Java in der Welt der Handys

Matthias Hemetsberger
Joseph Erlinger
Erwin Schwab
Rudi Dittrich

Überblick

- Einführung in MIDP
- Zusatzpackages:
  - WMA (wireless messaging API)
  - MMA (mobile media API)

MIDP = Mobile Information Device Profile

Einführung in MIDP

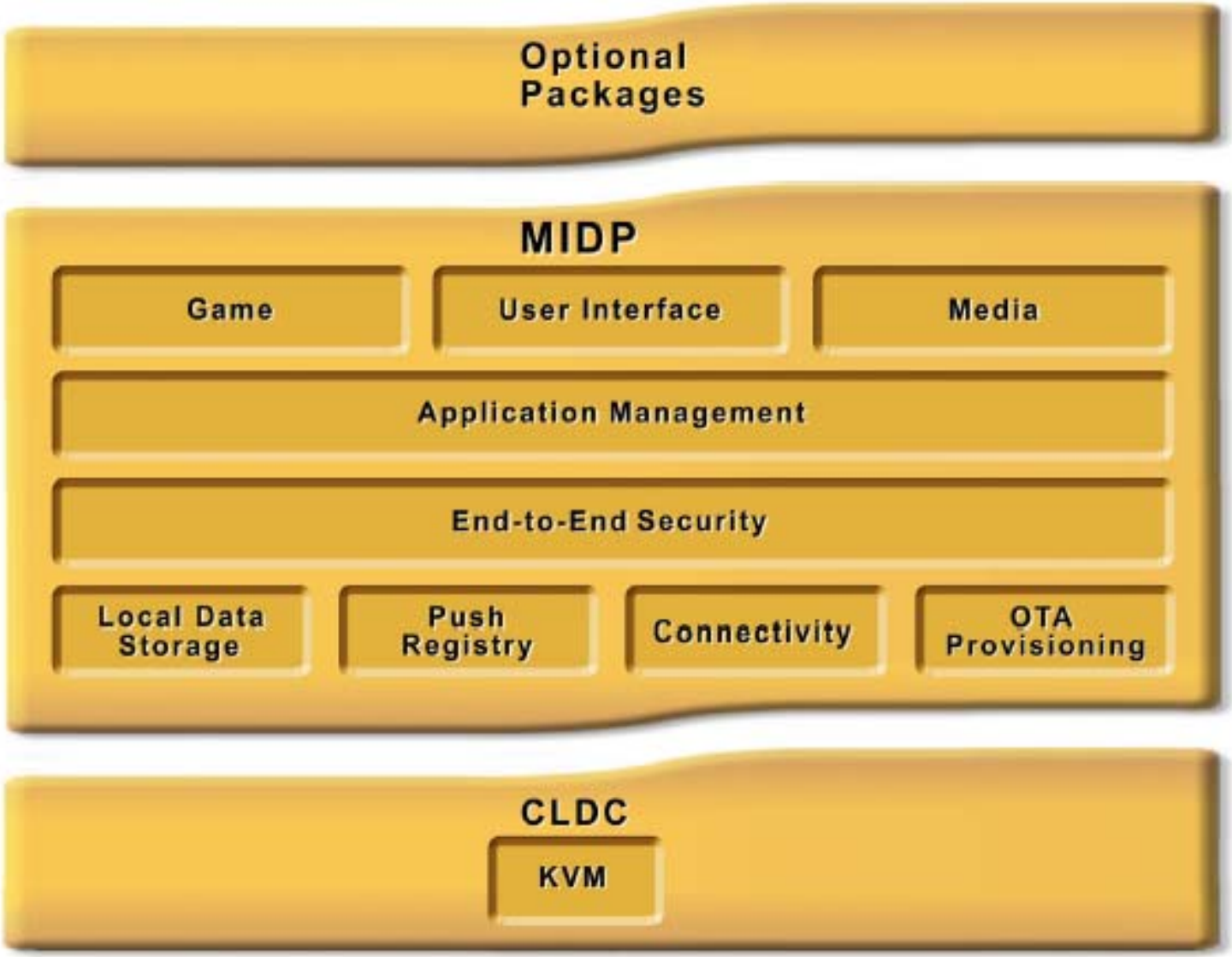
- MIDP = Mobile Information Device Profile
- Java basierend aufgebaut
- Mehrere API‘s und CLDC ergeben eine Entwicklingsumgebung
- Applikationen direkt am Endgerät

CLDC = Connected Limited Device Configuration

J2ME, CLDC, KVM, MIDP

- mobile und feste Plattformen möglich
- aus CLD hat sich CLDC entwickelt
- Herz der CLDC ist die KVM
- In dieser Konfiguration
  - keinen UI-Code implementiert
  - Geräteabhängige Netzwerkfähigkeit

J2ME = Java2 Micro Edition
CLDC = Connected Limited Device Configuration
KVM = K-Virtual Machine
MIDP = Mobile Information Device Profile


Optional Packages
MIDP
Game
User Interface
Media
Application Management
End-to-End Security
Local Data Storage
Push Registry
Connectivity
OTA Provisioning
CLDC
KVM

KVM

- Virtuelle Maschine:
  - Bibliotheken laden
  - Spezielle Klassen laden
- Entwicklungsziele
- Spezifikationen:
  - 40 Kbyte an reinem Objektcode
  - 16/32-bit RISC/CISC µP
  - Benötigter Speicher: 160 K, 128 für die KVM und deren Bibliotheken
  - multi-threading und Garbage Collection Fähigkeit

CLDC

- Connected Limited Device Configuration
- Bibiliotheken:
  - Java.lang (für fundamentale Klassen: Byte, Math, ...)
  - Java.io (für Ein - und Ausgabe)
  - Java.util (Kalender, Random Zahlen, etc.)
  - Java.microedition.io (für Netzwerkbasics)

MIDI

- API‘s:
  - User Interface : javax.microedition.lcdui
  - HTTP: javax.microdeition.io.HttpConnection
  - Persistence Storage : javax.microedition.rms
  - Application Lifecyle : javax.microedition.midlet
  - Timers : java.util.Timer & java.util.TimerTask
  - Neue Exception : java.lang.IllegalStateException
- UI-API besteht aus:
  - High level
  - Low level

Vorraussetzungen für MIDP

- Hardware:
  - 166MHz Prozessor oder schneller
  - 64 Mbyte RAM
  - 30 Mbyte Hauptspeicher
- Software:
  - Java 2 standard Edition (J2SE = Java Desktop Editon) SDK Version 1.3 oder höher
  - J2ME Wireless Toolkit (J2MEWTK)
  - Texteditor: Notepad, Emacs etc.
- Toolkit:
  - http://java.sun.com/products/j2mewtoolkit/

Wireless Messaging API

- Beschreibung
- Präsentation einer Message
- Sending and Receiving Messages
- Security im WMA
- Permission für MIDP Plattform

WMA Beschreibung

- Messaging Api basierend auf generic connection Framework
  - Definiert in CLDC
- Javax.microedition.io
  - Input / output –Funktion
  - Netzwerkfunktionalität
- Datenverwaltung in recourcenarmen Umgebungen (Handy)

CLDC = Connected Limited Device Configuration
WMA = Wireless Messaging API

Arbeitsprinzip

- Öffnen im client und im server - mode
- Modell der Datagrammfunktionalität
  - Auch für user datagramm protokoll (UDP) verwendet

Unterschiede

- Messaging interfaces und Datagram nicht kompatibel
- Simultane Benutzung ungeeignet
- Interface ist im javax.wireless.messaging definiert

Präsentation einer Message (1)

- Message besteht aus 2 Hauptteilen:
  - Einem Adresspart
  - Einem Datenpart
- Message präsentiert durch Instanz einer Klasse
- Interface besitzt Methoden die für alle Messages gleich sind

Präsentation einer Message (2)

- Im Paket javax.wireless.messaging ist Basisinterface enthalten
- Gleich für alle Messages
- Trägt den Namen: MESSAGE

MESSAGE (1)

- Zwei Subklassen von der Klasse Message:
  - Textmessage
  - Binarymessage
- Subklassen können Daten in Strings oder Bytearrays speichern.

MESSAGE (2)

- Spezielle Formate weder deklariert als Binarymessage noch als Textmessage
- Möglichkeit eigene Subinterfaces zu erstellen

Sending/Receiving Messages (1)

- Realisiert mit Hilfe eines connection interfaces
  - Name: Interface Message Connection
- Für Verbindungsaufbau benötigt man:
  - Objekt in der Applikation, welche die Messageconnection implementiert von der Klasse Connector
  - Uniform Recource Locator Connection String (URL) = Adressdefinition

Sending/Receiving Messages (2)

- Client Mode Connection (zum senden von Messages)
  - Benötigt: spezifische Adresse
- Server Mode Connection (zum empfangen von Messages)
  - Benötigt: URL-Connection String (identifiziert Empfänger)

Sending/Receiving Messages (3)

- Klasse Message Connection stellt ObjectFactory Methoden zur Verfügung
  - Action Listener zum empfangen von Messages (synchronisierte Blockmethode)
  - Security Exception
- GFC Methoden zum lesen von Streams
- Interface unterstützt keine Stream basierenden Operationen (Errormeldung)

Security (1)

- Message benötigt permission zum Ausführen von Operationen
- Implementierung Benutzerdefiniert
- Abhängig von:
  - Message-Typ
  - Adressierung

Security (2)

- Zwei Arten der Adressierung:
  - Device-Adressierung
  - Portnumber
- Security Exception:
  - Receive() und Send()
  - Priviligierte Messages
  - Inateraktion mit Netzwerk

Permission for MIDP 1.0

- Wenn JSR 120 – Schnittstellen vorhanden sind auf einem MIDP – fähigen Gerät, so gibt es keinen formalen Mechanismus um eine Permission zu identifizieren.
- Einige Systeme überlassen Entscheidung den Benutzer spezielle Operationen auszuführen
  - Wird Operation abgelehnt -> Security-Exception möglich

Javax.microedition.io

- Package beinhaltet Plattform network interfaces
  - Wurde verändert um diese Plattform für Nachrichtenverbindungen nutzen zu können
  - Ebenso beinhalten Package die Connectorklasse von MIDP 2.0
  - Beinhaltet Security Exception (als rückgabewert von Methode open())

Javax.microedition.io

- Wenn Nachrichtenverbindung auf einer MIDP 1.0 extra Plattform für Security Exception
- User wird gefragt und entscheidet über Security Exception

Connector Klasse (1)

- Klasse entwickelt um neue Connector Objecte zu erzeugen
- Erschaffung von Verbindungen dynamisch geschaltet durch Protokolldurchführungsklasse

Connector Klasse (2)

- Name der Protokolldurchführungsklasse aus Plattformnamen und Protokollnamen gebildet
- ParameterString der Ziel beschreibt gleich Url Format
- {schema}:[{target}][{parameter}]

Connector Klasse (3)

- Wählbarer zweiter Parameter offene Funktion
  - Ist modeflag welche Protokollhandhaber Absicht des Anrufcodes zeigt
  - Möglichkeiten verändern sich → Verbindung gelesen geschrieben oder beides
  - Gültigkeit des Flags ist protokollabhängig
  - zB. Verbindung zu Drucker – kein Lesezugang daher IllegalArgumentException

Connector Klasse (4)

- Wählbarer dritter Parameter ist logisches Kennzeichen (boolean flag)
  - Zeigt an ob der Anrufcode mit Time out Exceptions umgehen kann
  - Wenn diese Flag gesetzt und time out Bedingung erkannt, InterruptetdIOException möglich
  - Diese Flag nur Hinweis für Protokollhandhaber, keine Garantie für Fehlerausgabe
  - Wenn Flag nicht gesetzt, keine Exception

Connector Klasse (5)

- Da Verbindung häufig geöffnet, um Zugang zu Input Output Streams zu erhalten – hierfür eigene zweckmässige Funktionen vorgesehen

- Für genauere Info siehe DATAGRAMMCONNECTION

Mobile Media API

- Einleitung
  - API für Multimedia-Anwendungen
  - MMA kann für verschiedene Geräte verwendet werden
    - Handys
    - PDA‘s
  - Unabhängig von Formaten und Protokollen

Protocol-Content-Handling (1)

- Von API verwendete „high-level“ Objekte
  - DataSourceObject
  - PlayerObject
- Befinden sich im Paket javax.microedition.media.protocol und javax.microedition.media.Player
- DataSourceObject: fürs protocol-handling zuständig, Details bleiben verborgen

Protocol-Content-Handling (2)

- PlayerObject, zuständig fürs Content-handling
- Player wird erzeugt über Factorymechanismus
- Factory heißt Manager und befindet sich in javax.microedition.media.control

Feature Sets (1)

- Jeder Player hat bestimmte Eigenschaften, alle zusammen bezeichnet man als Feature Sets
- Feature Sets
  - Sampled Audio: VolumeControl, StopTimeControl
  - MIDI: VolumeControl, MIDIControl, TempControl, PitchControl, StopTimeControl

Feature Sets (2)

- Feature Sets:
  - ToneSequence: ToneControl (muss sein), VolumeControl und StopTimeControl (kann sein)
  - Interactive MIDI: MIDIControl (muss sein)
  - Video: VideoControl (muss sein)

Mobile Media API-RI

- Wurde von Sun speziell für CLDC/MIDP Software entwickelt
- In Java und native C-Code geschrieben
- Verwendung von C-Code für:
  - Native Codecs
  - Audio Output
  - Video Display
  - FloatingPoint-Berechnungen

CLDC = Connected Limited Device Configuration
MIDP = Mobile Information Device Profile