

SW-Entwicklung über alle Java-Varianten hinweg

Peter Rudolph
JavaForum 2004, Stuttgart

Einleitung

Mobilgeräte

Java-Varianten

Erfahrungen und Probleme

Lösungen

■ Unternehmensdaten mobil verfügbar machen

- Zugriff auf Backoffice (z.B. SAP, Oracle, ...)
- möglichst lange Offline arbeiten können
- Zugriff auf WebServices ermöglichen

■ Unabhängigkeit von den Zielgeräten

- Geschäftslogik nur einmal entwickeln
- GUI und Benutzerführung auf Zielgerät optimieren
- Zielgeräte sind: Handy, PDA, Tablet PC, Notebook



Einleitung

Mobilgeräte

Java-Varianten

Erfahrungen und Probleme

Lösungen

■ **Notebook**

- Intel-Kompatibel ab 500MHz
- ab 128 MB
- Farbdisplay 1024x768 oder mehr
- JDK 1.4 von SUN oder IBM

■ **Windows CE**

- RISC CPU ARM oder XScale (früher MIPS, Hitachi SH3/SH4), typisch 100-400MHz
- typisch 32 - 128 MB
- Farbdisplay, meist 240x320 (teilweise schon 480x640)
- Personal Java oder Personal Profile von diversen Drittanbietern

■ **Symbian OS (Nokia Communicator, SonyEricsson P800, ...)**

- RISC CPU 2-64 MB RAM, meist erweiterbar durch Speicher-Karte
- unterschiedlichste Display-Formate von Handy- bis PDA-Auflösung
- MIDP 1.0 oder 2.0, teilweise auch Personal Java fest eingebaut

■ Palm OS 4

- Motorola DragonBall (68k Familie) mit 16-33 MHz, 2-16MB RAM
- s/w Display 160x160, teilweise auch farbig, (Sony320x240)
- Java Micro Edition MIDP 1.0 von SUN
- kommerzielle JDKs für MIDP und Personal Edition schon seit Jahren angekündigt

■ Palm OS 5

- StrongArm oder XScale Prozessor, typisch 100 bis 400 MHz, 16-128MB RAM
- Display von 160x160 s/w bis 480x640
- Java Micro Edition MIDP 1.0 von SUN
- kommerzielle JDKs für MIDP und Personal Edition angekündigt

■ Java-Handy

- RISC-Prozessor, typisch 200 bis 500 kB RAM
- Display ab 96x64 s/w bis PDA-Auflösungen
- Java Micro Edition MIDP 1.0 (wenige mit 2.0) fest eingebaut

Einleitung

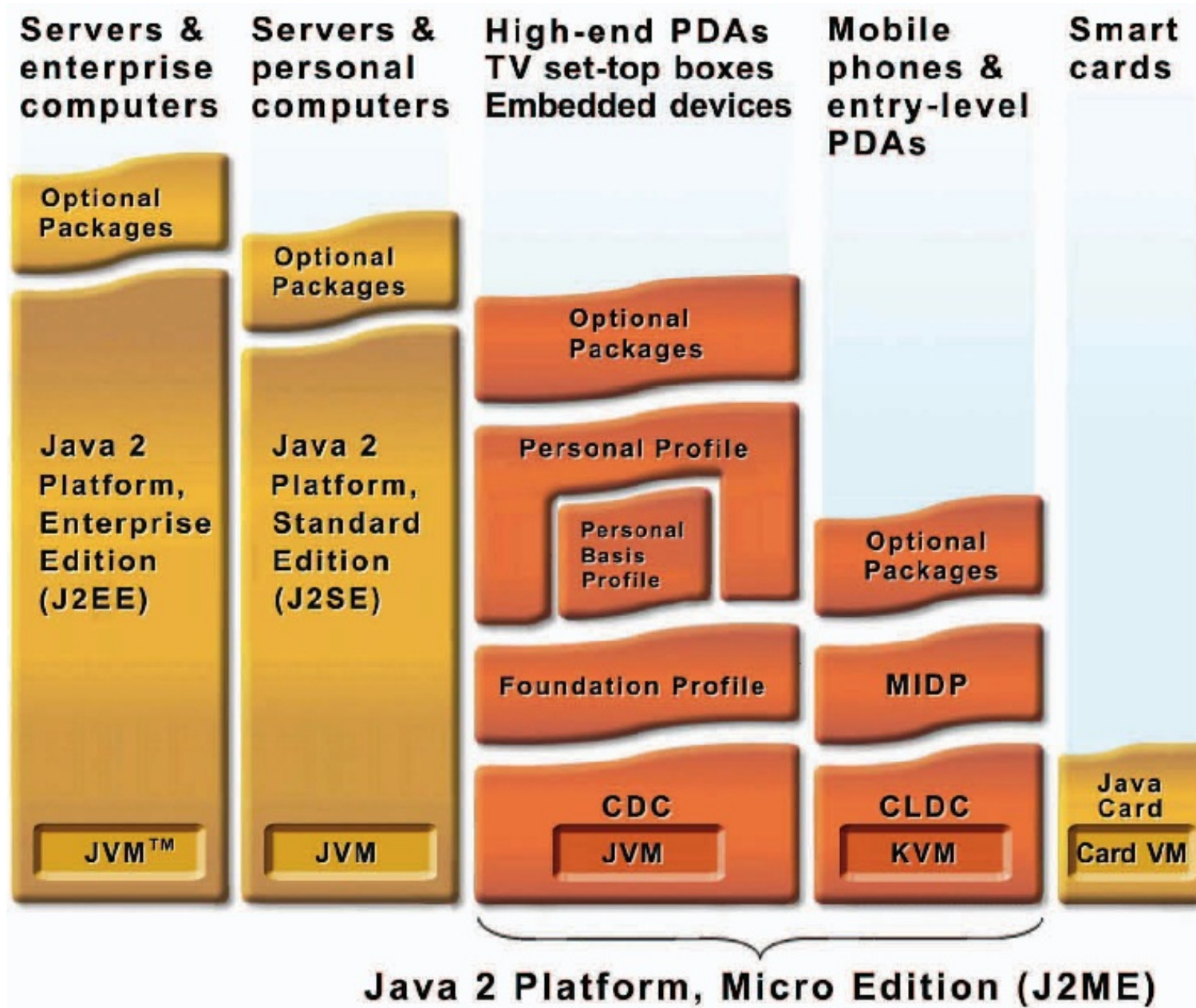
Mobilgeräte

Java-Varianten

Erfahrungen und Probleme

Lösungen

- **am weitesten entwickelt**
- **JDK 1.4 ist aktuell, JDK 1.5 im Beta-test**
- **Performance**
 - Just-In-Time Compiler
 - C++ ist ca. 2-3 mal so schnell
 - AWT ist sehr langsam (und damit auch Swing)
 - seit JDK 1.3 wird Performance optimiert
- **JDK 1.1**
 - erste brauchbare Java-Version
 - Swing nur als optionales Paket
 - Grundlage für Personal Java
- **JDK 1.2**
 - viele API Änderungen
 - Swing wird fester Bestandteil
 - Collection Classes (List, Map, ...)
- **JDK 1.3**
 - hauptsächlich Bugfixes und Performance verbessert
 - Grundlage für Micro Edition



Einleitung

Mobilgeräte

Java-Varianten

Erfahrungen und Probleme

Lösungen

■ **Personal Java auf CE**

- Collection API fehlt
- langsam, da JIT fehlt bzw. schlecht implementiert
- lästige Bugs in vielen Implementierungen, z.B. in bei Fensterpos., Key-Events, ...

■ **JDK 1.1.4 auf EPOC (Psion 5mx)**

- etwa gleich schnell wie auf CE (trotz Faktor 5 in CPU Speed)

■ **J2ME MIDP 1.0 auf Palm OS 4**

- sehr langsam (Palm V mit 16 MHz)
- VM selbst braucht nur 96kB RAM, unterstützt max. 256kB RAM
- LCDUI schränkt stark ein beim GUI, API insgesamt zu schwach für einen PDA

■ **JBed auf Palm OS 4**

- läuft auf Palm V mit 16 MHz flüssiger als SUN VM auf min 10x schnellerem iPaq
- compiliert auf PC oder beim Laden auf Palm

■ **Compilieren**

- bereits auf dem PC
- beim Laden der Klasse auf dem Palm

■ **Kommerzielle VM-Anbieter**

- reine PDA-Lösungen oder Portierungen von Embedded Java
- viele Lösungen wurden nie marktreif
- inzwischen viele im Besitz von Esmertec: JBed, Jeode, Kada

■ **Waba / SuperWaba**

- enthält nur java.lang - restliches API ist vollständig aber nicht kompatibel
- kurzer Test zeigte überzeugende Performance
- mächtige API vor allem für Mobilgeräte

■ MIDP API

- Winziges API: Vieles muss neu implementiert werden
- eigene API für GUI und HTTP
- kein Filesystem, dafür RecordStore
- Zugriff auf Handy-Eigenschaften wie Addressbuch und SMS erst mit 2.0

■ MIDP in der Praxis

- XML-Verarbeitung stößt an Performance-Grenzen
- LCDUI orientiert sich an WAP --> Benutzerführung katastrophal
- teilweise proprietäre Erweiterungen, z.B. Gaming-API im Siemens S45i
- derzeit rund 150 Java-fähige Handys mit unterschiedlichsten Implementierungen
- Referenzimplementierung auf Palm sehr langsam

■ Probleme der Java Personal Edition

- API entspricht nur JDK 1.1
- keine Collections, standard XML-APIs erfordern min. JDK 1.2
- Referenzimplementierung auf Windows CE sehr langsam
- beste Implementierung CrEme von NSICom (keine Einzellizenzen)
- GUI auf Basis von AWT wenig ansprechend, Table und Tree fehlen

■ Probleme mit kommerziellen VM-Implementierungen

- die meisten stecken in den „Kinderschuh“en
- viele Firmen kündigen jahrelang etwas an, ohne es jemals zu haben
- teilweise sehr teuer
- viele Anbieter für Embedded Systeme
- teilweise mit SWT statt AWT (z.B. PERC)

■ Probleme mit Geräte-Vielfalt

- GUI muß individuell angepaßt werden, Geschäftslogik sollte erhalten bleiben
- kleinste benötigte API bestimmt die verwendbare API
- Datenspeicherung: RecordStore oder XML-Datei oder SQL-Datenbank
- Anwendungsszenario beschränkt Geräte-Auswahl
- viel Einarbeitungsaufwand in Geräte und zugehörige Tools



Einleitung

Mobilgeräte

Java-Varianten

Erfahrungen und Probleme

Lösungen

■ Design-Entscheidungen

- eigene GUI-Bibliothek, die Controls (in MIDP auch Fenster) selbst zeichnet
- eigene Modell-Klassen
- eigener vor allem im Speicherbedarf optimierter XML-Parser und -Binder
- eigene Persistenz-Schicht mit Implementierungen für XML in Dateien und RecordStores sowie Datenbanken über JDBC
- eigene Replikation auf Logikebene statt auf Datenebene
- Entwicklung muss schnell gehen --> Zeit für Optimierung der Benutzerführung
- früher Prototyp des Clients erleichtert Kommunikation mit dem Kunden (Aufwand ca. 1 Personentag für 5 Masken)



Bestellung

MatNr.	Menge	ME	St.
81	1,0	ST	-2
FH-Stahltüre 875/1875, in Innenw			
9652	2,0	ST	-2
DK-Fenster me 116,6/108			

Such. ^ - Sich. Senden Abbr

Liefertermin 25.05.2004 0:00

Anlief. KLEINTE. ▾

1001_70 4.05.2004 4:01

Auftrag 878753 Dr. F. Kohl

Besteller 1001 pit

■ XML

- möglichst vieles wird in XML beschrieben
- Server interpretiert XML
- für Mobilgerät wird Code aus XML generiert
- eigener Parser, der extrem sparsam mit Speicher umgeht
- Kommunikation über SOAP

■ Eigene Modellklassen

- Ersatz für fehlende Collections
- Implementierung auf Basis von MIDP
- Definition in XML Schema



■ Gruppieren nach API

- MIDP für Handys, teilweise Unterscheidung von Version 1.0 und 2.0 nötig
- JDK 1.1 bzw. PersonalJava für Windows CE und große Symbian Handys
- JDK 1.4 für Desktop
- JDK 1.4 mit J2EE-APIs für Server

■ Gruppierung nach AWT-Plattformen

- Desktop mit JDK 1.4
- Windows CE mit CrEme VM
- Symbian-OS mit UIQ (Sony-Ericsson P800/900)

■ Compilieren

- ANT-Tasks für verschieden API-Gruppen
- regelmäßig laufen lassen um Verträglichkeit sicher zu stellen

■ Abstrakter Singleton

- ermöglicht Klasse mit systemspezifischer Methoden-Implementierung
- Abstrakte Klasse mit `getInstance()`
- initialisiert Instanz im Konstruktor und wirft Exception bei doppelter Erzeugung

■ Delegate (Hooks)

- Verzweigung in Methoden mit plattformabhängigem Code

```
public void focusChanged(boolean hasFocus) {
    platform.toggleKeyBoard(hasFocus);
}
```

```
public SkinDialog() {
    platform.setFocusable(this);
}
```

```
public abstract class SkinPlatform {
    public static SkinPlatform getInstance() {
        return instance;
    }
    public SkinPlatform() {
        if (instance != null) {
            throw new RuntimeException("Doppelt Initialisiert");
        }
        instance = this;
    }
    public abstract void setFocusable(SkinDialog skinDialog);
    public abstract void toggleKeyboard(boolean hasFocus);
}
```

```
public class CremePlatform implements SkinPlatform {
    public void setFocusable(SkinDialog skinDialog) { }
    public void toggleKeyboard(boolean hasFocus) {
        if (hasFocus) creme.Keyboard.show();
        else creme.Keyboard.hide();
    }
}
```

```
public class DesktopPlatform implements SkinPlatform {
    public void setFocusable(SkinDialog skinDialog) {
        skinDialog.setFocusable(true);
        skinDialog.setFocusTraversalKeysEnabled(false);
    }
    public void toggleKeyboard(boolean hasFocus) { }
}
```

■ Abstrakte Factory

- erzeugt plattform-abhängige Klassen
- ermöglicht Klassen mit plattformabhängiger **Implementierung** und **Vererbung**

```
public interface DialogComponent {
    public void show(boolean modal);
}
```

```
public class SkinDialog
extends java.awt.Container
implements DialogComponent {

    public SkinDialog(DialogView dlg) {
        window = new Window();
        window.add(this);
    }

    public void show(boolean modal) {
        window.show();
    }
}
```

```
public abstract class ComponentFactory {
    public ComponentFactory getInstance {
        return instance;
    }
    public abstract DialogComponent createDialog(DialogView dialogView);
}
```

```
public class ComponentFactorySkinned implements ComponentFactory {
    public DialogComponent createDialog(DialogView dialogView) {
        return new SkinDialog(dialogView);
    }
}
```

```
public class ComponentFactoryMidp implements ComponentFactory {
    public DialogComponent createDialog(DialogView dialogView) {
        return new MidpDialog(dialogView);
    }
}
```

```
public class DialogView extends View {
    private DialogComponent dlgComp;

    public DialogView(StringID viewID, String title) {
        dlgComp = ComponentFactory.getInstance().createDialog(this);
    }
    public void show(boolean modal) {
        dlgComp.show(true);
    }
}
```

■ Visitor

- Aufruf von Plattform neutralem Code mit Übergabe des Aufrufers
- anschließend "Callback" des plattformabhängigen Aufrufers

```
public class ViewFactory implements MFactory {
    public Model createInstance(StringID modelID,
        ComplexType type, ModelReader reader) {
        ChildList childList;
        Object[] attributes;
        attributes = new Object[type.attributeCount()];
        modelID = fetchAttributes(modelID, type,
            reader, attributes);
        childList = reader.loadChildren(modelID, type);
        if (modelType == DIALOGVIEW) {
            return createDialogView(modelID, type,
                attributes, childList);
        }
    }
}
```

```
public class XmlReader implements ModelReader {
    public ChildList loadChildren(StringID nodeID,
        ComplexType nodeType) {
        XmlParser parser = this.xmlParser;
        boolean finished = false;
        Model childModel;
        ChildList childModelList = new ChildList();

        //----- Create the new Model's sub tree
        while (!finished) {
            parser.next();
            if (parser.getEventType() == parser.START_TAG) {
                childModel = buildModel(nodeID.getNamespace(),
                    nodeType);
                childModelList.add(nodeID, childModel);
            }
        }
        return childModelList;
    }

    private Model buildModel(ComplexType childRules) {
        StringID relationID = xmlParser.getName();
        Type childType = childRules.getChildType(relationID);
        MFactory factory = root.getFactory(childType);
        return factory.createInstance(relationID, childType, this);
    }
}
```

■ **SUN's Seite zu Mobile Java**

developers.sun.com/techttopics/mobility/

■ **Mobilgeräte und Feature-Übersicht**

jal.sun.com/webapps/device/device

■ **Kommerzielle VMs**

- CrEme - NSICom (www.nsicom.com)
- PERC Virtual machine - NewMonics (www.newmonics.com)
- VisualAge Micro Edition von IBM (www.embedded.oti.com)
- JBed, Jeode, Kada - Esmertec (www.esmertec.com)

■ **Open Source VMs**

- Kaffe (www.kaffe.org)
- Waba (waba.sourceforge.net)
- SuperWaba (www.superwaba.com)

■ **Open Source - native Compiler**

- GCJ (gcc.gnu.org/java)
- Jump (sourceforge.net/projects/jump)

PI-Data GbR

<http://www.pi-data.de>

peter.rudolph@pi-data.de