 das Java Native Interface (JNI), welches der direkten Kommunikation mit Gerätetreibern und gerätespezifischen Betriebssystemfunktionen dient.

4.3.5. JBed Micro Edition CLDC

Die Jbed-Produktlinie der Firma Esmertec beinhaltet eine vollständige Sammlung von Java-Werkzeugen. Neben einer IDE-Entwicklungsumgebung ist mit Jbed RTOS eine Kombination aus Java Virtual Machine und Real-Time Operating System (RTOS) für eine Vielzahl von Embedded-Plattformen verfügbar.
Für mobile Endgeräte im Sinne dieser Arbeit (PDAs und Mobiltelefone) ist die Jbed Micro Edition CLDC[94] vorgesehen. Hierbei handelt es sich um eine optimierte Java Virtual Machine, welche die J2ME CLDC vollständig ersetzt. Im Gegensatz zu anderen JVMs interpretiert die Jbed VM den Bytecode nicht, sondern compiliert diesen in Maschinencode des Endgerätes und erlaubt auf diese Weise eine Steigerung der Ausführungsgeschwindigkeit bis zum Faktor 50.

Die Jbed Micro Edition CLDC kann direkt auf der Hardware des Endgerätes ausgeführt werden, läßt sich aber auch auf einem bestehenden Betriebssystem wie dem PalmOS verwenden. Bisher steht lediglich eine Implementation für Palm-Geräte (und kompatible) zur Verfügung, in Zukunft sollen weitere Plattformen unterstützt werden. Auf allen Palm-Organizern mit 8 MB sowie auf neueren Geräten mit 2 bzw. 4 MB ist Jbed lauffähig, bei Verwendung des Connection Framework wird ein PalmOS ab Version 3.3 vorausgesetzt.
Die entwickelten Applikationen können entweder wie üblich als Bytecode auf den Palm-Organizer aufgespielt werden, oder aber sie können mittels der Jbed-Software vorcompiliert werden und in Form einer .prc-Datei übertragen werden. Während im

[94] Esmertec, Inc., Jbed Micro Edition CLDC, http://www.esmertec.com/p_jbed_cldc_long.html (Stand: 07.09.2001)

ersten Fall auf dem Endgerät eine JVM installiert sein muß, ist dies bei der zweiten Variante nicht erforderlich - hier liegt nativer Code vor, welcher ohne JVM ausgeführt werden kann. Hierdurch sind hohe Geschwindigkeitssteigerungen möglich, außerdem werden so die Anforderungen an die Speicherkapazität minimiert.

Aufgrund der vollständigen Kompatibilität mit Suns Java 2 Micro Edition und der CLD-Konfiguration stehen dem Entwickler alle Klassen der CLDC zur Verfügung. Darüber hinaus sind in Form des „Jbed Profile for MID“ die Klassen des MIDP erhältlich (Esmertec ist Mitglied des Java Community Process). Auf Palm-Geräten ist außerdem auch der Zugriff auf die Palm-Datenbanken und Peripheriegeräte möglich.

Insgesamt bietet Esmertec mit der Jbed Micro Edition CLDC ein sehr vielversprechendes Paket an. Die IDE-Entwicklungsumgebung vervollständigt das positive Gesamtbild, kann aber auf Wunsch auch durch jede andere Java-Entwicklungssoftware ersetzt werden (vgl. Kapitel 6). Mit der Jbed Virtual Machine steht eine schnelle VM zur Verfügung, die sich im Gegensatz zu einigen anderen VMs aufgrund der hohen Kompatibilität mit der KVM zum Einsatz auf Palm-Geräten geradezu anbietet. Es bleibt abzuwarten, ob Jbed in Zukunft auch für andere Plattformen zur Verfügung stehen wird.

4.3.6. xKVM

xKVM[95] steht für „eXtendend KVM“ und ist eine Entwicklung der Programmierer der kAWT-Erweiterung, die in Kapitel 3.4.6. vorgestellt wurde.
Bei der xKVM (vormals „Color KVM“) handelt es sich um eine um einige Komponenten erweiterte KVM. Diese ist zur Zeit in zwei Versionen erhältlich: eine Version beinhaltet „nur“ die unten genannten Erweiterungen, in der anderen Version sind gleichzeitig die oben erwähnten kAWT-Klassen integriert.

Erweitert wurde die KVM um die folgenden Komponenten:

- Unterstützung von Farbe und Graustufen auf Palm-Geräten (PalmOS 3.5)
- 16-Bit Farbe für Handspring Prism-Geräte
- unterstützt das JogDial-Rad des Sony Clié
- auf HandEra 330-Geräten: 240 x 320 QVGA hochauflösende Grafik, Unterstützung für Minimierung und Maximierung des Silkscreens

[95] Kroll & Haustein GbR, xKVM, http://www.kAWT.de (Stand: 07.09.2001)

Es handelt sich hier sicherlich um sinnvolle Erweiterungen, vor allem bei den oben genannten Klassen der kAWT. Die Programmierer betonen jedoch, daß es sich bei der xKVM um eine VM für Forschungszwecke handelt und das Projekt eine Studie darstellt, inwiefern es möglich ist, die KVM zu erweitern und eigene Klassen zu integrieren.

4.3.7. CrEme

CrEme,[96] eine Entwicklung der israelischen Firma NSIcom, ist eine optimierte Virtual Machine für Geräte auf Basis von Windows CE. Der aktuellen Version 3.1 liegt die PersonalJava-VM der Firma Sun (JDK 1.1.8) zugrunde, welche an die Bedürfnisse mobiler Endgeräte angepaßt wurde.

Neben integrierten Optimierungen bezüglich Geschwindigkeit, Kompatibilität und Speicherverbrauch dürfte vor allem interessant sein, daß mit CrEme eine volle (PersonalJava-kompatible) Java-Unterstützung für alle Windows CE-Versionen bereit steht, während PersonalJava lediglich auf Windows CE 2.11 lauffähig ist.

4.3.8. SavaJe XE OS

Bei SavaJe XE[97] handelt es sich nicht um eine Virtual Machine, die auf einem Betriebssystem (OS) aufsetzt, sondern um ein Java-Betriebssystem, welches das bestehende OS vollständig ersetzt.
Erforderlich sind mindestens 12 MB nichtflüchtiger Speicher und 32 MB RAM, lauffähig ist SavaJe XE auf Geräten mit StrongARM-Prozessoren ab 190 MHz - diese Anforderungen erfüllen derzeit beispielsweise Compaqs iPAQ und das Psion netBook.
SavaJe XE basiert auf der Java 2 Standard Edition (J2SE), JDK 1.3, und bietet damit eine vollständige Java-Unterstützung, einschließlich PersonalJava und J2ME (bei Verwendung der AWT-Klassen), Swing, AWT, JNI, RMI und CORBA.

---

[96] NSIcom Ltd., CrEme, http://www.nsicom.com/products/creme.asp (Stand: 04.09.2001)

[97] SavaJe Technologies, SavaJe XE, http://www.savaje.com/products/savajexe.html (Stand 10.09.2001)

Da die Software direkt auf die Hardware zugreift und kein anderes Betriebssystem zwischengeschaltet ist, verspricht der Hersteller eine optimale Leistung, welche weit über die Leistung herkömmlicher Virtual Machines hinausgeht.

Derzeit befindet sich die SavaJe XE in einem späten Beta-Stadium, in dem noch nicht alle Funktionen verfügbar sind. Mit Erscheinen des ersten Release im September 2001 werden auch fehlende Funktionen wie Modem- oder Multimedia-Unterstützung integriert sein. Die Software wird nach einer Evaluierungsphase von 30 Tagen für ca. 100 US-Dollar erhältlich sein, evtl. wird es von Compaq eine iPAQ-Version geben, auf welcher bereits SavaJe XE vorinstalliert ist.

Die Technologie verspricht die Ausführung *aller* Klassen der J2SE, nicht nur die eingeschränkten Versionen von PersonalJava oder J2ME - im Grunde genommen eine interessante Leistung auf einem mobilen Endgerät. Allerdings wird sich erst in der Praxis herausstellen, ob es gelingt, große Marktanteile zu gewinnen. Der Endnutzer muß sich für eines der Betriebssysteme entscheiden, er kann SavaJe XE nicht parallel zu Windows CE benutzen, welches ursprünglich auf dem Endgerät installiert war. Somit gewinnt er zwar einen großen Spielraum beim Einsatz von Java-Software, doch eine breite Palette an Software für Windows CE steht dann nicht mehr zur Verfügung. Zwar ist es theoretisch möglich, mittels verschiedener Flash-Speicherkarten zwischen den Betriebssystemen zu wechseln, doch dies dürfte in der Praxis eher umständlich sein.

Wenn sich Java in Zukunft stärker im mobilen Sektor durchsetzt und damit auch das Software-Angebot steigt, hat ein Betriebssystem wie SavaJe XE sehr gute Chancen, andere Lösungen zu verdrängen. Aufgrund der marktbeherrschenden Stellung von PalmOS und Windows CE verbunden mit dem großen Softwareangebot für diese Plattformen ist es jedoch fraglich, ob dies gelingen wird.

4.3.9. Kaffe OpenVM

Die Kaffe Virtual Machine[98] ist - so der Hersteller Transvirtual Technologies - die erste „überall lauffähige“ Java-Implementation. Sie unterstützt sowohl die Java-

[98] Transvirtual Technologies Inc., Kaffe OpenVM, http://www.kaffe.org/
Vgl. auch http://www.transvirtual.com/datasheet.pdf (Stand 08.09.2001)

Spezifikationen von Sun als auch die von Microsoft, und dies auf annähernd allen verfügbaren Plattformen.
Dem gesetzten Ziel kommt Kaffe tatsächlich recht nahe. In der Liste der unterstützten Betriebssysteme befinden sich neben den Desktop-Betriebssystemen Windows 98/NT, Solaris, SunOS, Linux und Unix-Derivaten auch Embedded-Plattformen wie Windows CE, Win32, Linux, DOS und einige andere. Hardwareseitig werden u.a. x86-, 68xxx-, ARM-, PowerPC- und MIPS-Prozessoren unterstützt.
Kaffe basiert auf PersonalJava 3.0 entsprechend der Spezifikation 1.1.1, bietet aber zusätzlich Klassen zur Grafikausgabe, Dateimanagement, Ein- und Ausgabe sowie Netzwerkfunktionalität.
Es werden neben AWT 1.1 die folgenden PersonalJava-Klassenbibliotheken unterstützt:

```
java.lang, java.io, java.util, java.net, ja-
va.beans, java.awt, java.applet, ja-
va.lang.relect, java.util.zip, java.awt.image,
java.awt.datatransfer, java.awt.event
```

Der Compiler arbeitet wahlweise als Interpreter, JIT-Compiler oder auf Basis von vorcompiliertem Code (vgl. Kapitel 4.2.).

Während auf kleineren mobilen Endgeräten, wie den heute noch mit begrenzten Ressourcen ausgestatteten PDAs und Mobiltelefonen, die Java 2 Micro Edition mit ihren auf das Endgerät zugeschnittenen Konfigurationen und Profilen die bessere Wahl sein dürfte, ist der Einsatz von Kaffe auf größeren Endgeräten durchaus in Erwägung zu ziehen. Es werden viele Klassen unterstützt, es handelt sich um einen schnellen Compiler, und die Virtual Machine ist auf einer großen Zahl von Plattformen verfügbar.

4.3.10. Kada VM

Die Kada VM[99] für Win32- und PalmOS-Systeme ist in zwei Versionen erhältlich, die sich durch den benötigten Speicher und die Zahl der unterstützten Klassen unterscheiden:
Während die Kada Compact VM lediglich 155 KB Speicher benötigt, erfordert die Kada Standard VM 330 KB.

[99] Kada Systems, Kada VM, http://www.kadasystems.com/kada_vm.html (Stand: 01.09.2001)

Die Kompakt-Version beinhaltet einen Bytecode-Interpreter, Klassen-Lader, eine anspruchsvolle Garbage Collection und die Methoden der Pakete `java.lang, java.lang.reflect, java.io, java.net, java.util`.
Die Standard-Version bietet alle Funktionen der Compact VM sowie darüber hinaus die Pakete `java.util.zip` und `java.math` sowie einen Just-In-Time-Compiler (JIT).

Versionen für Windows CE, EPOC und RIM sollen demnächst erscheinen.

4.3.11. HP Chai VM

In der Chai-Produktlinie von Hewlett Packard ist neben einem Application Server auch eine Java Virtual Machine[100] enthalten, welche sich an Embedded-Systeme richtet. Sie ist lauffähig unter Windows CE und Red Hat Linux sowie - zur Evaluierung - auch unter Windows NT/x86.

Es werden die Klassen des JDK 1.1.8 unterstützt:

> `java.lang, java.io, java.util, java.net, java.awt, java.applet, java.lang.reflect, java.rmi, java.util.zip,`
> Java security (JDK 2.0).

Der Speicherbedarf („foot print“) beträgt 228 KB und bewegt sich damit im unteren Bereich im Vergleich mit anderen Virtual Machines für mobile Endgeräte.

Die Microchai VM ist kompatibel zur CLDC, eine Anpassung an die MIDP-Spezifikation ist in Vorbereitung.

4.3.12. Jeode Embedded VM

Bei der Jeode Embedded Virtual Machine (EVM)[101] von Insignia Solutions handelt es sich um eine kompakte VM, welche mit den Klassenbibliotheken von PersonalJava 1.2

[100] Hewlett Packard, Chai VM,
http://www.hp.com/products1/embedded/products/platform/chaivm.html (Stand: 02.09.2001)

[101] Insignia Solutions, Jeode Platform,
http://www.insignia.com/products/default.asp (Stand: 02.09.2001)

und EmbeddedJava 1.0.3 (einschließlich AWT) und einer IDE-Entwicklungsumgebung angeboten wird.
Laut Hersteller wurde bei der Entwicklung ein hoher Wert auf Kompatibilität zu Suns Java-Spezifikationen gelegt, sowie auf eine hohe Ausführungsgeschwindigkeit. Als Besonderheit gegenüber anderen VMs verfügt Jeode über einen Compiler mit „dynamic adaptive compilation“ (DAC), einer Technologie, welche ähnlich einem JIT-Compiler arbeitet und den Bytecode bei Bedarf vorcompiliert.

Die Jeode EVM ist neben Windows CE und NT auf einer Reihe von anderen Betriebssystemen für Embedded-Geräte lauffähig.

4.3.13. microJBlend

Aplix stellt Java-Umgebungen für Embedded-Systeme her. Bisher noch nicht veröffentlicht, ist mit „microJBlend für Windows CE“[102] erstmals eine Version für Windows CE geplant. Diese soll i-Appli (vgl. Kapitel 2.3.7.) und MIDP unterstützen.

4.3.14. Symbian EPOC JVM

Von Symbian, einem Hersteller von Betriebssystemen für mobile Endgeräte, stammt die Java-Implementierung für Psion-Organizer.[103,104]
Diese Java-Version ist auf Geräten mit dem EPOC-Betriebssystem lauffähig und liegt derzeit lediglich in der Version 1.00 vor. Java 1.1.4 wird unterstützt, außerdem JNI, RMI sowie eine AWT-Implementierung mit einem EPOC-„look-and-feel“. Unter Verwendung eines Kompatibilitäts-Paketes sind auch PersonalJava-Anwendungen lauffähig.

---

[102] Aplix Corp., JBlend, http://www.jblend.com/en/overview/microJBlend.html (Stand: 05.09.2001)

[103] Symbian, Java SDK for EPOC ER5, http://www.symbiandevnet.com/downloads/sdks/er5/er5javasdk.htm (Stand: 27.08.2001)

[104] Vgl. Psion, “Java Platform for Mobile Computers”, http://www.psionteklogix.com/main/java.htm sowie http://www.psion.com/downloads/jvm.html (Stand: 27.08.2001)

Das EPOC-Betriebssystem ist auch für Mobiltelefone verfügbar, doch nicht alle EPOC-Telefone unterstützen eine JVM.
Es ist geplant, in einer zukünftigen Version auch CLDC und MIDP zu unterstützen.[105]

4.3.15. Microsoft VM

Auch Microsoft bietet eine Virtual Machine an. Diese ist einerseits im Microsoft Internet Explorer enthalten, zum anderen aber auch im Rahmen des „Microsoft SDK for Java“ erhältlich.
Die aktuelle Version 4.0 ist zum JDK 1.1 kompatibel und für den Einsatz auf Windows-Systemen optimiert.

Aufgrund eines Rechtsstreites mit Sun wird Java von Seiten Microsofts derzeit jedoch nicht weiterentwickelt. Statt dessen versucht Microsoft, die Kunden zu einem Wechsel in Richtung C# zu bewegen. Es ist auch denkbar, daß Microsoft Java in einer eigenen Version weiterentwickeln wird, welche nicht mehr Bytecode-kompatibel zu anderen Sun-kompatiblen Virtual Machines sein wird.
Ergebnis wird in beiden Fällen sein, daß Microsofts Java-Version nicht mehr kompatibel zu den Standards von Sun sein wird und somit vermutlich nicht lange am Markt bestehen wird.

[105] Symbian, “Symbian on Java”, http://www.symbian.com/technology/keytech-bigjava.html (Stand: 11.09.2001)

5. Verwandte Technologien

5.1. Java-Derivate

Es gibt zahlreiche Programmiersprachen, mit denen sich Applikationen für mobile Endgeräte entwickeln lassen. Aufgrund der guten Portabilität und des weit verbreiteten Know-Hows bei einer breiten Basis von Entwicklern bietet sich Java in besonderem Maße an.
Es gibt allerdings einige an Java angelehnte Technologien, die eine einfache Entwicklung erlauben und dabei originäre Java-Technologien benutzen.

5.1.1. Programmiersprachen für JVM

So wie sich Virtual Machines konstruieren lassen, welche nicht mehr vollständig zu Java kompatibel sind, ist es umgekehrt auch möglich, mittels eines entsprechenden Compilers aus anderen Programmiersprachen einen Java-Bytecode zu erzeugen, der auf einer Standard-JVM ausgeführt werden kann.
Anstatt eine plattformspezifische ausführbare Datei zu erzeugen, wirft der Compiler einen Java-Bytecode aus. Selbstverständlich ist dies nicht mit jeder Programmiersprache möglich, doch es gibt eine Reihe von unterstützten Sprachen, und oft sind nur geringe Anpassungen an einem Quellcode erforderlich, um ein Programm einer anderen Sprache mit einem solchen Compiler umzuwandeln.
Es gibt bereits eine Unterstützung für Basic, Lisp, Logo, Cobol und eine Reihe anderer objektorientierter Programmiersprachen.
Eine gute Übersicht findet sich im Internet beispielsweise auf den Seiten von Robert Tolksdorf,[106] daher möchte ich an dieser Stelle nicht im einzelnen auf diese Sprachen und Compiler eingehen.

[106] Tolksdorf, Robert, „Programming Languages for the Java Virtual Machine", http://grunge.cs.tu-berlin.de/~tolk/vmlanguages.html (Stand: 08.09.2001)

5.1.2. Waba

Waba ist eine von der kalifornischen Firma Wabasoft entwickelte Programmierplattform für mobile Endgeräte. Waba definiert eine Sprache, eine Virtual Machine, ein Classfile-Format und eine Reihe von Foundation Classes.[107] Dabei ist Waba eng an Java angelehnt, die Syntax des Quellcodes entspricht der von Java. Somit sind bereits umfangreiche Kenntnisse bei Programmierern vorhanden. Sogar die Class-Dateien und der Bytecode von Waba entsprechen einer Teilmenge der von Java verwendeten Formate - dies ermöglicht die Nutzung bestehender Java-Entwicklungsumgebungen.

Die Waba Virtual Machine ist speziell für den Einsatz auf mobilen Endgeräten konzipiert. Es existieren Methoden für die Ansteuerung der grafischen Benutzerschnittstelle (GUI), zur Grafikausgabe, Eingabe per Zeichenstift, und auch Klassen zur Soundwiedergabe sind vorhanden. Allerdings ist Waba kein Java, besitzt also seine eigenen Klassen - und dies sind bei weitem nicht alle Klassen der J2SE.
Es stehen einige Portierungen für mobile Endgeräte zur Verfügung,[108] so zum Beispiel für Geräte mit PalmOS und Windows CE, für Newton, iPAQ und sogar einen Taschenrechner von Texas Instruments. Auch eine VM für MS-DOS ist erhältlich.
Darüber hinaus sind Waba-Programme mit Hilfe von „Brücken-Klassen“ sogar auf jeder Standard-JVM lauffähig, sowohl als Applikation als auch als Applet in einem Webbrowser.

Waba bietet sich für den Einstieg in die Programmierung für mobile Endgeräte an, da es leicht zu bedienen ist, gut für mobile Endgeräte geeignet ist und VMs für eine große Zahl von Plattformen erhältlich sind. Waba ist OpenSource, d.h. es ist kostenlos erhältlich und es fallen keine Lizenzgebühren an.

Vor allem bereitet die Installation des Waba SDK und die Installation auf einem PalmOS- oder Windows-CE-Gerät keine Schwierigkeiten, da alle benötigten Dateien in einem Paket enthalten sind. So lassen sich schnell erste Ergebnisse erzielen, auch

[107] Vgl. Wabasoft, Waba, http://www.wabasoft.com/products.shtml (Stand: 08.09.2001)

[108] Vgl. Wabasoft, “Waba VM Ports", http://www.wabasoft.com/vms.shtml (Stand; 08.09.2001)

beispielsweise in einer technischen Übung, selbst wenn bisher nur geringe Programmierkenntnisse vorhanden sind. [109]

Allerdings stellt sich die Frage, ob es nicht dennoch sinnvoll ist, sich gleich mit Suns Java 2 Micro Edition auseinanderzusetzen. Der Einstieg mag ein wenig komplizierter sein, zumal die Produktlinie von Sun Microsystems keineswegs auf den ersten Blick durchschaubar ist. Es ist ungleich schwieriger, alle benötigten Pakete und evtl. eine Emulator-Software herunterzuladen und zu installieren.
Dafür jedoch erhält man eine Umgebung, welche ebenfalls eine plattformunabhängige Entwicklung erlaubt, für eine Vielzahl von Endgeräten geeignet ist und vollständig zu den bestehenden Java-Standards kompatibel ist. Auf den WWW-Seiten von Sun[110] sowie auf vielen privaten Seiten[111] gibt es mittlerweile Anleitungen zur Installation und Konfiguration der J2ME-Software.
Wenn die Java 2 Micro Edition auch noch relativ neu ist, so wird sie mit ihren Konfigurationen und Profilen doch eine weite Verbreitung finden und einen De-facto-Standard darstellen.

[109] Vgl. Austermann, Anja, „Java zum Mitnehmen“, Toolbox, Ausgabe 4/2001 (Juli/August), Toolbox-Verlag, Wiedenzhausen, S. 6 ff.
Hier findet sich eine detaillierte Anleitung zum Einstieg in die Waba-Programmierung bis zur Entwicklung einer ersten Anwendung auf einem PDA.

[110] beispielsweise zur Konvertierung der MIDlets in PalmOS-Format:
Mahmoud, Qusay, „Developing Java Applications for Palm OS Devices”, http://developer.java.sun.com/developer/products/wireless/midp/articles-palm/convert/ (Stand: 09.09.2001)

[111] z.B. bei Robert Evans: http://webdev.apl.jhu.edu/~rbe/kvm/ (Stand: 08.09.2001)

5.2. WAP

Auf den ersten Blick erscheint es etwas kurios, WAP als eine mit Java verwandte Technologie zu bezeichnen. WAP ist ein Protokollstapel,[112] Java eine Programmiersprache bzw. die J2ME eine Java-Klassensammlung. Von daher haben beide nichts miteinander gemeinsam.
Erst bei näherer Betrachtung offenbaren sich Gemeinsamkeiten beider Technologien.
Sowohl WAP als auch Java sind geeignet, um mobil auf Informationen zuzugreifen und Inhalte auf einem Endgerät anzuzeigen.[113]

Bei WAP ist es möglich, dynamische WML-Seiten zu generieren, welche durch eine auf einem Webserver laufende Anwendung erzeugt werden. Ein Vorteil ist, daß hierzu auf dem Endgerät lediglich ein WAP-Browser installiert sein muß - weitere Ressourcen sind nicht erforderlich. Diese Voraussetzung erfüllen heutzutage die meisten Mobiltelefone. Auch sind flexiblere Applikationen möglich, als wenn diese fest auf dem Endgerät installiert sind. Beispielsweise sind benutzerspezifische Menüfunktionen wesentlich einfacher durch auf dem Server dynamisch generierte Menüs zu verwirklichen als mit Hilfe eines statischen Java-Programms, welches sich im Speicher des Endgerätes befindet.[114]

Sicherlich geht die J2ME weit über den Anspruch hinaus, lediglich Daten anzuzeigen. Vielmehr sind hier komplexe Anwendungen möglich, welche mit der Scriptsprache WML/WMLScript nicht realisierbar sind. Die Anwendungen laufen hier auf dem Endgerät ab. Nachdem eine Applikation und die gewünschten Daten auf das Gerät übertragen wurden, ist für die weitere Ausführung - im Gegensatz zu WAP - keine (teure) Netzverbindung mehr erforderlich. Dies ist bei den meisten Anwendungen wünschenswert und völlig ausreichend (beispielsweise bei Spielen).
Daten können persistent auf dem Endgerät gespeichert werden, und bei sicherheitsrelevanten Transaktionen ist der Benutzer bei der Verwendung von Java-MIDlets im Gegensatz zu WAP nicht auf Gateway-Server angewiesen, welche die Daten in ein anderes Protokoll umwandeln und damit ein Sicherheitsrisiko darstellen können.

---

[112] Vgl. Seminar zum Mobile Commerce, Wirtschaftsinformatik RWTH Aachen, http://www.wi.rwth-aachen.de/web/wi/mComGruppe2/index.html (Stand: 12.09.2001)

[113] Vgl. Haiges, Sven, „Java 2 Micro Edition und WAP“, Java Magazin, Heft 8/2001, S. 83 f.

[114] Vgl. Mortier, Peter, „WAP and J2ME”, http://www.mexeforum.org/articles.htm (Stand: 07.09.2001)

Mit Anwendungen der MIDP-Implementation ist außerdem eine weit detailliertere grafische Ausgabe möglich, da hier per Low-Level-Routinen der Bildschirm sowie andere Ressourcen des Endgerätes direkt angesprochen werden können. Dies ist mit WAP nicht möglich.

Zusammenfassend läßt sich feststellen, daß sich die Anwendungsgebiete beider Technologien zwar überschneiden, diese ihre Vorteile aber in sehr unterschiedlichen Bereichen aufweisen.
Java-Anwendungen unter J2ME stellen auf mobilen Endgeräten meist die beste Lösung dar, da die Netzverbindung nicht ständig benötigt wird - dies spart Kosten und Zeit für den Benutzer. Der Einsatz von WAP ist immer dann sinnvoll, wenn aktuelle Daten übertragen werden müssen und die Darstellung als Text ausreicht, oder wenn die Anwendung so komplex ist, daß sie nur auf einem Server ausgeführt werden kann, der über die nötigen Ressourcen verfügt.
Es bietet sich an, die Vorteile beider Technologien zu kombinieren. WAP ist hervorragend in der Lage, die im MID-Profil der J2ME fehlenden Komponenten zu ergänzen. Mit Absicht wurde in dieser Spezifikation kein Application Manager (JAM)[115] definiert, so daß dieser Teil mittels WAP-Technologien gefüllt werden kann. Mit einem WAP-Browser können dann MIDlets von einer Webseite ausgewählt werden, um sie auf das Endgerät herunterzuladen.

[115] Vgl. Kapitel 3.4.1.

5.3. Kompressionstechnologien

Ein Vorteil von WAP ist, daß nur Textinformationen und somit nur geringe Datenmengen an das Endgerät übertragen werden müssen. Bei komplexen Java-Applikationen oder zur Übertragung eines MIDlets an ein Endgerät sind diese Datenmengen ungleich höher.
An dieser Stelle setzen Kompressionstechnologien wie beispielsweise jaccelerator und jampagne der Münchener Firma Syntion[116] ein, welche - einmal auf dem Webserver und Endgerät installiert - den Datenstrom des Bytecodes durch intelligente Verfahren wie z.B. unter Nutzung einer syntaxorientierten Kodierung komprimieren.[117]

Auf diese Weise lassen sich die hochbelasteten Funknetze wesentlich entlasten. Dies ist positiv für die Netzbetreiber, welche nicht so früh in neue Bandbreiten investieren müssen, sowie für Endnutzer, für die sich die Verbindungsdauer verkürzt. Auch für Gerätehersteller ist diese Technologie interessant, da komprimierte Applikationen auch auf dem Endgerät weniger Speicher belegen.
jaccelerator liegt bisher in einer EPOC- und einer PalmOS-Version vor.

[116] Syntion AG, http://www.syntion.de/

[117] Vgl. Großwendt, Volkmar, „Java komprimiert“, Java Magazin, Heft 8/2001, S. 80 ff.

6. Werkzeuge für die Entwicklung von Java-Programmen

6.1. Emulatoren

Bei der Entwicklung von Software für mobile Endgeräte ist es in der Regel nicht möglich, die Applikation auf allen erforderlichen Plattformen, allen verfügbaren Endgeräten mit ihren unterschiedlichen Bildschirmgrößen und Ressourcen zu testen. Kaum ein Entwickler hat eine Vielzahl von Mobiltelefonen und PDAs zur Verfügung. Vielfach befinden sich die Geräte noch gar nicht auf dem Markt und stehen selbst noch im Entwicklungsstadium.
Daher ist es unerläßlich, die Applikation vorab mit Hilfe eines Emulators auf der Entwicklungsplattform (z.B. Windows) einem Test unterziehen zu können. Emulatoren sind teilweise in Entwicklungsumgebungen enthalten, das J2ME Wireless Toolkit stellt Emulationen verschiedener Endgeräte für MIDP zur Verfügung, und darüber hinaus sind auch freistehende Emulatoren erhältlich. Vor allem die IDEs der Endgerätehersteller wie Nokia, Motorola und Blackberry enthalten Emulationen der Endgeräte. So lassen sich die entwickelten Programme mit geringem Aufwand zuverlässig bereits am PC testen.

Einer dieser Emulatoren ist der Palm OS Emulator, der im folgenden vorgestellt wird. Emulatoren, die in die Entwicklungsumgebungen integriert sind, werden in Kapitel 6.2. besprochen.

Palm OS Emulator
Der Palm OS Emulator (POSE) erlaubt es, Anwendungen des PalmOS auf einer Windows-Plattform zu testen und zu korrigieren (Debugging). Emuliert wird das PalmOS, welches in Form eines ROM-Files (wie es auch im Hardware-PDA verwendet wird) installiert werden muß. Es lassen sich verschiedene Skins (Schablonen) für die Ansicht des entsprechenden Endgerätes laden, so daß alle Kombinationen von Gerätetyp und Betriebssystem am Desktop-Bildschirm geprüft werden können (vgl. Abbildung 10).
Nicht testen läßt sich die Infrarot-Schnittstelle, die HotSync-Funktionen stehen jedoch zur Verfügung.

Der Palm OS Emulator steht frei im Internet zum Abruf bereit,[118] die erforderlichen ROM-Dateien müssen aus einem Palm Organizer geladen werden bzw. sind nach (kostenloser) Anmeldung zum Palm OS Developer Program auf den Internet-Seiten von PalmOS erhältlich.

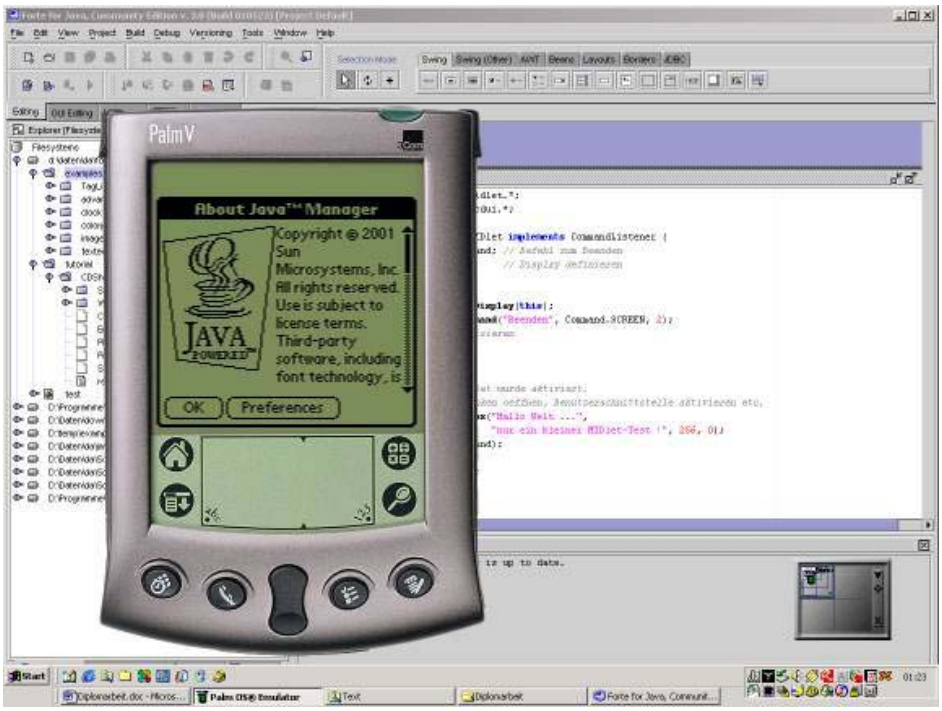

Abbildung 10: Der Palm OS Emulator (hier in Form eines Palm V)

6.2. Entwicklungsumgebungen

Es existieren zahlreiche Entwicklungsumgebungen für Java, von denen einige eine spezielle Unterstützung für die Entwicklung von Software für mobile Endgeräte bieten. Manche Produkte enthalten lediglich Kommandozeilen-Hilfsprogramme, andere bieten eine vollständige grafische Umgebung mit Code-Editor.

Im folgenden möchte ich einige nützliche Programme vorstellen. Darüber hinaus sind noch einige weitere Werkzeuge z.B. von IBM, Nokia, Motorola, RIM (Blackberry), Webgain und anderen erhältlich.[119]

[118] Palm Inc., POSE, http://www.palmos.com/dev/tech/tools/emulator/ (Stand: 01.09.2001)

[119] Vgl. Giguère, Eric, „Java 2 Micro Edition“, S. 193 ff.

6.2.1. J2ME Wireless Toolkit

Mit dem J2ME Wireless Toolkit,[120] welches zur Zeit in der Version 1.0.3 Beta vorliegt, bietet Sun dem Entwickler von MIDP-Anwendungen eine Hilfestellung bei der Entwicklung seiner Applikationen.

Ursprünglich werden die Java-Komponenten per Kommandozeile gesteuert, was besonders Anwendern Probleme bereitet, die bisher über keinerlei MS-DOS-Kenntnisse verfügen. Die Programme müssen compiliert sowie die Manifest- und die JAD-Datei erstellt werden.

Diese Aufgaben lassen sich mit dem Toolkit wesentlich vereinfachen. Darüber hinaus bietet das Toolkit einen Emulator, mit dem sich verschiedene MIDP-kompatible Endgeräte simulieren lassen (vgl. Abbildung 11). Hier stehen bisher ein Standard-Telefon mit Graustufen, eines mit Farbunterstützung, ein Telefon mit Minimal-Ausstattung, ein Motorola i85s sowie ein RIM Blackberry 957 zur Auswahl, in der neuesten Version ist außerdem eine komfortable Anbindung an den Palm OS Emulator integriert.

Eine grafische Entwicklungsumgebung ist nicht enthalten, doch es stehen Module für die Integration in die Forte-IDE bereit, mit denen sich das Toolkit nahtlos in diese Umgebung integrieren läßt. Auch eine Einbindung in Borlands JBuilder ist möglich.

Abbildung 11: Das J2ME Wireless Toolkit

Problematisch ist lediglich, daß weder eine realistische Simulation der Ausführungsgeschwindigkeit erfolgen kann, noch ein Test des Download-Vorganges einer MIDlet-Suite per Application Manager simuliert werden kann - ein solcher ist bisher nicht integriert.[121] Doch diese Restriktionen sind in annähernd allen Entwicklungsumgebungen vorhanden.

6.2.2. Netbeans

Bei Netbeans[122] handelt es sich um eine grafische Java-Entwicklungsumgebung (vgl. Abbildung 12). Sie ist selbst in Java programmiert, wird im Rahmen eines Open-Source-Projektes entwickelt und ist somit auf unterschiedlichen Plattformen lauffähig.
Die IDE, welche im Moment in der Version 3.2.1 vorliegt, erlaubt die Verwaltung von Java-Projekten. Sie verwaltet dabei die zugehörigen Quellcode- und Klassendateien und beinhaltet neben einem komfortablen Quellcode-Editor die Möglichkeit, den Code per Mausklick zu compilieren und Fehler zu beheben (Debugging).

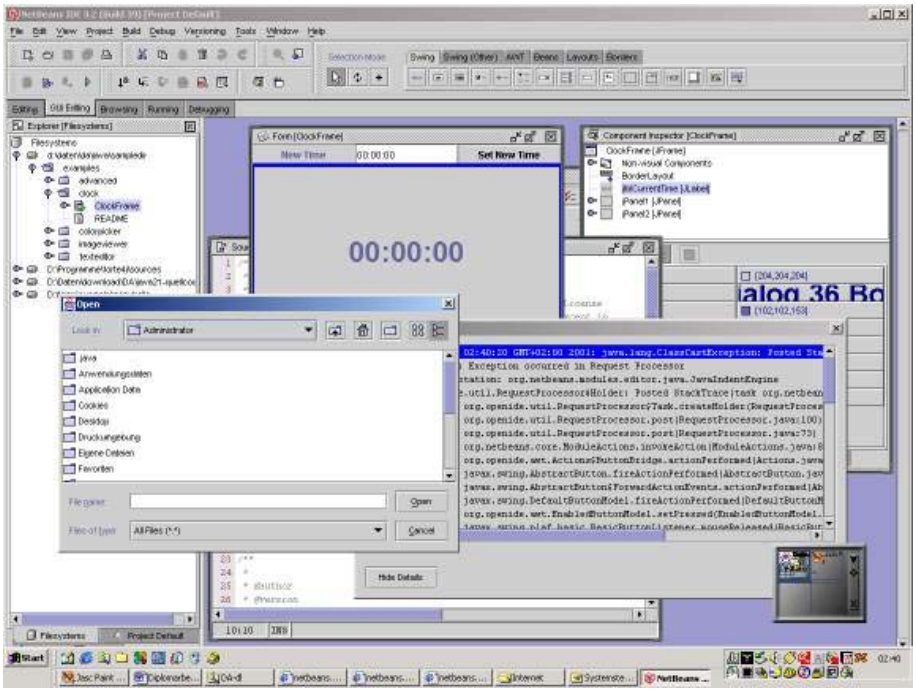

Abbildung 12: Die Benutzeroberfläche der Netbeans IDE

[120] Sun Microsystems, J2ME Wireless Toolkit, http://developer.java.sun.com/developer/earlyAccess/j2mewtoolkit/ (Stand: 10.09.2001)

[121] Vgl. Glahn, Kay, „Vom Labor aufs Handy“, Java Magazin, Heft 3/2001

[122] http://www.netbeans.org/

6.2.3. Forte

In der Forte-Reihe bietet Sun auch eine Entwicklungsumgebung für Java an.[123] Dabei handelt es sich im Prinzip um die Netbeans-IDE, die allerdings unter anderen Lizenzbedingungen vertrieben wird.
Die Software wird in drei Versionen angeboten: die Forte for Java Community Edition ist kostenfrei erhältlich, während die Internet- und Enterprise Edition nur gegen Entgelt angeboten werden. Sie besitzen gegenüber der Community Edition einige Zusätze für die Programmierung für Webserver bzw. im Rahmen der J2EE, außerdem steht für die kostenpflichtigen Produkte eine Support-Hotline zur Verfügung.

Im Gegensatz zu Netbeans läuft Forte etwas zuverlässiger und stabiler, allerdings werden neue Technologien erst etwas später als in Netbeans integriert.
Die grafische Oberfläche ist in beiden Programmen annähernd identisch.

6.2.4. Borland JBuilder Mobile Set Nokia Edition

Für das JBuilder-Entwicklungspaket von Borland ist mit dem JBuilder Mobile Set[124] eine Erweiterung erhältlich, welche die kostenlos erhältliche JBuilder 5 Personal Edition um eine J2ME-Kompatibilität erweitert. Auch eine Integration in die JBuilder 5 Professional oder Enterprise Edition ist möglich, hier stehen weitere Funktionen zur Verfügung.
Zusätzlich läßt sich die Nokia Developer Suite[125] integrieren, welche die Funktionen des JBuilder Mobile Sets weiter ergänzt. Sie bietet Hilfestellung bei der Erstellung von MIDlet-Suites und Application Descriptors sowie einige Automatisierungsfunktionen.

Installiert man alle drei genannten Pakete, so erhält man eine sehr umfangreiche Umgebung zur Entwicklung von MIDP-kompatiblen Applikationen, einschließlich Code-Editor, Werkzeugen zur Erzeugung von MIDlet-Suites und einem Device Emulator.[126]

---

[123] Sun Microsystems, Forte4Java, http://www.sun.com/forte/ffj/index.html (Stand: 07.09.2001)

[124] Borland, JBuilder Mobile Set Nokia Edition, http://www.borland.com/jbuilder/mobileset/ (Stand: 22.08.2001)

[125] Nokia, Nokia Developer Suite for J2ME, http://www.forum.nokia.com/ (Stand: 24.08.2001)

[126] Vgl. Glahn, Kay, „JBuilder meets Nokia", Java Magazin, Heft 8/2001, S. 78 f.

Von dem Zusatz „Nokia Edition“ darf man sich nicht täuschen lassen, denn selbstverständlich werden MIDP-kompatible Applikationen erzeugt, die auf jedem kompatiblen Endgerät lauffähig sind - nicht nur auf Nokia-Geräten. Der Name wurde lediglich aus Marketinggründen gewählt und resultiert aus der Zusammenarbeit mit der Firma Nokia, welche die Zusatzwerkzeuge der Nokia Developer Suite beigesteuert hat und die Funktionen des Emulators (in Form eines Nokia-Telefons) bereitstellt.

6.2.5. Zucotto WHITEboard

Die Firma Zucotto bietet, wie bereits in Kapitel 4.2.6. beschrieben, ein Hardware-Entwicklungskit zur Entwicklung von Java-Prozessoren an. Passend zu diesem ist eine Software-Entwicklungsumgebung erhältlich, welche MIDP-kompatibel ist und sich nicht nur für das oben genannte XPRESSOboard eignet, sondern auch für die Entwicklung anderer J2ME-Anwendungen.
Die IDE namens „WHITEboard SDK”[127] basiert auf der IDE von Netbeans, ist jedoch speziell für die Entwicklung mobiler Applikationen angepaßt worden. Sie unterstützt die J2ME vollständig, einschließlich CLDC und MIDP. Leider ist WHITEboard nur für die Windows-Plattform erhältlich, obwohl Netbeans als Java-Anwendung ursprünglich auch auf anderen Plattformen lauffähig wäre.
Die IDE zeichnet sich durch ihren modularen Aufbau aus, denn der Device Emulator, Device Editor und Packager können auch eigenständig genutzt oder sogar in andere Entwicklungsumgebungen integriert werden.
Der Device Emulator unterstützt ein PDA-Profil, ein Standard-MIDP-Telefon sowie einen Pager, der Device Editor erlaubt darüber hinaus jedoch auch die Definition eigener Profile. Der Packager dient der Erstellung von MIDlet-Suites einschließlich JAR-Archiv, Manifest-Datei und der JAD-Datei.[128]

Mit dem WHITEboard SDK erhält man eine fast schon ideale Entwicklungsumgebung für CLDC- und MIDP-Applikationen. Zusätzlich ist sogar eine Bluetooth-Edition erhältlich, welche die benötigte Hardware umfaßt und die Entwicklung von Bluetooth-Anwendungen erlaubt.

---

[127] Zucotto Wireless, WHITEboard SDK,
http://www.zucotto.com/whiteboard/product_downloads.html (Stand: 11.09.2001)

[128] Vgl. Glahn, Kay, „Vom Labor aufs Handy“, Java Magazin, Heft 3/2001

7. Zusammenfassende Betrachtung und Ausblick

Entgegen der weit verbreiteten Ansicht, Java sei eine träge, komplizierte Programmiersprache, die kaum für Geräte mit eingeschränkten Ressourcen geeignet ist, befindet sich Java mit den Technologien der Java 2 Micro Edition auf dem richtigen Weg, die Voraussetzungen im mobilen Bereich besser zu erfüllen, als manche andere Programmiersprache. Mit den Konfigurationen und Profilen der J2ME lassen sich die Applikationen besser an die unterschiedlichen Endgeräte anpassen, als dies mit anderen Programmiersprachen möglich wäre.
Sicherlich wären andere Sprachen wie zum Beispiel C++ ebenso geeignet, doch mittlerweile findet Java eine breite Unterstützung bei Geräteherstellern und Programmierern. Inzwischen befindet sich eine Vielzahl von Produkten auf dem Markt, die neue Anwendungsmöglichkeiten bieten und eine gute Basis für den Mobile Commerce darstellen.

Leider sind diese Technologien nicht immer kompatibel zueinander. Dies ist zwar nicht in jedem Fall erforderlich, doch es vereinfacht die Portierung unter den einzelnen Geräten und damit die Verbreitung von Software wesentlich, wenn es klar definierte Standards gibt. Hier setzt der „Java Community Process“ an, der es jedem interessierten Hersteller und Entwickler ermöglicht, konstruktiv auf die Definition neuer Standards Einfluß zu nehmen und an der Form neuer Technologien mitzuarbeiten. Auf diese Weise kann verhindert werden, daß sich Java - wie es zuvor schon bei einigen anderen vielversprechenden Technologien eingetreten ist - trotz einer guten Funktionalität mangels verläßlicher Standards nicht durchsetzen kann.
Es gibt zwar zur Zeit einige Hersteller, welche proprietäre Lösungen auf den Markt bringen, doch die Java 2 Micro Edition wird mit ihren Konfigurationen und Profilen allein schon aufgrund der hohen Zahl der unterstützenden Hersteller und Entwickler in Zukunft den Standard im bezug auf Java-Technologien für mobile Endgeräte darstellen. Vermutlich wäre es für die Gesamtentwicklung besser, wenn sich möglichst viele Hersteller dem Java Community Process anschließen würden.

Nun ist es an der Zeit, mit Hilfe von neuen Technologien wie der dritten Generation der Mobilfunknetze (GPRS, HSCSD, UMTS) und neuen, leistungsfähigeren Endgeräten die Basis für neue Anwendungen zu schaffen.

Hierfür wird neue Software benötigt werden. Java bietet sich für diese Aufgabe geradezu an. Bei einer steigenden Zahl von Endgeräten wird die Ausstattung dieser Geräte immer unterschiedlicher werden, so daß die Notwendigkeit verschiedener Konfigurationen immer größer werden wird. Die J2ME verfügt aufgrund ihres offenen Entwicklungskonzeptes (Java Community Process) über ein geeignetes Potential, auch längerfristig den Anforderungen der Programmierung mobiler Endgeräte gewachsen zu sein.

Stehen erst einmal leistungsfähige Endgeräte in Form von PDAs und Mobiltelefonen mit den entsprechenden Übertragungskapazitäten in den Mobilfunknetzen bereit, so werden sicherlich viele Anwender diese neuen Technologien einsetzen.
Die erforderliche Softwareplattform steht mit Java jedenfalls schon bereit.

Literaturverzeichnis

Austermann, Anja, „Java zum Mitnehmen“, Toolbox, Ausgabe 4/2001 (Juli/August), Toolbox-Verlag, Wiedenzhausen

Ericsson, „Mobile Applications with J2ME, A White Paper“, http://www.ericsson.de/upload/download/white_paper_j2me.pdf (Stand: 19.08.2001)

Giguère, Eric, „Fallacies About Java 2 Micro Edition“, http://www.ericgiguere.com/microjava/fallacies.html (Stand: 15.08.2001)

Giguère, Eric, „Java 2 Micro Edition“, Professional Developer´s Guide Series, John Wiley & Sons, Inc., New York, Toronto, Weinheim, 1999

Glahn, Kay, „Java macht mobil“, Java Magazin, Heft 3/2001

Glahn, Kay, „JBuilder meets Nokia”, Java Magazin, Heft 8/2001

Glahn, Kay, „VM Anywhere - Virtuelle Java Maschinen auf fast jeder Plattform“, Java Magazin, Heft 5/2001, S. 40

Glahn, Kay, „Vom Labor aufs Handy“, Java Magazin, Heft 3/2001

Glahn, Kay, Meyen, Sebastian, „Die Straßen von SF“, Java Magazin, Heft 8/2001, http://www.javamagazin.de/ausgaben/2001/8/artikel/3/online.shtml (Stand: 24.07.2001)

Großwendt, Volkmar, „Java komprimiert“, Java Magazin, Heft 8/2001, S. 80 ff.

Haiges, Sven, „Java 2 Micro Edition und WAP“, Java Magazin, Heft 8/2001

Java Magazin, Software & Support Verlag, Frankfurt, Heft 5/2001

Lemay, Laura, Cadenhead, Roger, „JAVA 2”, Markt + Technik Verlag, München, 2001

Lindholm, Tim, Yellin, Frank, "The Java Virtual Machine Specification, Second Edition”, Addison Wesley, Reading (MA), 1999, http://java.sun.com/docs/books/vmspec/2nd-edition/html/VMSpecTOC.doc.html (Stand: 27.08.2001)

Llobregat, Fernando, “Developing Applications Using the MIDP/UI APIs”, JavaOne-Conference 2000 San Francisco, http://servlet.java.sun.com/javaone/javaone2000/pdfs/TS-1666.pdf (Stand: 13.08.2001)

Mahmoud, Qusay, „Developing Java Applications for Palm OS Devices”, http://developer.java.sun.com/developer/products/wireless/midp/articles-palm/convert/ (Stand: 09.09.2001)

Mahmoud, Qusay, „The J2ME Platform - Which APIs Come from the J2SE Platform ?“, http://developer.java.sun.com/developer/technicalArticles/wireless/midpapi/ (Stand: 23.08.2001)

Merck, Martin, Sun Microsystems GmbH, „Java 2 Micro Edition - Die Basis für Wireless Geräte“ auf der Sun-Konferenz „The Future is Wireless“ im Mai 2001 in Neuss, http://www.sun.de/Downloads/Praesentationen/Wireless/Vortraege/1/Merck.pdf (Stand: 01.08.2001)

Moreira, Paulo, Micro Java Network, “From PersonalJava to J2ME: Some Introductory Ideas”, http://www.kvmworld.com/news/perspective/personaljava_j2me?content_id=1440 (Stand: 13.07.2001)

Mortier, Peter, „WAP and J2ME”, http://www.mexeforum.org/articles.htm (Stand: 07.09.2001)

Raner, Mirko, „Der Motor: Implementation der Java Virtual Machine“, Java Magazin, Heft 5/1999

Seminar zum Mobile Commerce, Wirtschaftsinformatik RWTH Aachen, http://www.wi.rwth-aachen.de/web/wi/mComGruppe2/index.html (Stand: 12.09.2001)

Sun Microsystems, „Java 2 Micro Edition (J2ME) Technology for Creating Mobile Devices, White Paper”, http://java.sun.com/products/cldc/wp/KVMwp.pdf (Stand: 23.08.2001)

Sun Microsystems, “Java Remote Method Invocation Specification”, http://java.sun.com/products/jdk/1.2/docs/guide/rmi/spec/rmiTOC.doc.html (Stand: 03.09.2001)

Sun Microsystems, “Technical Overview of EmbeddedJava Technology”, http://java.sun.com/products/embeddedjava/overview.html (Stand: 18.08.2001)

Sun Microsystems, “PersonalJava Specification 1.2”, http://java.sun.com/products/personaljava (Stand: 20.08.2001)

Sun Microsystems, „The J2ME Platform - Which APIs Come from the J2ME Platform?", http://developer.java.sun.com/developer/technicalArticles/wireless/midpapi/ (Stand: 23.08.2001)

Thomas, Karsten, „Push-Technologien", http://www.hemmerden.de/mcommerce/ (Stand: 12.09.2001)

wap.de, "NTT DoCoMo mit mobilen Internet-Service auf Java-Basis", http://www.wap.de/News/Archiv/2001/01/N010131ntt,version=3.html (Stand: 13.07.2001)

Yellin, Frank, "Inside the K Virtual Machine", Sun Microsystems, JavaOne-Conference, http://servlet.java.sun.com/javaone/javaone2000/pdfs/TS-1507.pdf (Stand: 05.09.2001)

Weitere Quellen und hilfreiche Adressen im World Wide Web:

aJile Systems, http://www.ajile.com/products.htm

Aplix Corp., JBlend, http://www.jblend.com/en/overview/microJBlend.html (Stand: 05.09.2001)

ARM, ARM Jazelle Technology, http://www.arm.com/armtech/jazelle?OpenDocument (Stand: 07.09.2001

Borland, JBuilder Mobile Set Nokia Edition, http://www.borland.com/jbuilder/mobileset/ (Stand: 22.08.2001)

Clark, Rob, „Distributed Java VM", http://www.cs.hmc.edu/~rclark/dj/index.html (Stand: 07.09.2001)

Credis Suisse, youtrade, http://www.credit-suisse.ch/de/direktzugriff/pcbank/youtrade/index.html (Stand: 10.09.2001)

Compaq, iPAQ Blackberry, http://www.compaq.com/products/handhelds/blackberry/index.html

EmbeddedJava, http://java.sun.com/products/embeddedjava/ (Stand: 18.08.2001)

Esmertec, Inc., Jbed Micro Edition CLDC, http://www.esmertec.com/p_jbed_cldc_long.html (Stand: 07.09.2001)

Extensible Markup Language, entwickelt vom World Wide Web Consortium (W3C), http://www.w3c.org/

Evans, Robert, http://webdev.apl.jhu.edu/~rbe/kvm/ (Stand: 08.09.2001)

Hewlett Packard, Chai VM, http://www.hp.com/products1/embedded/products/platform/chaivm.html (Stand: 02.09.2001)

IBM, J9 VM, http://www.embedded.oti.com/learn/vaesvm.html (Stand: 20.08.2001)

Insignia Solutions, Jeode Platform, http://www.insignia.com/products/default.asp (Stand: 02.09.2001)

JavaCheck, http://java.sun.com/products/personaljava/javacheck.html (Stand: 21.08.2001)

Java Community Process (JCP), http://jcp.org/

Java Community Process, JSR-30, http://jcp.org/jsr/detail/30.jsp (Stand: 30.07.2001)

Java Community Process, JSR-36, http://jcp.org/jsr/detail/036.jsp (Stand: 30.07.2001)

Java Community Process, JSR-46, http://jcp.org/jsr/detail/46.jsp (Stand: 02.09.2001)

Java Community Process, JSR-62, http://jcp.org/jsr/detail/62.jsp (Stand: 02.09.2001)

Java Community Process, JSR-66, http://jcp.org/jsr/detail/66.jsp (Stand: 02.09.2001)

Java Community Process, JSR-75, http://jcp.org/jsr/detail/75.jsp (Stand: 04.09.2001)

Java Community Process, JSR-118, http://jcp.org/jsr/detail/118.jsp (Stand: 05.09.2001)

Java Community Process, JSR-120, http://jcp.org/jsr/detail/120.jsp (Stand: 06.09.2001)

Java Community Process, JSR-135, http://jcp.org/jsr/detail/135.jsp (Stand: 06.09.2001)

Java Community Process, JSR-901, http://jcp.org/jsr/detail/901.jsp (Stand: 02.08.2001)

JavaMobiles, http://www.javamobiles.com/

JavaMobiles, http://www.javamobiles.com/midlets/index.html (Stand: 06.08.2001)

JavaMobiles, „List of JVM for PDA“, http://www.javamobiles.com/jvm.html#home

JStamp, http://jstamp.systronix.com/info.htm

Kada Systems, Kada VM, http://www.kadasystems.com/kada_vm.html (Stand: 01.09.2001)

Kroll & Haustein GbR, xKVM / kAWT, http://www.kAWT.de (Stand: 07.09.2001)

Nokia, Nokia Developer Suite for J2ME, http://www.forum.nokia.com/
(Stand: 24.08.2001)

NSIcom Ltd., CrEme, http://www.nsicom.com/products/creme.asp (Stand: 04.09.2001)

Palm Inc., PalmOS-Emulator, http://www.palmos.com/dev/tech/tools/emulator/

PersonalJava Emulation Environment, http://java.sun.com/products/personaljava/pj-emulation.html (Stand: 21.08.2001)

Psion, “Java Platform for Mobile Computers”,
http://www.psionteklogix.com/main/java.htm
http://www.psion.com/downloads/jvm.html (Stand: 27.08.2001)

PTSC, http://www.ptsc.com/

Qualcomm Inc., http://www.qualcomm.com/

SavaJe Technologies, SavaJe XE, http://www.savaje.com/products/savajexe.html
(Stand 10.09.2001)

Schmidt, Marc, “List of Java Compilers and Virtual Machines”,
http://www.geocities.com/marcoschmidt.geo/java.html

Sun Microelectronics, picoJava, http://www.sun.com/microelectronics/picoJava/

Sun Microelectronics, Sun Community Source Licensing,
http://www.sun.com/microelectronics/communitysource/ (Stand: 07.09.2001)

Sun Microsystems, “CLDC and the K Virtual Machine (KVM)”,
http://java.sun.com/products/cldc/ (Stand: 29.08.2001)

Sun Microsystems, Forte4Java, http://www.sun.com/forte/ffj/index.html
(Stand: 07.09.2001)

Sun Microsystems, Java 2 Enterprise Edition, http://java.sun.com/j2ee/

Sun Microsystems, Java 2 Micro Edition, http://java.sun.com/j2me/

Sun Microsystems, Java 2 Standard Edition, http://java.sun.com/j2se/

Sun Microsystems, Java Card Technology, http://java.sun.com/products/javacard/
(Stand: 19.08.2001)

Sun Microsystems, JavaPhone API, http://java.sun.com/products/javaphone/
(Stand: 30.06.2001)

Sun Microsystems, MIDP Reference Implementation for Microsoft Windows, http://javashoplm.sun.com/ECom/docs/Welcome.jsp?StoreId=5&PartDetailId=MIDP-1.0-WIN-G-CS&TransactionId=communitySource (Stand: 30.08.2001)

Sun Microsystems, Mobile Information Device Profile (MIDP) for Palm OS, http://java.sun.com/products/midp/palmOS.htm (Stand: 30.08.2001)

Sun Microsystems, “PersonalJava Application Environment”, http://java.sun.com/products/personaljava/ (Stand: 20.08.2001)

Sun Microsystems, Spotless, http://www.sun.com/research/spotless/ (Stand: 26.07.2001)

Symbian, Java SDK for EPOC ER5, http://www.symbiandevnet.com/downloads/sdks/er5/er5javasdk.htm (Stand: 27.08.2001)

Symbian, “Symbian on Java”, http://www.symbian.com/technology/keytech-bigjava.html (Stand: 11.09.2001)

Syntion AG, http://www.syntion.de/

Tolksdorf, Robert, „Programming Languages for the Java Virtual Machine”, http://grunge.cs.tu-berlin.de/~tolk/vmlanguages.html (Stand: 08.09.2001)

Transvirtual Technologies Inc., Kaffe OpenVM, http://www.kaffe.org/ sowie http://www.transvirtual.com/datasheet.pdf (Stand 08.09.2001)

Wabasoft, Waba, http://www.wabasoft.com/products.shtml (Stand: 08.09.2001)

Wabasoft, “Waba VM Ports", http://www.wabasoft.com/vms.shtml (Stand; 08.09.2001)

Zucotto Wireless, WHITEboard SDK, http://www.zucotto.com/whiteboard/product_downloads.html (Stand: 11.09.2001)

Zucotto Wireless, XPRESSOboard HDK, http://www.zucotto.com/products/xpressoboard_index.html (Stand: 07.09.2001)

Eine Fassung dieser Diplomarbeit im Adobe Acrobat PDF-Format ist unter der Adresse `http://www.hemmerden.de/diplomarbeit/` abrufbar.

Versicherung

Ich versichere hiermit, daß ich die vorliegende Arbeit selbständig und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe. Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten und nicht veröffentlichten Schriften entnommen sind, sind als solche kenntlich gemacht. Die Arbeit ist in gleicher oder ähnlicher Form noch nicht als Prüfungsarbeit eingereicht worden.

Grevenbroich, 13.09.2001