

Fachhochschule Gießen

Wireless Java

Seminarausarbeitung

Tim Pommerening

SS 2003

Mittenaar, 31.03.2003

Inhalt

1	EINLEITUNG	3
1.1	DIE JAVA 2 PLATTFORM.....	3
1.2	WIRELESS JAVA	3
2	VIRTUELLE MASCHINEN	4
2.1	CVM	5
2.2	KVM	5
2.2.1	<i>Einschränkungen der KVM</i>	5
3	KONFIGURATIONEN	6
3.1	CDC	6
3.2	CLDC	7
3.3	DIREKTER VERGLEICH	7
4	PROFILE	8
4.2	DAS MID PROFIL	8
4.2.1	<i>MIDP 1.0</i>	8
4.2.1	<i>MIDP 2.0</i>	8
4.2	DAS PDA PROFIL.....	9
4.3	DAS FOUNDATION PROFIL.....	9
4.4	DAS PERSONAL PROFIL	9
5	ALTERNATIVE IMPLEMENTIERUNGEN	11
5.1	PERSONAL JAVA	11
5.2	WABA.....	11
5.3	J9	11
5.4	JEODE.....	11
5.6	SAVAJE	11
6	SMARTCARDS	12
6.1	CARD VM.....	12
6.2	JAVACARD API	12
7	FAZIT	13
A	LITERATUR	14
B	ABBILDUNGEN	14

1 Einleitung

Das folgende Dokument ist die Ausarbeitung des „Wireless Java“ Vortrags für das Rechnernetze Seminar an der Fachhochschule Gießen-Friedberg im Sommersemester 2003. Es gibt zuerst eine Übersicht über die Java 2 Plattform von SUN und behandelt dann schwerpunktmäßig die Java 2 Micro Edition, die speziell für mobile Endgeräte ausgelegt ist. Dabei wird auf jede der Schichten dieser API gesondert eingegangen und die Unterschiede aufgezeigt, die sich für die große Vielfalt an mobilen Endgeräten ergeben. Abschließend werden noch kurz Alternativen zur Java 2 Micro Edition vorgestellt und ein Ausblick auf Java für Smartcards gegeben.

1.1 Die Java 2 Plattform

Die Java 2 Plattform wurde und wird von der Firma SUN entwickelt und stellt das Grundgerüst von Java dar. Sie kann in drei Unterbereiche unterteilt werden, die so genannten Editionen. Jede dieser Editionen enthält eine Vielzahl an vorgefertigten Klassen, die für die speziellen Anwendungsbereiche optimiert sind. Abbildung 1 zeigt, wie sich die drei Java 2 Editionen zueinander verhalten. Die

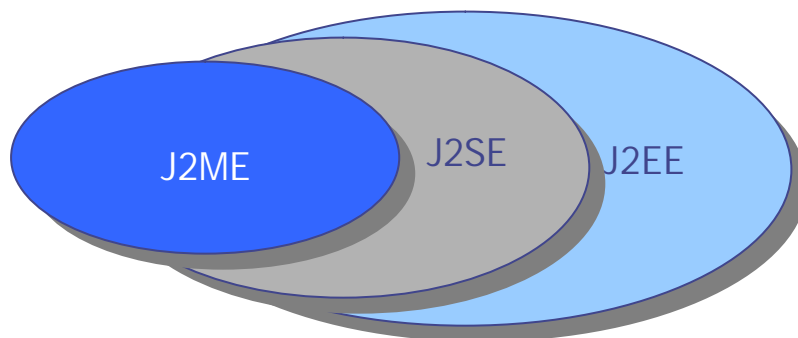


Abbildung 1: Die Java 2 Plattform (Vgl. [MM03])

bekannteste Java Edition ist die Java 2 Standard Edition. Sie enthält die wichtigsten APIs für die Programmentwicklung im Anwendungsbereich und verwendet auch die gebräuchlichste virtuelle Maschine für die Ausführung des Programmcodes, die so genannte Java Virtual Machine (JVM). Die Standard Edition bildet eine Untermenge der Java 2 Enterprise Edition. Dies ist die umfang-

reichste Edition und bietet neben der Funktionalität der Standard Edition außerdem erweiterte Funktionen für den Netzwerk- und Serverbereich, sowie Datenbanksysteme. Dies ist vor allem im Unternehmensbereich interessant. Die Enterprise Edition verwendet die so genannte **Hotspot Virtual Machine**. Die Kapselung zwischen den Editionen dient einzig und allein dem Performancegewinn. Als dritte Edition muss noch die Java 2 Micro Edition genannt werden. Diese Edition sollte eigentlich wiederum eine Untermenge der Standard Edition sein. Es wurde jedoch schnell klar, dass das bei der Vielzahl der mobilen Gerätearchitekturen nicht durchzuhalten war. Daher wurde die Micro Edition derart modifiziert, dass sie auf den mobilen Geräten sinnvoll eingesetzt werden kann. Sie bildet nun keine Untermenge der J2SE mehr, sondern enthält nur noch Teile dieser Edition, dafür aber auch APIs, die in der Standard Edition nicht vorhanden sind [MM03].

1.2 Wireless Java

Wenn von Wireless Java die Rede ist, dann meint man vor allem die bereits im letzten Abschnitt eingeführte Java 2 Micro Edition. Es gibt zwar auch alternative Ansätze, um Java für mobile Geräte verfügbar zu machen, aber die J2ME ist sicherlich als Referenz zu betrachten und daher auch Hauptgegenstand dieser Ausarbeitung. Durch die Aufteilung der J2ME in APIs aus der Standard Edition und gerätespezifische APIs stand man vor dem Dilemma, dass man sowohl für jede Gerätearchitektur unterschiedliche Funktionalität benötigte, gleichzeitig aber die Plattformunabhängigkeit der Java-Programme gewährleisten musste. Aus diesem Grund wurde für die Micro Edition ein Schichtenmodell aus mindestens vier Schichten entwickelt, wobei auf die oberste Schicht dabei beliebig aufgebaut werden kann. Abbildung 2 zeigt dieses Schichtenmodell. Die unterste Schicht bildet die virtuelle Maschine, die direkt auf dem Betriebssystem liegt und gewährleistet, dass

die Applikation auf jeder Rechnerarchitektur ausgeführt werden kann, die eine kompatible virtuelle Maschine einsetzt. Auf diese aufbauend folgt eine Schicht mit Basis APIs, die auf allen mobilen javafähigen Geräten verfügbar sein müssen. Diese Basis APIs bilden auch die Schnittmenge zur Standard- und zur Enterprise Edition. Konkret gesagt befinden sie sich im Paket `java.lang`, welches auf allen Editionen gleichermaßen vorhanden ist. Darüber liegt die Konfigurationsschicht und auf dieser schließlich die Profilschicht. Die obersten beiden Schichten bilden immer noch einen

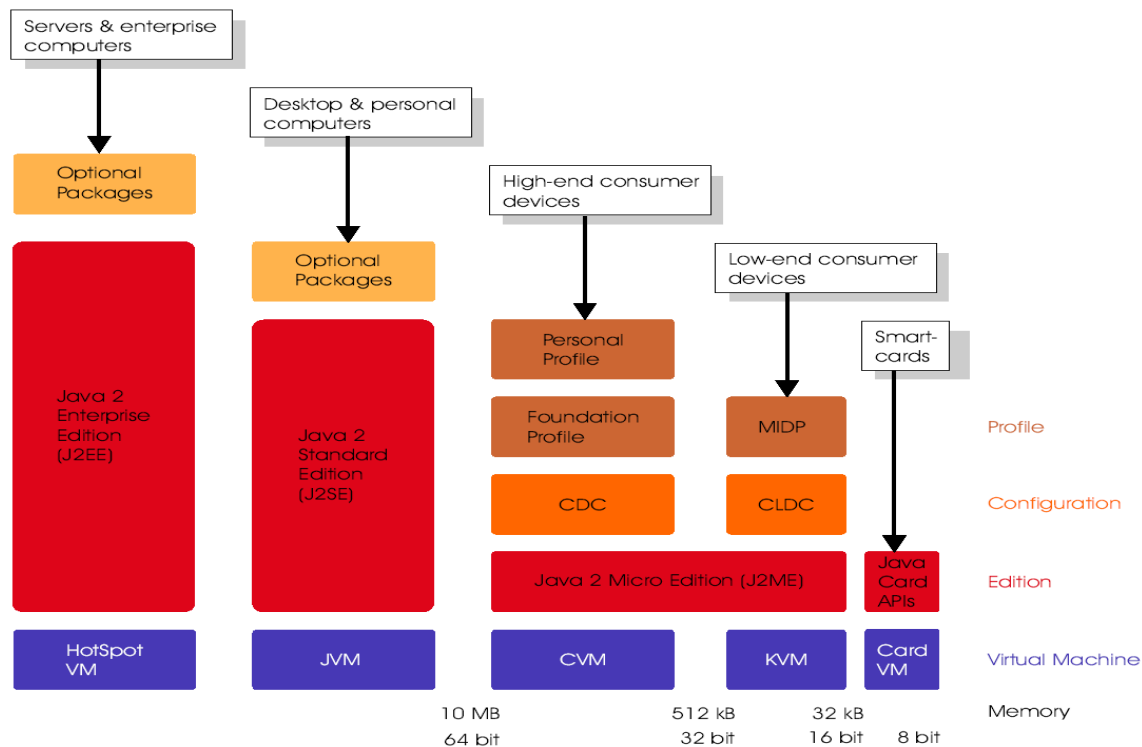


Abbildung 2: Schichten der Java 2 Plattform (Vgl. [FRIN02])

gemeinsamen Nenner für bestimmte Gerätegruppen. Die Profilschicht kann sich auch in mehrere Unterschichten aufteilen und es ist sogar möglich auf einer Konfigurationsschicht mehrere Profilschichten parallel zu verwenden. Der genauere Aufbau dieses Schichtenmodells wird in den folgenden Kapiteln von unten nach oben behandelt. Dabei wird sich an Abbildung 2 orientiert.

2 Virtuelle Maschinen

Die virtuelle Maschine ist Grundschrift einer jeden Java Implementation. Sie sorgt dafür, dass der vom Java-Compiler erzeugte Bytecode auf allen Rechnerarchitekturen und Betriebssystemen ausgeführt werden kann, für die eine virtuelle Maschine verfügbar ist. Für den mobilen Bereich genügte es nicht, nur eine virtuelle Maschine zu verwenden, die Geräte zu unterschiedlich sind. So gibt es Geräte im Mobilfunkbereich, die mit einigen hundert Kilobyte an Speicher auskommen müssen und nur wenige MHz an Rechenleistung besitzen, während im PDA-Bereich die Entwicklung mittlerweile so stark zugenommen hat, dass es bereits Geräte gibt, die sich, was den Speicher angeht zwischen 10 MB und 100 MB bewegen und bis 500 MHz Prozessorleistung beinhalten. Auf einigen Geräten ist es sogar möglich, die Java 2 Standard Edition auszuführen und zu nutzen. Das ist natürlich der Optimalfall, da dann keinerlei Portierungsarbeit mehr zwischen einer Java Desktop-Anwendung und einer gleichartigen mobilen Anwendung erfolgen muss. Für die anderen Fälle im mobilen Bereich gibt es jedoch zwei verschiedene virtuelle Maschinen. Zum einen gibt es die **C-Virtual Machine (CVM)**, die

hauptsächlich auf PDAs und leitungsgebundenen Kleingeräten, wie Sat-Receiver und Bildtelefone verwendet wird. Zum anderen gibt es die Kilo Virtual Machine (KVM), die im Bereich der Mobiltelefone Verwendung findet. Gemeinsam haben die beiden Maschinen, dass sie für den ressourcenschonenden Einsatz entwickelt worden sind.

2.1 CVM

Das „C“ in CVM steht für „Connected“. Das bezieht sich auf die ursprünglich geplante Verwendung dieser virtuellen Maschine für Geräte, die sowohl einen Netzanschluss besitzen, als auch einen Netzwerkanschluss, der über Kabel realisiert wird [SCHW01]. Dies ist ja bei den meisten PDAs auch der Fall, denn Sie besitzen in der Regel eine Kabelverbindung zum PC, um die Daten abzugleichen. Die CVM ähnelt der Standard Java Virtual Machine sehr stark, so dass sie in manchen Schaubildern auch weggelassen wird. Sie ist kompatibel zur JVM 1.3, jedoch nicht abwärtskompatibel. Das heißt, dass keine Klassen oder Methoden verwendet werden können, die in der Java Dokumentation als **deprecated** also „überholt“ gekennzeichnet sind. Zusätzlich besitzt sie einige Optimierungen. Sie kann im Gegensatz zur JVM den Java-Bytecode direkt aus dem ROM ausführen [MOLT02].

2.2 KVM

Im Gegensatz zur CVM hat die KVM sehr viele Einschränkungen und ist damit nicht kompatibel zur JVM. Allerdings kann die KVM bereits ab einem verfügbaren Speicher von 128 kB eingesetzt werden. Daher auch der Name Kilo Virtual Machine. 32 kB setzt die KVM dabei für die auf ihr laufenden Applikationen voraus und 96kB benötigt sie selbst. Sie umfasst nur 50-82kB Objektcode (hängt vom Zielgerät ab). Auch an Rechenleistung wird wenig vorausgesetzt. Auf einem 16 Bit Prozessor mit 25 MHz ist die KVM bereits ausführbar. 32 Bit Prozessoren ermöglichen eine verbesserte Performance [MOLT02]. Die KVM eignet sich daher sehr gut für die Verwendung in mobilen Kleinstgeräten, also Handys.

2.2.1 Einschränkungen der KVM

Damit ein vernünftiger Einsatz auf mobilen Kleinstgeräten möglich werden konnte, entschied sich SUN dafür, die KVM einigen drastischen Einschränkungen zu unterwerfen. Diese Einschränkungen sind [FRIN02]:

- **Keine Gleitkommaberechnungen:** Wegen der fehlenden Hardware-Unterstützung können die Datentypen `float` und `double`, sowie entsprechende Wrapperklassen nicht unterstützt werden. Jedoch kann eine softwareseitige Simulation über Festkomma erfolgen. Dazu wird das Paket `MathFP` benötigt.
- **Kein `finalize()`:** Die Superklasse `Object` enthält normalerweise diese Methode, die vom Garbage Collector aufgerufen wird. Um diesen zu vereinfachen, hat man sich dafür entschieden, `finalize()` nicht zu implementieren.
- **Eingeschränkte Fehlerbehandlung:** Es gibt im Wesentlichen nur zwei Klassen, die für die Fehlerbehandlung zuständig sind. Das sind: `java.lang.VirtualMachineError` und `java.lang.OutOfMemoryError`. Zum einen resultiert diese Einschränkung aus den Ressourcen-Beschränkungen, zum anderen soll vermieden werden, dass die Programmierer sich mit gerätespezifischen Fehlern befassen, um die Plattformunabhängigkeit zu gewährleisten.

- **Kein Java Native Interface (JNI):** Zum einen wird aus Sicherheitsgründen vorausgesetzt, dass der Satz an nativen Funktionen fest ist. Zum anderen wäre eine vollständige Umsetzung des JNI ohnehin zu Speicherintensiv.
- **Kein Nutzer-definierter Class Loader:** Der eingesetzte Class Loader kann aus Gründen der Sicherheit nicht geändert werden.
- **Kein Reflection:** Reflection bildet die Basis für **Remote Method Invocation (RMI)** oder für die **Serialisierung** von Objekten. Außerdem wird es für das **JVM Debugging Interface (JVMDI)** und **Profile Interface (JVMPI)** verwendet. Auch diese Streichung erfolgte aus Platzmangel.
- **Keine Thread Groups und Daemon Threads:** Multithreading ist allerdings implementiert. Jedoch müssen Operationen auf Threadgruppen selbst über explizite Collections realisiert werden.
- **Keine lose Kopplung:** Die KVM sieht keine Möglichkeit vor, ein Programm lose an ein Objekt zu kopplen.

3 Konfigurationen

In Abbildung 2 erkennt man, dass über der VM-Schicht die Editionsschicht liegt. Wie bereits in Abschnitt 1.2 erklärt wurde, enthält diese Schicht die gleichen APIs, die auch in den beiden anderen Editionen vorhanden sind. Für die Editionsschicht ist daher kein extra Kapitel veranschlagt. Auf die Editionsschicht setzt die Konfigurationsschicht auf. Alles was oberhalb der Editionsschicht liegt, ist nicht mehr Schnittmenge zur J2SE. Was aber ist genau eine Konfiguration? Dies lässt sich am kürzesten mit einem Zitat aus [MM03] sagen:

„Eine Konfiguration umfasst also die Menge aller implementierten Bibliotheken und Funktionen der virtuellen Maschine und zielt auf eine bestimmte Gerätegruppe. Sie gliedert die Geräte aber weniger nach Funktion, sondern mehr nach technischen Gegebenheiten, wie z.B. Speicher, Bandbreite oder Rechenleistung.“

Aus diesem Grund gibt es äquivalent zu den beiden im vorherigen Kapitel besprochenen virtuellen Maschinen in der Konfigurationsschicht zwei unterschiedliche Konfigurationen. Auf die CVM baut die **Connected Device Configuration (CDC)** auf. Die **Connected Limited Device Configuration (CLDC)** liegt oberhalb der KVM und wird daher für mobile Kleinstgeräte verwendet.

3.1 CDC

Die CDC ist etwas mächtiger, als die CLDC, da sie für High-end Geräte entwickelt wurde. Sie benötigt daher auch 2-4 MB Speicher und ist deshalb für mobile Kleinstgeräte uninteressant. Außerdem ist sie im Gegensatz zur CLDC für dauerhafte Netzwerkverbindungen mit hoher Bandbreite ausgelegt. Sie verwendet daher auch TCP/IP. Da die CDC eine Superklasse der CLDC ist, ist diese, ausreichende Ressourcen einmal vorausgesetzt, auch auf der KVM einsetzbar. Dies wäre jedoch riskant, falls Funktionen aufgerufen würden, die die virtuelle Maschine nicht unterstützt. Umgekehrt kann die CLDC natürlich problemlos auf der CVM Verwendung finden.

3.2 CLDC

Die CLD Konfiguration baut auf die KVM auf und wurde daher auch für die Verwendung auf Mobiltelefonen, Pager und PALM-Handhelds entwickelt. Die Konfiguration bildet eine Unterklasse

Art	Minimum	Maximum	Bemerkung
Speicherbedarf, gesamt	160 kB	512 kB	für die Java Plattform
persistenter Speicher	128 kB		JVM und CLDC Bibliotheken
Arbeitsspeicher	32 kB		Java Runtime und Heap
Prozessor	16 bit	32 bit	
Geschwindigkeit	25 MHz		
Energieversorgung	Batterie		
Netzwerk-Verbindung		9600 bps	flüchtig, drahtlos

Abbildung 3: Hardwareempfehlungen für CLDC (vgl. [FRIN02])

zur CDC und ist genau wie die KVM für sehr geringe Ressourcen optimiert. Abbildung 3 zeigt, ab wann eine sinnvolle Anwendung gewährleistet werden kann. Die Prozessorvoraussetzungen entsprechen dabei Sinnvollerweise denen für die KVM. Wurden für die KVM nur 128kB an Speicher benötigt, so muss dieser bei Verwendung von CLDC nur um ein Geringes größer sein. Ab 160 kB Gesamtspeicher kann man bereits arbeiten. Maximal können 512 kB verwaltet werden. Im Gegensatz zur Fähigkeit der CDC, dauerhafte Netzwerkverbindungen zu ermöglichen, ist in CLDC nur der Einsatz von flüchtigen Verbindungen mit geringer Bandbreite vorgesehen.

3.3 Direkter Vergleich

Als Abschluss dieses Kapitels erfolgt noch ein direkter Vergleich zwischen den beiden vorher behandelten Konfigurationen.

CDC

Ausgerichtet auf Geräte mit graphischer Oberfläche

Speicher: 2-16 MB
Mind.: 512 kB ROM, 256 kB RAM

32 Bit Prozessor

Komplette Implementierung der JVM (CVM)

Hohe Bandbreite für Datenkommunikation

Permanente Netzwerkverbindung unterstützt

CLDC

Nur minimale graphische Oberfläche

Speicher: 160-512 kB
Mind.: 32 kB RAM für Laufzeitobjekte

16 oder 32 Bit Prozessor

Beschränkte JVM (z.B. KVM)

Geringe Bandbreite für Datenkommunikation

Keine permanente Netzwerkverbindung

Ausgelegt für Geräte mit geringem Stromverbrauch
Einsatz in Massenartikeln

4 Profile

Die Profilschicht setzt direkt auf die Konfigurationsschicht auf. Es gibt spezielle Profile für die verschiedenen Anwendungsbereiche, die für den entsprechenden Bereich spezielle APIs enthalten. Dabei können auch mehrere Profile nebeneinander oder übereinander verwendet werden. Im Folgenden werden die Eigenschaften der Profile für das **Mobile Information Device Profile (MIDP)** in der Version 1.0 und der neuen Version 2.0, sowie dem Profil für PDAs vorgestellt.

4.2 Das MID Profil

Das MID-Profil wurde entwickelt, um die wichtigste Funktionalität für mobile Endgeräte mit besonders kleiner Speicherkapazität zu liefern. Es setzt auf der CLDC Schicht auf und ist der kleinste gemeinsame Nenner mobiler Endgeräte.

4.2.1 MIDP 1.0

Die Spezifikation MIDP in der Version 1.0 wurde bereits vor mehreren Jahren implementiert und ist in der Version 1.0.4_01 des J2ME Wireless Toolkit von Java verfügbar. Es stellt APIs auf den folgenden Gebieten zu Verfügung:

- Anwendungen definieren und steuern
- Text und Graphik darstellen
- Eingaben verarbeiten
- Daten in einfache Datenbanken ablegen
- Herstellen einer Netzwerkverbindung über eine HTTP-Untermenge
- Verwenden von timergesteuerten Ereignissen.

Diese eigentlich sehr grundsätzlichen und geringen Profil-APIs reichten den Entwicklern allerdings schon sehr schnell nicht mehr aus und so wurde bereits kurz nach Veröffentlichung an einer Erweiterung der Spezifikation gearbeitet.

4.2.1 MIDP 2.0

Da die MIDP 1.0 Spezifikation weitgehend auf Standard Sicherheitsfunktionen verzichtete, mussten die Entwickler diese selbst erzeugen und zu jedem Programmpaket hinzufügen, was den ohnehin spärlichen Speicherplatz auf den mobilen Geräten zusätzlich verringerte. MIDP 2.0 wurde daher mit einer eigenen Sicherheitsapi ausgerüstet, die in den Bereichen liegt, die zuvor oft selbst implementiert werden mussten. Spezifiziert wurde das **WAP Certificate Profile (WAPCERT)**, welches auf der **Internet X.509 Public Key Infrastructure (PKI)** und der **Certificate Revocation List (CRL)** basiert. Dadurch können in Zukunft leicht Zertifikate erstellt und sichere HTTPS Verbindungen erzeugt werden. Auch im Bereich der Netzwerktauglichkeit war MIDP 1.0 sehr eingeschränkt. MIDP 2.0 unterstützt neben dem in Version 1 bereits vorhandenen HTTP-Protokoll noch weitere. Dies sind:

HTTPS
SSL/TLS } Notwendig für das versenden Sicherheitsrelevanter Daten.

TCP / IP Sockets
UDP / IP Datagramme } für Netzwerkfähigkeit auf niedrigerer Ebene für Kabelverbindungen.

Während diese neuen Verbindungstypen sicher nützlich und wichtig sind, ist als wirkliche Neuerung vor allem die **PushRegistry** Klasse zu nennen. In dieser Klasse kann ein **MIDlet (MIDP Anwendung)** einen Lauscher installieren, der später das nicht laufende MIDlet startet, sobald es über das Netz angefordert wird. Auf diese Weise kann ein MIDlet Dienste bereitstellen, während es nicht läuft. So bleibt der geringe Speicherplatz für andere Anwendungen frei.

Außerdem unterstützt die 2.0 Spezifikation auch einfache Audio und Video Ausgaben, die bisher eher PDA Profilen vorbehalten waren. Diese **Media API** kann so für hochqualitative Klingeltöne eingesetzt werden. Außerdem wird sie für eine weitere API benötigt: Die **Gaming API**.

Die später zur MIDP 1.0 hinzugefügte **Over The Air Funktionalität OTA** wurde ebenfalls integriert. Damit ist es möglich, MIDlets ohne Anschluss an einen Rechner zu beziehen. Schließlich wurden noch einige Schnittstellen erweitert. MIDP 2.0 befindet sich derzeit in der Beta Phase und ist von Sun zu beziehen [MIDP].

4.2 Das PDA Profil

Dieses Profil wird in Abbildung 2 nicht dargestellt. Es enthält das MID Profil vollständig und könnte so parallel zum MIDP gezeichnet werden. Das PDA Profil wurde für Geräte entwickelt, die einen größeren Speicher beinhalten und auch eine höhere graphische Darstellung ermöglichen. Ebenso wie das MID Profil setzt PDAP auf die CLDC Schicht auf. Voraussetzung für dieses Profil sind mindestens 512 kB Speicher, die entweder als RAM oder aus einer RAM/ROM Kombination zur Verfügung stehen müssen. Die Auflösung muss dabei mindestens 20.000 Pixel betragen. Außerdem muss ein Zeigergerät zur Verfügung stehen und eine Möglichkeit, Buchstaben einzugeben. Dies kann z.B. über eine virtuelle Tastatur realisiert werden. Es soll außerdem auch eine Untermenge des Java **Abstract Window Toolkit (ATW)** unterstützt werden. Das PDA Profil ist bisher allerdings nur spezifiziert. Eine Implementierung hat von Sun noch nicht stattgefunden.

4.3 Das Foundation Profil

Wie man in Abbildung 2 sehen kann, baut das Foundation Profil auf CDC auf. Es ist nicht vergleichbar mit den bisher behandelten Profilen, denn es stellt ein Basisprofil dar, auf das weitere Profile aufbauen können. Daher stellt es auch keine Klassen- oder Benutzerschnittstelle zur Verfügung. Enthalten in diesem Profil sind Pakete, die man bereits aus der Standard Edition kennt:

- `java-lang`
- `java.util`
- `java.net`
- `java.io`
- `java.text`
- `java.security`

Natürlich erhöht das Profil auch die Hardwareanforderungen. Diese sind nun auf 1024 kB persistenten Speicher und 512 kB Arbeitsspeicher gestiegen.

4.4 Das Personal Profil

Das Personal Profil baut auf das Foundation Profil auf und ist nicht parallel zu diesem zu sehen. Das Personal Profile ist dazu da, Personal Java auf der Basis von J2ME verfügbar zu machen. Es bietet bereits das vollständige AWT Paket und ist vor allem auf Web-Anwendungen im Applet-Bereich ausgerichtet. Verfügbar sind folgende Pakete:

- `java.applet`
- `java.awt` (incl. der wichtigsten Unterpakete)
- `java.beans`
- `javax.microedition.xlet`

Auch diese neue Unterschicht erhöht natürlich erneut den Ressourcenbedarf. Diesmal allerdings signifikant. Es werden nun 2,5 MB persistenter und 1 MB flüchtiger Speicher benötigt.

Es gibt noch mehrere Profile und es werden in Zukunft sicher auch noch die verschiedensten hinzukommen. Die im letzten Abschnitt besprochenen sind aber die Grundprofile und werden immer dann benötigt, wenn man seine Applikation so schreiben will, dass Plattformunabhängigkeit gewährleistet ist. Weitere Profile können natürlich auch auf das Foundation Profil aufsetzen. So gibt es z.B. noch das RMI Profil, das die gleichen Hardwareanforderungen stellt, wie das Personal Profil. Abbildung 4 zeigt noch einmal den Aufbau der Konfigurations- und Profilschicht und stellt einige mögliche zusätzliche Profile vor.

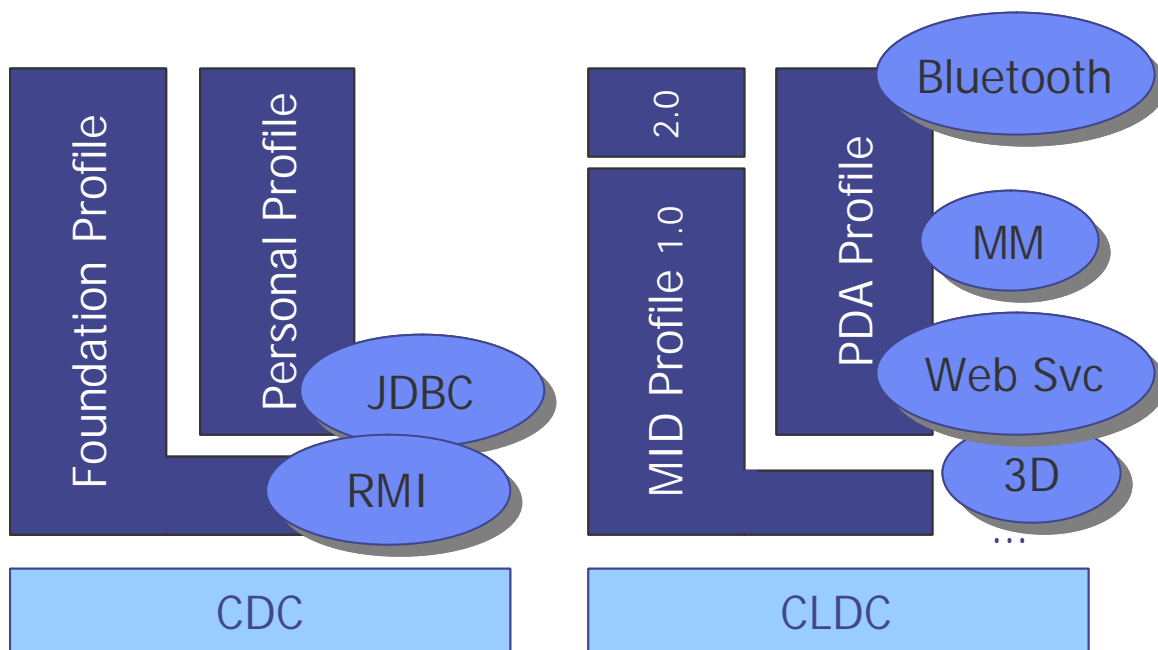


Abbildung 4: Modularer Aufbau der Profilschicht

Als Nachteil dieses modalen Aufbaus muss man natürlich den Umstand nennen, dass Software, die spezielle Profile verwendet nicht mehr wirklich plattformunabhängig ist und nur auf den Umgebungen läuft, auf denen die entsprechenden Profile verfügbar sind.

Von denen in Abbildung 4 gezeigten optionalen Profilen ist die Spezifikation bereits abgeschlossen. Die Implementation lässt bei vielen jedoch noch auf sich warten. So ist es z.B. außerordentlich schwer, eine funktionierende Bluetooth API zu finden, da SUN selbst sich bei der Implementation von Zusatzprofilen scheinbar sehr zurück hält, obwohl die Spezifikation für ein Bluetooth Profil bereits im März 2002 abgeschlossen wurde.

5 Alternative Implementierungen

In der Ausarbeitung wurde vor allem auf die Java 2 Micro Edition eingegangen, da diese von SUN weiterentwickelt wird und die wohl derzeit populärste Implementierung ist. Das Wireless Toolkit von SUN bietet dabei auch einige komfortable Tools, wie Emulatoren von verschiedenen mobilen Geräten, wie Handys oder PDAs. Es gibt jedoch auch andere Ansätze für kabelloses Java, die sich teilweise an die von SUN festgelegten Spezifikationen halten, teilweise aber auch eigene Ziele verfolgen. Die bekanntesten werden hier kurz vorgestellt.

5.1 Personal Java

Personal Java wurde auch von SUN entwickelt. Es ist sozusagen der Vorgänger der Micro Edition. Personal Java verwendet die Java 1 Plattform und wird scheinbar auch nicht mehr weiterentwickelt. Dabei wird weitestgehend das JDK 1.8 eingesetzt. Das macht es kompatibel zu CDC, so dass es heute noch für die CVM verwendet werden kann. Im „Low-cost mobile“ Bereich findet es daher allerdings keine Verwendung.

5.2 Waba

Die Sprache Waba von der Firma Wabasoft ist genau genommen kein Java, denn es verwendet gänzlich andere Basisklassen. Durch die auf Palm-OS und Windows CE angepasste virtuelle Maschine wird natürlich auch die Portabilität eingeschränkt. Dieser Umstand wird jedoch durch die enorme Geschwindigkeit wieder wett gemacht, an die keine andere Implementierung heranreicht. Außerdem stellt Waba viel mehr GUI Komponenten zur Verfügung, als in MIDP vorhanden sind. Waba wird als Open Source Produkt vertrieben.

5.3 J9

J9 ist die virtuelle Maschine von IBM. Für die Entwicklung von Java Applikationen für mobile Geräte stellt IBM auch gleich eine Entwicklungsumgebung zur Verfügung. Diese heißt **Visual Age Micro Edition (VAME)** und verfügt über eine komfortable IDE.

5.4 Jeode

Jeode von der Firma Insignia ist eine weitere Alternative. Diese Firma stellt gleich mehrere VMs her, von denen jede ein bestimmtes Publikum ansprechen soll. So gibt es bspw. eine **PDA Edition** und eine **Mobile Foundation Edition**. Diese Implementierungen sind kompatibel zu Personal Java und richten sich daher vor allem an Benutzer von PDAs. Die Jeode VMs können allerdings nicht kostenlos heruntergeladen werden.

5.6 SavaJe

SavaJe bietet eine weitere virtuelle Maschine für den Pocket PC Bereich unter Windows CE an. Diese Modelle verwenden die **Advanced Risc Machine (ARM)** Prozessoren. Implementierungen gibt es für das Personal Profile 1.0 und Mobile Information Device Profile (MIDP). Von SavaJe gibt es auch ein komplettes Betriebssystem in Java: SavaJe XP.

6 Smartcards

Die Idee einer durch einen VLSI Chip gesteuerten Karte wurde bereits 1968 in Deutschland patentiert. Durch die rasante Hardwareentwicklung wurden ab Mitte der siebziger Jahre schließlich Testkarten hergestellt und in den achtziger Jahren erkannten die Banken deren Nutzen. Gegen Ende der neunziger Jahre schließlich hat sich immer mehr gezeigt, dass Java wohl zur Steuersprache für Smartcards werden würde, da man eine einfache und sichere Ausführungsplattform benötigte. Der aktuelle JavaCard 2.0 Standard definiert die Grundfunktionalität einer Chipkarte im so genannten **JavaCard Runtime Environment (JCRC)**. Diese Umgebung besteht aus der **JavaCard Virtual Machine (CardVM)** und der **JavaCard API**, die im Folgenden erläutert werden.

6.1 Card VM

Da Smartcards im allgemeinen typische Speicherkonfigurationen von 1 kB RAM, 16 kB EEPROM und 24 kB ROM haben, ist die Unterbringung der JVM eine der größten Herausforderungen was das Design von Smartcards angeht. Daher musste die CardVM noch stärker abgespeckt werden, als die virtuellen Maschinen im Mobilfunkbereich. Nicht unterstützt werden daher folgende Eigenschaften:

- Die Datentypen: long, double, float
- Characters und Strings
- Mehrdimensionale Felder
- Dynamisches Nachladen von Klassen
- Sicherheits-Manager
- Garbage Collection (eingeschränkt)
- Multithreading
- Objektserialisierung
- Klonen von Objekten

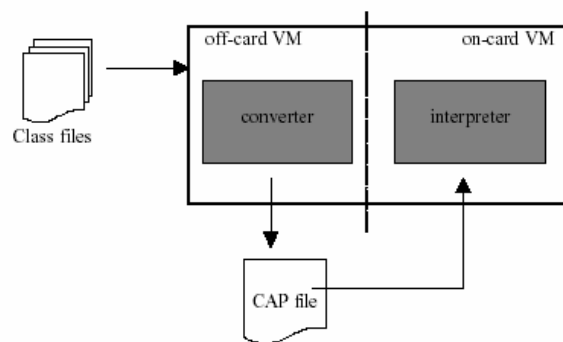


Abbildung 5: Zweiteilung einer Card VM

Eine vollständige Implementierung der VM ist derzeit auf einer Smartcard dennoch nicht möglich. Daher findet noch eine Zweiteilung statt. Der Bytecodeinterpreter befindet sich nach wie vor in der VM auf der Smartcard. Der zweite Teil, ein Konverter befindet sich außerhalb der Karte auf einem PC. Dieser wandelt Java Class-Files in CAP-Files um, die anschließend auf die Smartcard übertragen und dort interpretiert werden können. SUN arbeitet derzeit an einem Card-Prozessor, der direkt die CardVM enthält. Ein solcher Prozessor würde natürlich Geschwindigkeits- und Speicherprobleme beheben.

6.2 JavaCard API

Das eigentliche Kernstück der Laufzeitumgebung ist die JavaCard API. Diese Schicht beinhaltet die Systemklassen und besteht aus den drei Komponenten **Framework-Klassen**, **Erweiterungen** und **Installer**. Dabei enthalten die Framework-Klassen die grundlegenden Java-Pakete, wie `java.lang`, `javacard.framework`, `javacard.security` und `javacardx.crypto`. Unter Erweiterungen versteht man anwendungsspezifische Bibliotheken. Der Installer ermöglicht das Ablegen von Cardlets (das sind die JavaCard Anwendungen) auf der Smartcard [THEU01].

7 Fazit

Die Idee von Java war es ja eigentlich, eine Möglichkeit zu schaffen, die eine plattformunabhängige Implementierung von Software bereitstellte. Diese Idee hat sich auch im Bereich von Heimanwendungen, im Firmenbereich und sogar im Bereich der Softwareentwicklungswerkzeuge durchgesetzt. Viele Firmen bieten heute sogar Entwicklungsumgebungen an, die komplett in Java geschrieben und daher theoretisch unverändert, praktisch mit wenig Aufwand auf den verschiedensten Rechnerarchitekturen ausführbar sind.

Im „Wireless“ Bereich ist diese Idee meiner Meinung nach gänzlich verworfen worden. Es ist heute praktisch nicht möglich, eine Anwendung für ein bestimmtes mobiles Endgerät zu entwickeln und dabei davon auszugehen, dass diese auch von anderen Geräten ausgeführt werden kann. Es sei denn die Hersteller verwenden ausschließlich die Kombination von CLDC und MIDP. Alles Weitere würde nur zu Problemen führen. Da allerdings jeder Hersteller etwas Besonderes bieten will, macht er sein Gerät fähig, die verschiedensten Zusatzprofile zu verwenden. Und so kommt es dann, dass in Kauf genommen wird, dass ein MIDlet eines Gerätes schon auf der Vorgängerversion des gleichen Gerätes nicht mehr ausführbar ist, nur um etwa über eine 3D API ein sich drehendes Symbol anzuzeigen. Vielleicht hält sich Sun ja aus diesem Grund mit der Entwicklung von Profilen derart zurück. Der Grund, warum Java auch eingesetzt wird, ohne die Plattformunabhängigkeit zu Verfügung zu stellen, liegt meiner Meinung nach daran, dass sich „Java“ als Schlagwort zum einen gut verkaufen lässt, zum anderen eine Entwicklungsplattform zur Verfügung gestellt wird, die einfach und sicher ist.

Die echte Plattformunabhängigkeit wird jedoch in den Mobilfunkbereich erst dann Einzug finden, wenn die Hardware der Geräte es zulässt, die gleiche Edition zu verwenden, die auch im nicht mobilen Bereich Anwendung findet. Einige PDAs sind bereits heute so stark, dass man eine ganz normale J2SE auf ihnen ausführen kann.

Was den Bereich der Smartcards angeht, so ist Java dort weiterhin auf dem Vormarsch. Das zeigen die aktuellen Pressemeldungen. Erst im Januar hat Sharp eine neue auf Java basierende Smartcard herausgebracht, die über 1 MB Speicher verfügt. Im Februar gab es eine Nachricht, dass sich das WLAN SmartCard Consortium formiert hat.

Tim Pommerening

A Literatur

Bücher und Internetseiten

- [FRIN02] ***Mobile Computing – Betriebssysteme und Entwicklungsumgebungen***
Gabriele Frings, 12. Juni 2002 (Seminararbeit)
URL: <http://www.4friendsonly.org/papers/MobileComputing.pdf>
- [MIDP] ***Sun Java MIDP***
Informationen und Downloads zu MIDP 1.0 und MIDP 2.0
URL: <http://java.sun.com/products/midp>
- [MM03] ***Programmierung mobiler Endgeräte mit Java 2 Micro Edition (J2ME)***
Martin Müller, 2003 Version 0.2.3
Email: Martin@windgaesser.de
URL: <http://www.windgaesser.de/>
- [MOLT02] ***Betriebssysteme und Java-Programmierung auf dem iPAQ***
Michael Moltenbrey: Seminararbeit 2002
Email: moltenml@rupert.informatik.uni-stuttgart.de
URL: http://www.informatik.uni-stuttgart.de/ipvr/as/lehre/seminar/docws02/Betriebssysteme_Java_iPAQ_Ausarb_Michael.pdf
- [SCHW01] ***Java für eingebette Systeme***
Jens Schwarz, Seminar: Objektorientierter Systementwurf SS2001
Email: Jens.Schwarz@student.uni-tuebingen.de
URL: <http://www-ti.informatik.uni-tuebingen.de/deutsch/lehre/seminar/SoSe2001/>
- [THEU01] ***Java, Embedded Systems, and Mobile Commerce***
Han Jon Theus, Markus Leu, Seminararbeit
URL: <http://www.ifi.unizh.ch/~riedl/lectures/Theus.pdf>

B Abbildungen

ABBILDUNG 1: DIE JAVA 2 PLATTFORM (VGL. [MM03])	3
ABBILDUNG 2: SCHICHTEN DER JAVA 2 PLATTFORM (VGL. [FRIN02])	4
ABBILDUNG 3: HARDWAREEMPFEHLUNGEN FÜR CLDC (VGL. [FRIN02])	7
ABBILDUNG 4: MODULARER AUFBAU DER PROFILSCHICHT	10
ABBILDUNG 5: ZWEITEILUNG EINER CARD VM	12