Seminararbeit
„verteilte und parallele Systeme I“
6. Fachsemester
SS2002

zum Thema

Java 2 Micro Edition (J2ME)

von

D. Hellmuth
H. Mohr

15. Juni 2002

Inhaltsverzeichnis

1. Einleitung

Auf der JavaOne99, der weltweit größten Java Entwickler Konferenz, die alljährlich in San Francisco stattfindet, wurde von SUN Microsystems Inc. erstmals die Neugliederung der Java Technologie vorgestellt. Diese Neugliederung beinhaltet die drei Sparten Java 2 Micro Edition (J2ME), Java 2 Standard Edition (J2SE) und Java 2 Enterprise Edition (J2EE).

Java 2 Enterprise Edition (J2EE):

Die J2EE ist besonders für die Unternehmen ausgelegt, die ihren Kunden und Lieferanten solide und skalierbare Internetbusiness-Server-Lösungen zur Verfügung stellen wollen.

Java 2 Standard Edition (J2SE):

Die J2SE bezieht sich mehr auf den derzeit etablierten Desktopmarkt. Damit sollen Applikationen und Applets entwickelt werden, welche auf dem Client verwendet werden. Stichworte: Swing, Java2D, Java Sound

Java 2 Micro Edtion (J2ME):

Der Einsatzbereich der Micro Edition liegt vorwiegend im Bereich der mobilen Applikationen und der Möglichkeit, Java Applikationen auch mit beschränkten Ressourcen zu nutzen. Die J2ME bietet Unterstützung für viele Konsumgüter und embedded systems, wie z.B. Mobiltelefone, PDA's, Internetbildtelefone, digitale Settopboxen, Unterhaltungselektronik, Navigationssysteme und Netzwerkswitches.

Bisher hat beinahe jeder Hersteller von Geräten mit beschränkten Ressourcen eine eigene Entwicklungsplattform für die Applikationen benutzt. Durch die Verwendung der Micro Edition besteht nun endlich die Möglichkeit, Produkte schneller und plattformübergreifend zu entwickeln, wie das schon auf den Desktop gebräuchlich ist. Weiter fällt auch das Erlernen einer neuen Programmiersprache weg, da die Applikationen von J2ME aufwärtskompatibel sind.

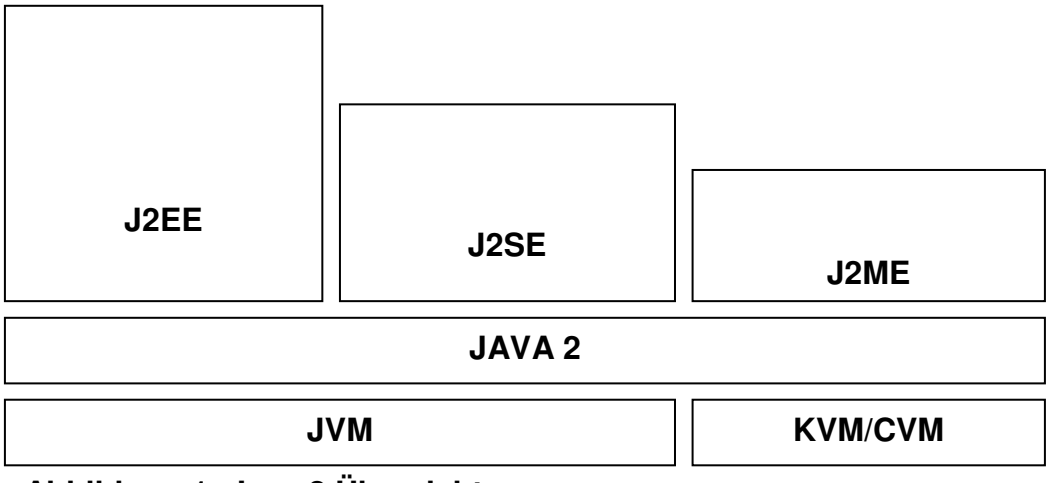

Abbildung 1: Java 2 Übersicht

2. Aufbau der Java 2 Micro Edition:

Der Aufbau der J2ME lässt sich in vier Schichten aufteilen:

1. die unterste Ebene bildet die Betriebssystemschicht (wie z.B. Linux, Windows32, Solaris, PalmOS oder WinCE)
2. auf der Betriebssystemschicht setzt eine plattformabhängige Implementierung der virtuellen Maschine auf, die KVM (Kilobyte Virtual Machine) oder die CVM (Connected Virtual Machine)
3. die dritte Ebene ist die Konfigurationsschicht
4. die Profilschicht setzt auf den Konfigurationen auf und bietet gerätespezifische Funktionen, welche z.B. die CLDC Konfiguration um GUI Klassen und Mechanismen für die persistente Datenhaltung erweitern.

Die Schichten drei und vier werden zu einer API (Satz von Bibliotheken – die allerdings gegenüber dem Original mächtig abgespeckt ist) zusammengefasst.

Derzeit sind für die J2ME zwei Konfigurationen verfügbar. Zum einen die Connected Limited Device Configuration (CLDC) die unter der JSR (Java Specification Request) 36 und die Connected Device Configuration (CDC), die unter JSR 30 im Zuge des Java Community Process (siehe Anhang) spezifiziert wurde. Die CLDC ist schon seit Juni 2000 verfügbar und eine Referenzimplementierung für PalmOS. Für die CDC und die Foundation Profil gibt es derzeit durchaus eine Referenzimplementierung, diese sind allerdings nicht für mobile Geräte verfügbar und werden daher nur zur Vollständigkeit in Abbildung 2 aufgenommen. Auf die bereits erwähnten Konfigurationen setzen die sogenannten Profile auf.

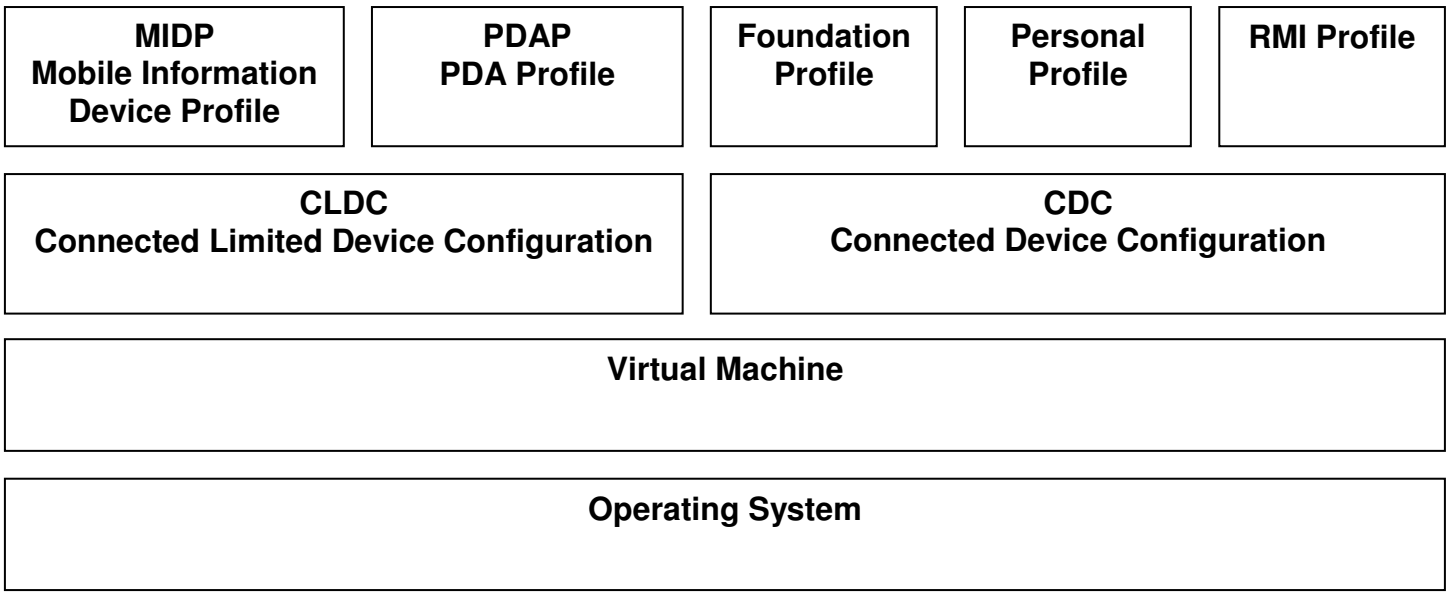

Abbildung 2: J2ME Aufbau

Die J2ME ist nicht wie Jcontrol oder JavaCard (siehe Anhang) für 8bit-Systeme verwendbar, da die Ressourcenanforderungen diese Systeme überfordern würde. Die Virtuelle Maschine kann auf Speicher von 128 KB bis 512 KB zugreifen und mit einem 16- oder 32 Bit-Prozessor arbeiten. Der Programmcode der virtuellen Maschine ist selbst nur 40 bis 80 KB groß. Die Datenanbindung ist mit 9600 Baud spezifiziert.

Die Spezifikation ist wie bei den meisten „Java-Grundlagen" sehr abstrakt und flexibel gehalten, um eine möglichst große Anpassung zu ermöglichen. Als Anwendung dieser Plattform sind z.B. kostenpflichtige Applikationen gedacht, die ein Handybesitzer auf sein Telefon herunterladen und starten kann. Es sind auch grafikintensive Anwendungen wie Netzbasierte Navigationssysteme vorgesehen. Für den Entwickler stehen Pakete wie „kAWT“ zur Entwicklung von Grafikapplikationen zur Verfügung. Schließlich wurde von den Entwicklern auch ein „Preverifier“ entwickelt, mit dem sich die Applikation überprüfen und verkleinern lässt.

3. Konfigurationen

Konfigurationen beschreiben die verfügbaren Grundfunktionen der virtuellen Maschine für eine Klasse von Geräten mit ähnlicher Leistungsfähigkeit. Dies schließt sowohl die Low-Level Klassen für Ein- und Ausgabe ein, als auch den Java Sprachumfang, etwa ob Fließkommazahlen unterstützt werden. Eine Konfiguration umfasst eine virtuelle Maschine, Klassenbibliotheken und API's, die auf die Eigenschaften dieser Geräte zugeschnitten sind.

Derzeit existieren zwei dieser Konfigurationen, welche aufgrund Ihrer Anforderungen an die Zielplattform sehr unterschiedliche Untermengen der J2SE Klassen unterstützen. Die Connected Limited Device Configuration für Mobiltelefone, Pager und PDAs und die Connected Device Configuration (CDC) für etwas leistungsfähigere Geräte, wie Settopboxen, Bildtelefone und Spielkonsolen.

3.1 Connected Limited Device Configuration (CLDC)

Die CLDC-Konfiguration ist für kleinere Geräte wie Handys, einfache PDAs, Pager, Mobiltelefone definiert. Sie verfügen über einen Speicher von 128 - 512 KB. Sie können Netzverbindungen mit maximal 9600 Baud betreiben und arbeiten in der Regel mit Batterie. Das CLCD nutzt eine eigene virtuelle Maschine, die KVM genannt wird. Für den Palm-Pilot ist zum Beispiel eine Implementierung der KVM verfügbar, ebenfalls aber für Mobiltelefone. Sie ist nicht für 8 Bit-Systeme gedacht. Diesen Bereich deckt die JavaCard besser ab. Hersteller wie IBM haben eigene virtuelle Maschinen im Angebot, die oft schneller in der Ausführung sind.

Die Pakete der J2ME sind eine Teilmenge der J2SE, insbesondere:

- java.lang (VM-Systemklassen)
- java.io (Datei Ein-/Ausgabe)
- java.util (Datenstrukturen)
- java.net (UDP-Datagramme und Datei-Ein-/Ausgabe)
- java.text (Minimale Funktionalität für Internationalisierung, speziell Fehlermeldungen)
- java.security (Minimale Funktionalität, Verschlüsselung für serialisierte Objekte)

Für Mobiltelefone existiert bereits das Mobile Information Device Profil (MIDP), dass GUI-Klassen wie auch Datenbanklassen für die persistente Datenhaltung auf Mobiltelefonen beinhaltet. Da die CLDC-KVM für Geräte spezifiziert wurde, deren Prozessoren möglicherweise keine Fließpunktarithmetik unterstützen, stehen die Datentypen float und double nicht zur Verfügung. Eine Lösung für dieses Problem stellt zum Beispiel die Bibliothek MathFP dar, die Festkomma-Arithmetik auf Basis des 32-Bit Integer-Datentyps der KVM zur Verfügung stellt. Außerdem fehlt der KVM aus Sicherheitsgründen die Unterstützung für das Java Native Interface (JNI) wodurch dem Entwickler der Zugriff auf Betriebsystemaufrufe verwährt bleibt. Darüber hinaus ist die Reflection-API stark eingeschränkt, wie auch die Objekt Finalisation, so das der Java Garbage Collector nicht unterstützt wird.

Eine weitere Einschränkung der KVM in der CLDC Konfiguration ist, dass die KVM keine Mechanismen zur Klassenverifizierung beinhaltet. Klassendateien, die in der KVM ausgeführt werden sollen, müssen vorab auf dem Desktop geprüft werden. Dieser sogenannte „Preverify“ -Schritt fügt Hinweise in die Klassendatei ein, die dem KVM-Classloader die Prüfung der Klassen erleichtert und diese aufwendige Arbeit auf Mobilen Geräten beschleunigen soll.

Für die Entwicklung einer Anwendung für die Connected Limited Device Configuration ist wieder ein auf dieser Konfiguration aufbauendes Profil nötig. Da die CLDC eine Teilmenge der CDC darstellt, können CLDC-Anwendungen auf CDC-Geräten laufen, solange dort das entsprechende Profil vorliegt.

3.2 Connected Device Configuration (CDC)

CDC zielt auf leistungsfähigere Geräte wie z. B. Webpads, Bildtelefone, Spielkonsolen und PDAs, mit einer Leistungsfähigkeit verglichen mit den Pocket PCs, ab. Randdaten, welche diese Plattformen unterstützen müssen, sind Systemspeicher von mindestens 256kB RAM und 512kB ROM sowie Netzwerkverbindungsmöglichkeiten mit mehr als 9600Bps.

Die CDC (Connected Device Configuration) beinhaltet eine komplette JVM, die Java 2 CVM (Connected Virtual Machine) und die minimal erforderlichen API Klassen für das System. Für Applikationen wird zusätzlich ein Profil benötigt, dass „Foundation Profile“. Dies beinhaltet alle übrigen Klassen und APIs. Bei Bedarf (z.B. für GUI) müssen noch zusätzliche Profile verwendet werden.

3.3 Vergleich CDC – CLDC

CDC ist einer Superklasse von CLDC und ist somit auch für CLDC Geräte einsetzbar. Alle Geräte, welche Personal Java unterstützen, können leicht auf CDC portiert werden. Zu bemerken ist aber, dass es keine feste Grenze gibt zwischen den Geräten der CLDC und CDC Klasse. An Abbildung 3 ist es klar ersichtlich, dass die CDC und CLDC nicht ausschließlich aus J2ME Klassen bestehen müssen, sondern dass die jeweilige Applikation um eigene Klassen erweitert werden kann.

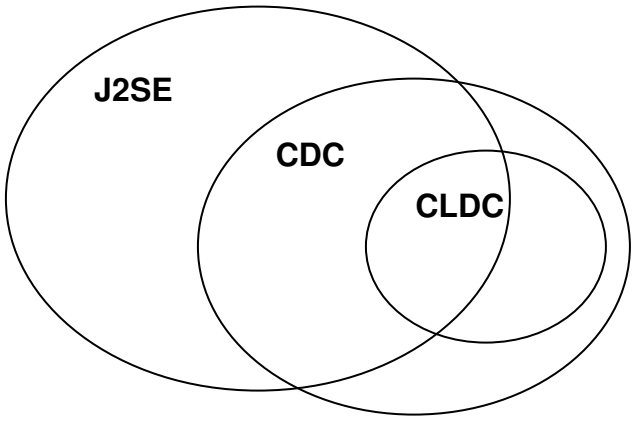

Abbildung 3

CDC	CLDC
• Vor allem für Geräte mit grafischer Oberfläche ausgerichtet • Speicher: 2-16 MB • min. 512KB ROM • min. 256KB RAM • Komplette Implementierung der CVM • Grosse Bandbreite für die Datenkommunikation (mehr als 9600 bps) • TCP/IP • permanente Verbindung	• Minimale grafische Oberfläche • Speicher: 160-512KB • min. 128 KB ROM für Java VM und CLDC-Libraries • min. 32 KB for Java runtime and objects • 16bit oder 32bit Prozessor • Beschränkte VM, die KVM • Geringe Bandbreite für die Datenkommunikation (bis 9600 bps) • normalerweise kein TCP/IP • keine permanente Verbindung • meist drahtlos • Ausgelegt für Geräte geringem Stromverbrauch (Batterie, Akku)

4. Profile

Profile setzen direkt auf Konfigurationen auf und erweitern diese um grafische Oberflächen und Möglichkeiten zur persistenten Datenhaltung. Sie bildet damit eine API um komplette Applikationen zu entwickeln. Profile können ineinander verschachtelt sein oder aufeinander aufbauen. Sie werden im Java Community Process verabschiedet.

Da es unterschiedliche Zielgeräte mit unterschiedlichen Anforderungen gibt, wird auch hier wieder zwischen Profilen für die Konfiguration CLDC und CDC unterschieden, die sich in unterschiedlichen Stufen des Spezifikationsprozesses befinden.
Für die Connected Limited Device Configuration sind dies das Mobile Information Device Profil und das PDA Profil. Das Foundation Profil, das Personal Profil und das RMI Profil liegen für die Connected Device Configuration vor.

4.1 CLDC Profile

4.1.1 Mobile Information Device Profile (MIDP)

Das MIDP Profil ist für die Konfiguration CLDC entwickelt und definiert eine API für Geräte mit persistentem Speicher und Benutzerschnittstellen. Die Benutzerschnittstelle des MIDP wird auf einer semantischen Ebene beschrieben und legt kein konkretes Layout fest, wie beispielsweise beim AWT der Java 2 Standard Edition. Dadurch bleibt die Plattformabhängigkeit selbst bei eingeschränkten Bildschirmflächen und stark unterschiedlichen Formaten erhalten. Für die API grafischer Oberflächen, ist das LCDUI definiert.

In der Implementierung fällt eine gewisse Ähnlichkeit zu Applets auf, daher nennen sich die Applikationen auch Midlets. Midlets haben keine main()-Methode, sondern erben wie Applets von einer abstrakten Oberklasse (javax.microedition.midlet.MIDlet). Zusätzlich zu der LCDUI-Bibliothek (High-Level-API) kann jedes Gerät eigene Low-Level-APIs anbieten, mit dem es möglich ist direkt auf den Bildschirm zuzugreifen um z.B. Diagramme darzustellen. Dabei ist zu beachten, dass in einem dieser geerbten Klassen nur das High-Level oder das Low-Level API verwendet werden kann.

Zusätzlich kommen Klassen für Sicherheit, Internationalisierung und das spezielle Generic Connection Framework hinzu. Dies dient zum Aufbau und Verwalten von Verbindungen, damit das Gerät Kontakt zur Außenwelt aufnehmen kann.

Ein Gerät das dem MIDP Profil gerecht werden will, sollte folgende minimal Charakteristiken aufweisen:

- Displaygröße: 96 x 54 Pixel
- Farbtiefe: 1-bit
- Pixel shape (aspect ratio): ca. 1 : 1
- und als Eingabegerät: “one-handed keyboard”, “two-handed keyboard” oder Touchscreen

Das Mobile Information Device Profile beinhaltet Schnittstellen :

- zum Herunterladen von Java Programmen
- für die HTTP Unterstützung
- zum Festspeicher
- zum graphischen Userinterface

4.1.2 Personal Digital Assistent Profile (PDAP)

Das PDAP Profil ist für PDAs, wie den Palm-Pilot, spezifiziert. Das PDA-Profile geht von Handgeräten mit geringen Ressourcen und einer Anzeigefläche kleiner als 20.000 Pixel aus. Häufig verfügen solche Geräte auch über kein Zeigegerät oder bieten keine Zeicheneingabe an.

Es werden folgende Mindestanforderungen an das Profil gestellt:

- mind. 512kByte Speicher für die Java VM und Bibliotheken
- Batteriebetrieb
- ein einfacher Festspeicher für Applikationsdaten
- die Klassen für die Bildschirmausgabe sollen an das Standard AWT angelehnt sein

4.2 CDC Profile

Für die Connected Device Configuration liegen drei grundlegende Profile vor. Zu den Referenzimplementierungen der Connected Device Configuration werden in Zukunft noch weitere Profile hinzukommen.

Das **Foundation Profile** umfasst Socket Klassen und enthält Klassen aus java.lang und java.io. Das Foundation Profile wird nicht direkt selbst verwendet, sondern dient als Grundlage für weitere Profile, wie das Personal Profile.

Das **Personal Profile** erweitert das Foundation Profile und stellt unter anderem eine grafische Benutzerschnittstelle (AWT) zur Verfügung. Anwendungen die unter Personal Java entwickelt wurden, können in das Personal Profile übernommen werden.

Das **RMI Profile** ermöglich Remote Method Invocation und kann mit der RMI API der Java 2 Standard Edition zusammenarbeiten.

5. J2ME spezifische APIs

Neben den Standard KVM Java Klassen die eine Untermenge der Java Standard Edition bilden sind in der CLDC spezielle Java Pakete enthalten die sich im Namensraum javax.microedition befinden. Dazu zählen das Generic Connection Framework, welches ein Framework für Netzwerkverbindungen zur Verfügung stellt, das Record Management System, um Daten persistent in Datenbanken speichern zu können und das LCDUI, dass zur Erstellung von grafischen Benutzeroberflächen auf Mobiltelefonen genutzt wird.

5.1 Das Generic Connection Framework

Bei diesem Framework handelt es sich um eine Vereinheitlichung von Netzwerkzugriffen, das nur aus einer einzigen Klasse und einer Vielzahl von Schnittstellen für Verbindungstypen besteht. Bei der Java 2 Standard Edition bedient man sich, um eine Netzverbindung zu einem anderen Rechner herzustellen, der Klassen Socket oder Datagram aus dem Paket java.net. In der Micro Edition geschieht der Verbindungsaufbau generell über die Klasse Connector. Diese Klasse liefert bei erfolgreichem Verbindungsaufbau eine konkrete Klasse, welche die Schnittstelle Connection implementiert zurück, und kann in eine der in Abbildung 4 dargestellten Verbindungsarten konvertiert werden, da alle weiteren Schnittstellen von Connector abgeleitet sind.

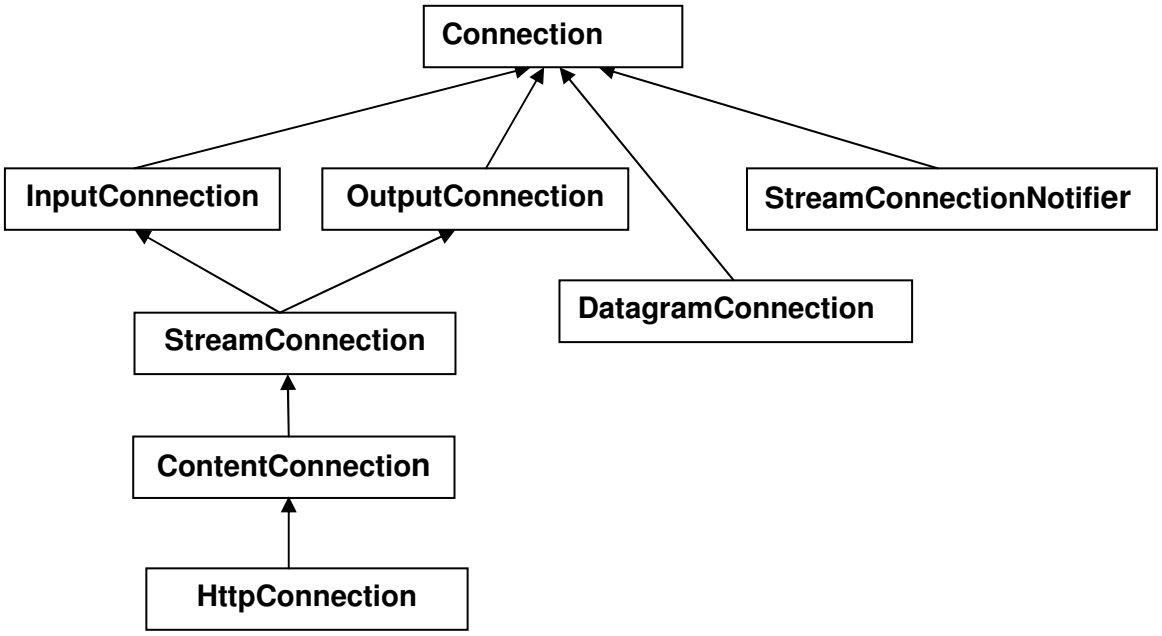

Abbildung 4: Das Generic Connection Framework

Während in der Java 2 Standard Edition Klassen z.B. zur Socket- oder Datagramm-Verbindung im Paket java.net vorliegen, muss in der Java 2 Micro Edition auf das Generic Connection Framework im Paket javax.microedition.io zurückgegriffen werden.

Das Generic Connection Framework vereinheitlicht die unterschiedlichen Verbindungarten und fasst sie in einer „Factory“ zusammen. Das Framework besteht aus einer Klasse Connector, deren statische open-Methode eine Instanz des Datentyps Connection zurückliefert. Connection ist das Interface von dem alle weiteren Schnittstellen des Generic Connection Framework abgeleitet sind. Die von der open-Methode erhaltene Verbindung muss deshalb unter Umständen mit einem Typecast in die konkrete Verbindungsform konvertiert werden.

Mit dem Generic Connection Framework ist es für Hersteller möglich für Geräte auf die sie die KVM portieren, weitere Verbindungsprotokolle, z.B. Bluetooth oder IrDA, zu unterstützen.

5.2 Das Record Management System (RMS)

Da die Geräteklasse für welche die CLDC Konfiguration konzipiert wurde im allgemeinen kein Dateisystem für die Speicherung von Benutzerdaten zur Verfügung stellen, abstrahiert das Record Management System Zugriff auf Datenbanken. Die RMS ist abhängig vom Gerätehersteller, ob z.B. die Speicherung in statischem Speicher oder auf Speicherkarten durchgeführt wird.

Das javax.microedition.rms Paket, besteht aus nur einer Klasse RecordStore, welche die gesamte Funktionalität für die persistente Datenspeicherung auf Mobilen Geräten wie z.B. das Öffnen eines bestehenden oder das Anlegen eines neuen RecordStores, wie auch das Schließen und Löschen zur Verfügung stellt. Das RMS unterstützt nur die Speicherung von „Byte-Arrays“ die über einen Index der bei „1“ beginnt angesprochen werden können, wobei die Größe der einzelnen „Records“ unterschiedlich sein kann. Dieser Zusammenhang wird in Abbildung 5 schematisch dargestellt. Da das RMS keine Datenbank Datentypen unterstützt ist der Anwendungsentwickler selbst dafür zuständig beim Speichern eines Java-Datentyps diesen in ein Byte-Array und beim Lesen dieses Byte-Array wieder in einen Java-Datentypen zu konvertieren.

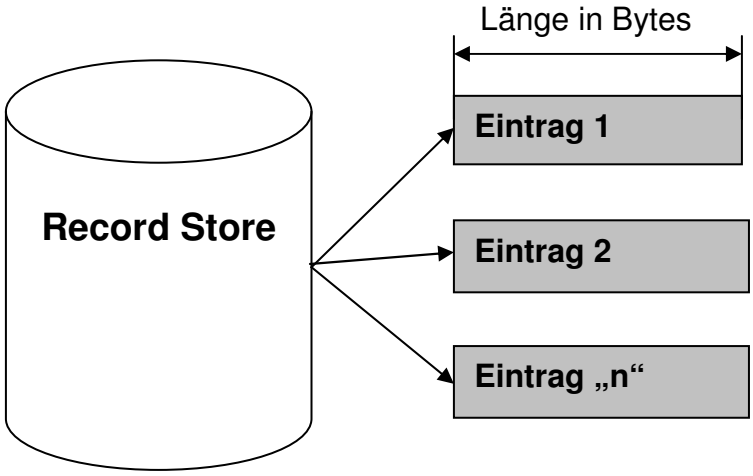

Abbildung 5: Schematischer Aufbau eines Record Stores

5.3 Das LCDUI des Mobile Information Device Profils (MIDP)

Bei dem LCDUI handelt es sich um die Benutzerschnittstelle die im Zuge des MID Profils für Mobiltelefone spezifiziert wurde, siehe CLDC Profile. Eine Übersicht über die im LDCUI verfügbaren GUI Klassen zeigt folgende Abbildung [3].

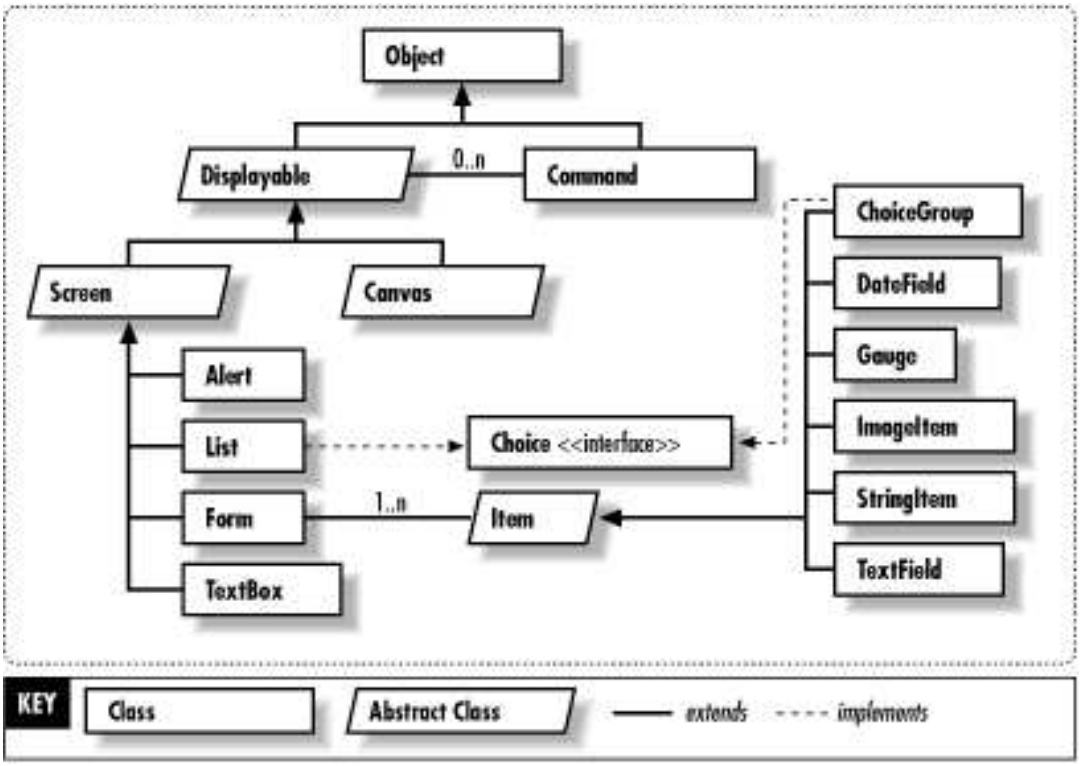

Abbildung 6: Die im LDCUI verfügbaren GUI Klassen

5.4 APIs von Drittanbietern

Viele der bereits erwähnten Einschränkungen wie z.B. das fehlende GUI API für die PalmOS basierte CLDC Referenzimplementierung, die fehlende Unterstützung für Fließpunktzahlen wie auch der eingeschränkte Sprachumfang machen die Verwendung von Drittanbieter Bibliotheken die speziell für die KVM entwickelt wurden notwendig. Im folgenden wird das kAWT Projekt, wobei es sich um eine AWT Untermenge für die KVM handelt, und MathFP, dass Fixkomma Integer Arithmetik zur Verfügung stellt beschrieben.

5.4.1 Das Kilobyte Abstract Window Toolkit (kAWT) für die KVM

Da die in der Palm-KVM zur Verfügung gestellten GUI Klassen sehr palmspezifisch sind erschweren sie die Portierung von Desktop Applikationen auf die CLDC KVM . Ein weiteres Problem dieser GUI Klassen ist die begrenzte Unterstützung der Ereignisbehandlung wie auch das Fehlen von einem Layout Manager, dass eine absolute Positionierung der Dialogelemente erfordert. Das kAWT, welches eine Untermenge der Desktop AWTs Funktionalität beinhaltet, behebt viele dieser Einschränkungen. Dadurch sind alle kAWT Applikationen aufwärtskompatibel und können ohne Änderungen der Programmquellen auch auf dem Desktop PC ausgeführt werden.

Im Gegensatz zu dem Desktop AWT basiert die kAWT Bibliothek nicht auf Betriebsystem Dialogelementen, sondern greift nur minimal auf die Palm-KVM Routinen für Grafik und Ereignisbehandlung zurück wodurch die kAWT Implementierung vollständig in Java erfolgen konnte. Durch diese minimale Abhängigkeit zur Geräteplattform ist die Portierung auf andere mobile Plattformen relativ einfach und sogar schon für MIDP-Geräte verfügbar.

5.4.2 MathFP Fixkomma Integer Arithmetik

Wie im Abschnitt Konfigurationen erwähnt, sind in der CLDC Konfiguration die elementaren Datentypen float oder double nicht verfügbar. Somit scheint es mit der KVM unmöglich einfache Anwendungen wie z.B. ein Programm zur Währungsumrechnung zu implementieren. Diesem Missstand hilft die Bibliothek MathFP von Onno Hommes ab. Sie stellt Fixkomma-Arithmetik auf Basis des 32-Bit Integer-Datentyps der KVM zur Verfügung. Hierbei werden die oberen 12 Bits für den ganz Zahl Teil und die unteren 19 Bits für den Dezimalteil genutzt. Bei dieser Aufteilung ergibt sich ein Wertebereich von -524287.9997 bis 524287.9997. Die Bibliothek besteht aus einer Sammlung statischer Methoden, die mathematische Operationen von der einfachen Addition bis zur Quadratwurzel oder Exponentialfunktion zur Verfügung stellen. Zusätzlich sind Methoden vorhanden, um Integer oder Strings in das MathFP Format, und zurück, zu konvertieren.

6. Beispiele

6.1 Codebeispiel: "Hello World" für MIDP

```
// Copyright 2000-2001 by Sun Microsystems, Inc.,
// 901 San Antonio Road, Palo Alto, California, 94303, U.S.A.
// All rights reserved.

package examples.helloworld;
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

public class HelloWorld extends MIDlet implements CommandListener {
    private Command exitCommand;
    private TextBox tb;

    public HelloWorld() {
        exitCommand = new Command("Exit", Command.EXIT, 1);
        tb = new TextBox("Hello MIDlet", "Hello, World!", 15, 0);
        tb.addCommand(exitCommand);
        tb.setCommandListener(this);
    }

    protected void startApp() {
        Display.getDisplay(this).setCurrent(tb);
    }

    protected void pauseApp() {}
    protected void destroyApp(boolean u) {}

    public void commandAction(Command c, Displayable d) {
        if (c == exitCommand) {
            destroyApp(false);
            notifyDestroyed();
        }
    }
}
```

6.2 Kompilieren einer Anwendung

Bei der Erstellung von Anwendungen der Micro Edition sind nicht nur die eingeschränkten API's ein Hindernis, sondern auch die im Vergleich zu anderen Java Plattformen komplizierten Compilierungsschritte. Zusätzlich ist die Erstellung dieser Anwendungen je nach Zielplattform unterschiedlich. Zunächst muss die Java-Quelldatei wie gewohnt in eine .class-Datei compiliert werden. Dem Java-Compiler muss jedoch mit dem Parameter bootclasspath mitgeteilt werden, dass die KVM-Sytembibliotheken anstelle der Java-Standardbibliotheken benutzt werden.

Um die Ladezeiten der Klassendateien zu verkürzen, muss anschließend der Preverifier durchlaufen werden, der Hinweise für die Verifizierung in die Klassendatei einträgt. Der Preverifier überprüft auch die Gültigkeit der Datentypen. Zuletzt muss die verifizierte Klassendatei in ein je nach Zielplattform verschiedenes Format gepackt werden. Die folgende Abbildung zeigt den Verlauf einer Kompilierung.

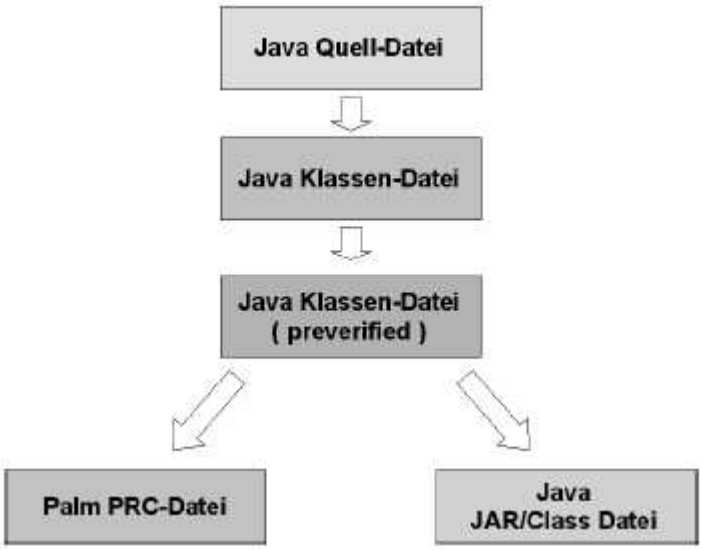

Abbildung 7: Kompilierung

7. Anhang

7.1 Tools

Durch die Aktualität der Micro Edition versuchen viele Anbieter ihre Entwicklungsplattformen durchzusetzen. Beispiele von Entwicklungsplattformen für die CLDC/MIDP:

- Forte Java (Sun Microsystems): Erweiterung mit „Wireless Toolkit“
- CodeWarrior for Java
- IBM Visual Age Micro Edition
- Borland Jbuilder: Erweiterung mit “Handheld Express”

7.2 JavaCard

Die JavaCard-Spezifikation ermöglicht es, Java auf Smartcards laufen zu lassen. Sofern sich der Programmierer an die Java Card Application Environment (JCAE) gehalten hat ist die Plattformenabhängigkeit auch auf Smartcard gewährleistet. Es ist sogar möglich mehrere Applikationen auf einer einzigen Karte laufen zu lassen. Die Plattform ist zu den existierenden Kartenstandards kompatibel. Eine Besonderheit ist die Möglichkeit der Aktualisierung der Karte. Ein Hersteller einer Karte kann eventuelle Änderungen und Verbesserungen auf der Karte neu installieren, ohne die Kartenhardware ersetzen zu müssen.

7.3 Jcontrol

Die Java-Implementierung ist speziell auf embedded Systeme abgestimmt. Sie ist auch für 8bit Systeme geeignet und wurde für Steueranwendungen optimiert. Jcontrol lässt sich mit JavaCard der Firma Sun vergleichen. Die virtuelle Maschine dieser Plattform benötigt nur ca. 20kByte Speicherplatz und ist somit auf einem sehr ressourcenbeschränkten System lauffähig. Die signifikanten Größeneinsparungen bei der virtuellen Maschine wurden durch Einschränkungen der Grafik und Internetfunktionalität bewerkstelligt. Es wurde unter anderem nur ein kleiner Teil von java.lang.* übernommen.

Die herkömmliche virtuelle Maschine der Desktop-Java-Plattform verwendet intern eine Wortbreite von 32bit. Durch Halbierung der Wortbreite auf 16bit konnte die Hälfte des benötigten Speichers eingespart werden. Datentypen wie long, double, int und oat sowie deren Bytecodes wurden weggelassen. Der Bytecode der Applikation wird in dieser Form direkt aus dem ROM interpretiert, da auf den Host-Systemen für JIT -Compiler nicht genügend Speicher zur Verfügung steht.

Trotz der gravierenden Einschränkungen wurde nicht auf die Garbage Collection und das Multithreading verzichtet, es entstehen aber Kompatibilitätsprobleme mit Anwendungen, die auf einer anderen Plattform basieren. Wir können hier also nur von einer bedingten Plattformenabhängigkeit sprechen.

Um den Zugriff auf E/A-Zusatzeinheiten zu gewährleisten, wurden spezielle Klassen implementiert, die diese Architekturabhängig repräsentieren. Eine Interrupt gesteuerte Ereignisabarbeitung wird mittels einer mit Java-Objekten konfigurierbaren Zustandsmaschine realisiert.

Es wurde auf die Zugriffskontrolle wie protected, public verzichtet. Der Bytecode wird nicht auf Korrektheit überprüft und muss daher auf der Entwicklungsplattform vor dem Aufspielen auf das eingebettete Systeme überprüft und getestet werden.

Der Aufbau der JControl-Plattform untergliedert sich in 3 Ebenen:

1. Digitale Ebene:
Diese Ebene stellt die digitalen E/A-Funktionen zur Verfügung, wie z.B. das Setzen oder Lesen eines PortPins. Die Funktionen sind abstrakt gehalten und keinen bestimmtem Anwendungen zugeordnet. Somit können auch bei einem Wechsel der Plattform im Allgemeinen die selben Dienste zu Verfügung gestellt werden. Diese Ebene ist für alle JControl-Umsetzungen identisch.

2. Komponentenorientierte Ebene:
Sie stellt die anwendungsorientierten Funktionen für die Steuerung von Komponenten zur Verfügung. Hier können beispielsweise Schrittmotoren oder LCD's bedient werden ohne dies explizit programmieren zu müssen.

3. Geräteorientierte Ebene:
In dieser Ebene können komplexe Geräte gesteuert werden. Die Funktionen müssen aber hierbei speziell an das Gerät angepasst werden. Diese Ebene ist weniger flexibel, aufwendiger zu implementieren, jedoch sehr komfortabel zu verwenden.

7.4 Java Community Process

Die den Konfigurationen und Profilen der Java 2 Micro Edition zu Grunde liegenden Spezifikationen, werden durch den Java Community Process (JCP) verfasst. Der JCP wurde 1995 von Sun gegründet und besteht momentan aus ca. 300 Firmen und Privatpersonen, unter anderem Sun, IBM, SAP und Siemens (Microsoft ist nicht Mitglied). Das Tagesgeschäft wird vom Process Management Office (PMO) bei Sun erledigt, während die Spezifikationen in den Expertengruppen entwickelt werden.
Die Grundlage für eine Spezifikation bildet dabei ein Java Specification Request (JSR), der die vier Phasen des Java Community Process durchläuft. Nach jeder der vier Phasen entscheidet das Exekutiv- Komitee (EC) des JCP, ob der Specification Request die nächste Phase erreicht.

In der Einleitungsphase wird zunächst von einem Mitglied des JCP eine Anfrage (JSR) für eine neue Spezifikation oder Überarbeitung einer bestehenden Spezifikation gestellt. An der anschließenden Diskussion dieser Spezifikation können sich neben den Experten und Mitgliedern auch Nichtmitglieder beteiligen.

Darauf folgt die Entwurfsphase, in der eine Expertengruppe für den Specification Request von der JCP Führung eingesetzt wird. Die Expertengruppe verfasst einen ersten Entwurf der Spezifikation.

In der dritten Phase wird die Spezifikation öffentlich diskutiert und die endgültige Fassung von der Expertengruppe erstellt. Diese wird von der JCP Führung durch die Referenzimplementierung (RI) und das Technology Compatibility Kit (TCK) vervollständigt. Mit Hilfe des TCK kann für eine Implementierung einer Spezifikation überprüft werden, ob sie dieser Spezifikation tatsächlich entspricht. Danach wird die endgültige Fassung vom Exekutivkomitee freigegeben und die Expertengruppe aufgelöst.

Eine freigegebene Spezifikation bleibt in der darauffolgenden Wartungsphase, in der nötige Updates vorgenommen werden.

Java Specification Requests der Java 2 Micro Edition:

Bezeichnung	**JSR #**	**Phase (Stand 09.10.01)**
Connected Limited Device Configuration	JSR 30	Phase 3 (Proposed Final Draft)
Mobile Information Profile	JSR 37	Phase 3 (Proposed Final Draft)
PDA Profile	JSR 75	Phase 1 (JSR Review Ballot)
Connected Device Configuration	JSR 36	Phase 3 (Proposed Final Draft)
Foundation Profile	JSR 46	Phase 3 (Proposed Final Draft)
Personal Profile	JSR 62	Phase 1 (JSR Approval Ballot)
RMI Profile	JSR 66	Phase 3 (Proposed Final Draft)

8. Zusammenfassung

Die Java-Plattformen wie die Java 2 Micro Edition ermöglichen die Entwicklung von Anwendungen für eingebettete Systeme mit der Java-Technologie.

In diesem sehr heterogenen Bereich mit unterschiedlichsten Geräten und Plattformen sind die vielfältigen Einsatzmöglichkeiten und die Portabilität von Java ein großer Vorteil. Auch die sonst üblichen Argumente, wie Wiederbenutzung von Software, Sicherheit, Langlebigkeit sprechen hier genauso für Java, wie der inzwischen große Pool an Java-Entwicklern, auf die zurückgegriffen werden kann.

Dennoch wird deutlich, das für die heute erhältlichen mobilen Unterhaltungsgeräte mit geringen Ressourcen, viel Aufwand betrieben werden muss (z.B. Kompilierung), um für sie Java-Anwendungen zu erstellen. Besonders in den Fällen in denen kein Dateisystem vorhanden ist und der Umweg über C-Code nötig ist, sollte die Entscheidung für oder gegen Java genau abgewägt werden.

Wenn die Geräte der nächsten und übernächsten Generation die Einschränkungen durch die zur Verfügung stehenden Ressourcen, die geringe Bandweite der Verbindungen und die Schwierigkeiten bei der Ein- und Ausgabe nach und nach verlieren, wird die Entwicklung von Java-Anwendungen für eingebettete Systeme ähnlich komfortabel, wie für Desktop-Anwendungen und Applets.

Java in embedded Systems: Pro und Contra:

Pro	**Contra**
• Dynamisches Laden der Software • Hochgradig portabel • Unterstützung von Netzwerken • Modern, einfach, sicher • Gutes Sicherheitsmodell (Änderungen bei CLDC) • Verwendung der Standard APIs • Eine einheitliche Sprache	• Java ist zu langsam • zu umfangreich (API) • anspruchsvoll (zu hoher Level) • nicht echtzeitfähig

9. Quellenangabe

[1] http://www.javasoft.com/
[2] http://www.onjava.com/
[3] http://www.microjava.com
[4] http://www.sun.de/Downloads/Praesentationen/Wireless/Vortraege/1/Merck.pdf
[5] http://java.sun.com