Mobile Systeme
Grundlagen und Anwendungen standortbezogener Dienste

Location Based Services in the Context of Web 2.0

Department of Informatics - MIN Faculty - University of Hamburg
Lecture Summer Term 2007

Dr. Thilo Horstmann

CLDC

NMEA

MIDP

Google Earth

OpenGIS

SQL

KML

Bluetooth

Mash-Ups

Web 2.0

J2ME

Loxodrome

Euler Spaces

RDMS

GPS

PostGIS

GPX

Maps

JSR 179

Polar Coordinates

API

Threads

Today: J2ME (1)

- What's special to the development of mobile devices?
- What's different to standard Java programming?
- Basic development and deployment cycle
- My first Midlet: "Hello World"

I know Java, why Java for mobile devices?

Write once, run everywhere?

Characteristics of Mobile Devices vs. Desktop

	Ordinary Cell phone	**PDA, Smartphone**	**Desktop**
CPU	12-30MHz	50-400MHz	3.x GHZ
Volatile Memory	512kB-1MB	2-64MB	> 1GB
Persistent Memory	2-4 MB (flash)	16MB-2GB (Memory Card)	120GB
Graphics	101x64 (2 bpp)	240x320 (24 bpp)	2560x1600 (32 bpp)
User Input	Number keyboard	Mini-QWERTY, Touch Screen, Wheel, Joystick	Full QWERTY, Mouse
Network	< 56 kBit/s (expensive)	< 3,6MBit/s (expensive)	Giga-Byte (cheap)
External Devices	proprietary, if any	Bluetooth, USB, proprietary	any
OS	proprietary	Symbian, Windows Mobile	Unix, Mac OS, MS

Observations and implications: Limited Device

- You will hardly develop on the mobile device itself!
  - Development Device != Target Device
- We need some means to develop on a workstation and deploy our software to the target machine
  - We need some deployment facilities (BT, serial (cable) connection), IrDA, Over the Air (OTA))
  - Cross development
- Use of simulators: Simulation of device and the environment
  - simulation of incoming calls, etc.
- How to debug? Debugging on a local machine is fine but often not sufficient

Observations and implications: Display

- Display size differ significantly on various devices
  - Often it does not make sense to simply scale down a 640x480 true color screen to 101x64 with 4 colors. You need to adopt the application
  - Small display size allows for one window (at a time) only
    - No arrangement of Windows (like on Desktop)
    - Different window management
  - Display not always visible (backlight powers off to save battery power)
- Generally, is the display the right output device?
  - Speech output, event notification using sound, vibration

Observations and implications: User input

- General Limited user input capabilities
- User interaction strongly depends on available hardware
  - keyboard, stylus pen, joystick, wheel, touch screen (apple iPhone)
  - Interfaces built for some input device may not be suitable for others
- Again, does manual user input fit into the user context?
  - Speech input output
- A mobile device is not a miniaturized PC. It will be used differently!

Observations and implications: Network

- Wireless Network
  - unreliable, connection may be disrupted at any time
  - download transfer rates may differ from 56kB/s (GPRS) up to 3.6MBit/s (UMTS/HSDPA) or even 300Mbit/s (WLAN 802.11n)
  - generally expensive to use
  - not always on du to power constraints
- When designing your networked application think carefully about using the network. Often it is cheaper (in terms of money!) to keep a connection alive rather than re-opening it again. However, sometimes it is the other way around :-(

Observations and implications: Security

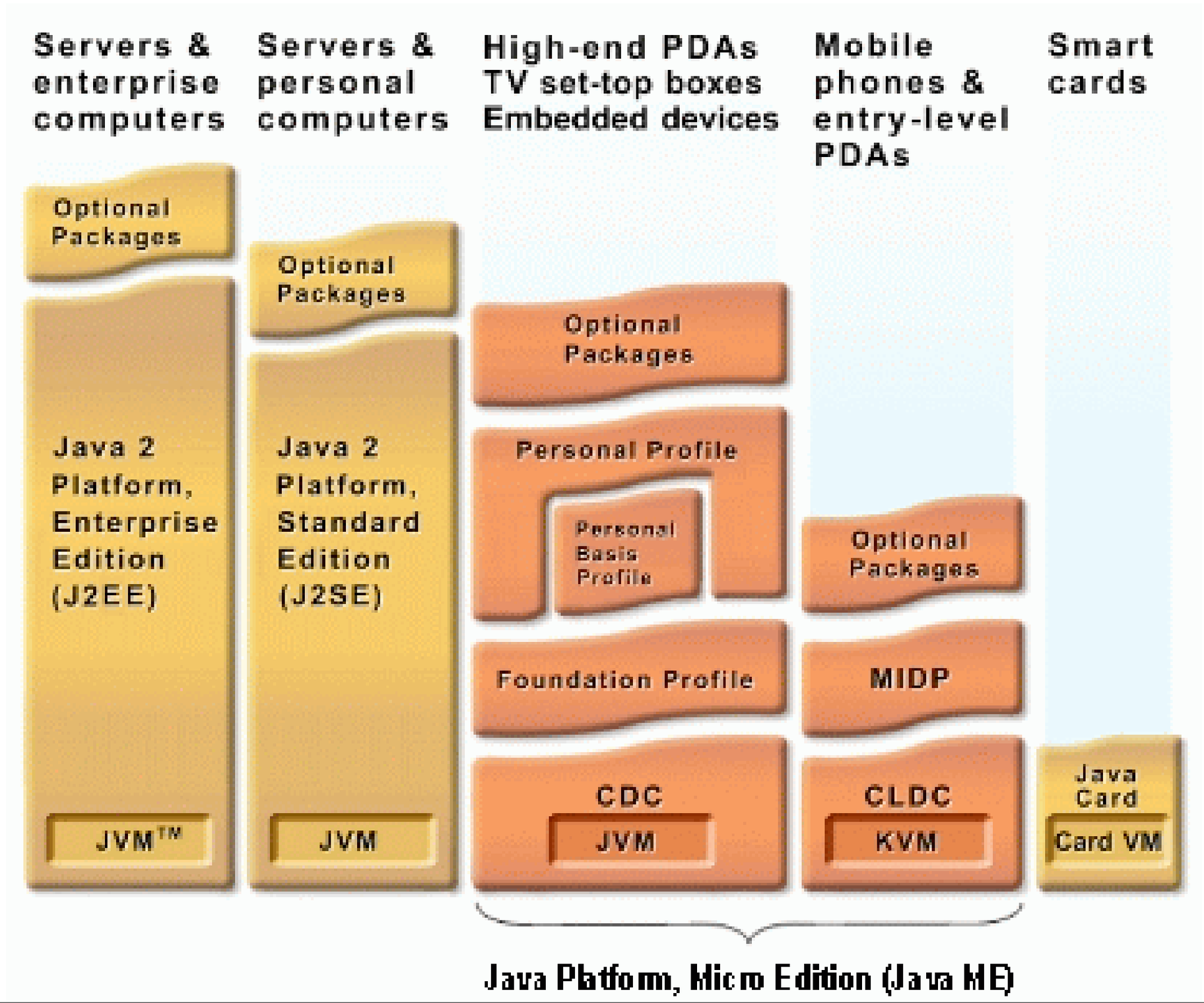
- Since the misuse of phone resources can be very dangerous in terms of costs, privacy violation, accounting issues, system stability, and others vendors often restrict the use of certain APIs.
  - phone API, Network API, Bluetooth
- Vendor / model / carrier / firmware dependent
  - An application running on one device does not necessarily pass security checks on another device
- Often, Vendor certification necessary
  - http://javaverified.com/

Main development lines

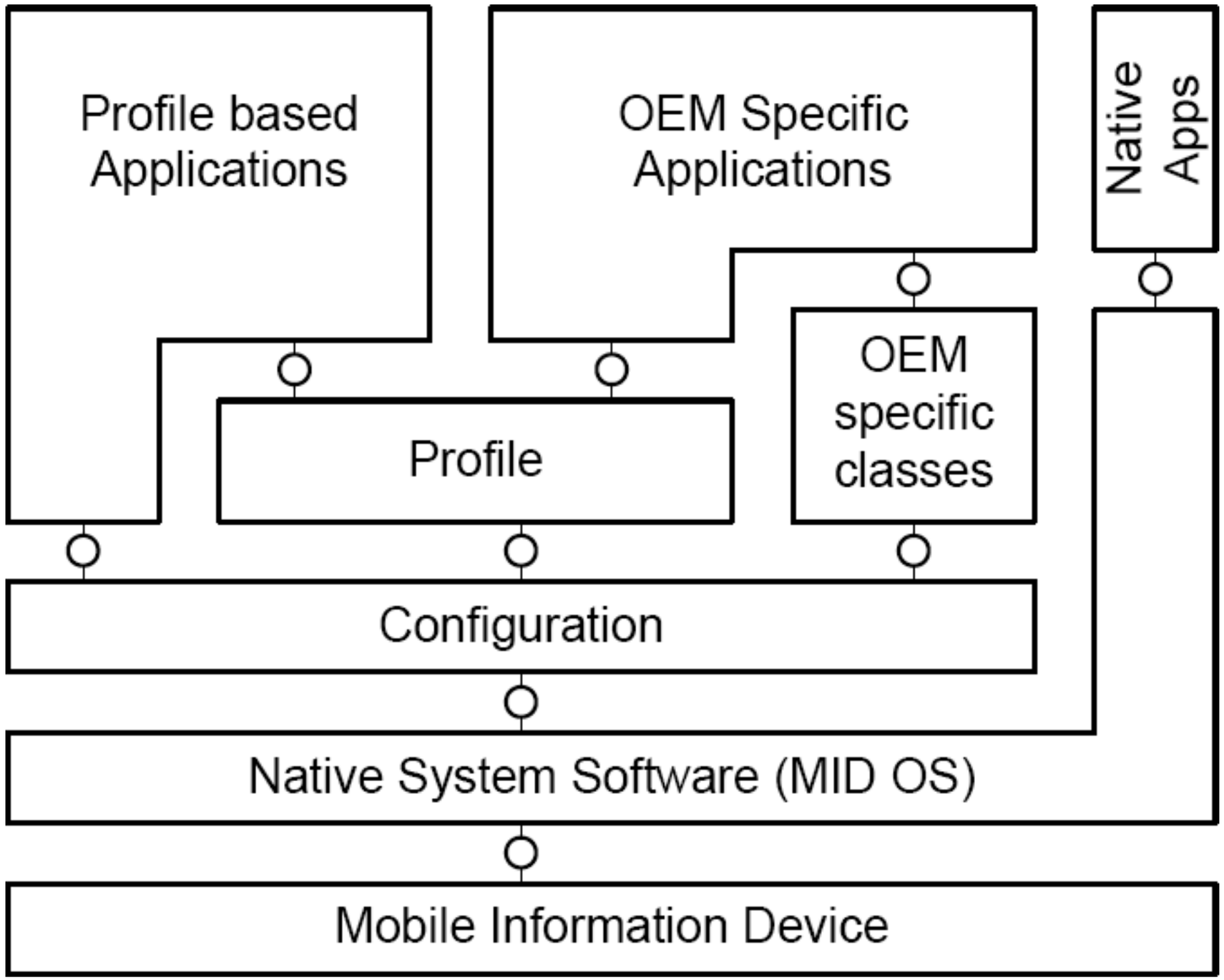
- J2ME
- .NET for Mobile
    - managed code
- Adobe Flash
- Native:
    - Symbian C++
    - C++ (unmanaged)
- Emerging
    - Scripting (open source) languages: Python for Series 60

The Java 2 Micro Edition (J2ME)

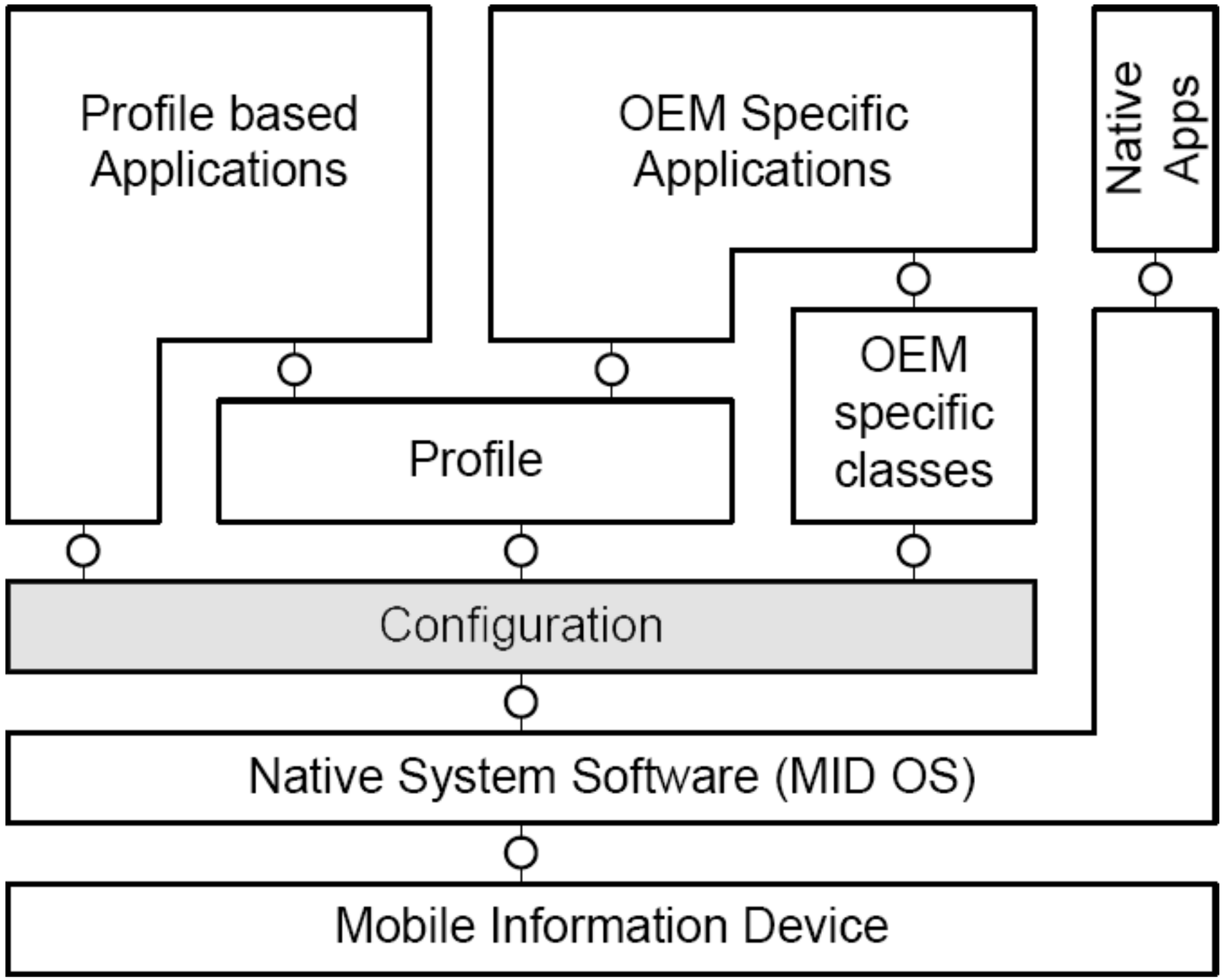
J2ME in the Java World (Source: Sun)



J2ME – Architecture



J2ME – Configuration



J2ME Configuration - Requirements

- A J2ME configuration shall only define a minimum complement or the “lowest common denominator” of Java technology. All the features included in **a configuration must be generally applicable to a wide variety of devices**. This means that the scope of the Connected Limited Device Configuration is limited and possibly incomplete for real target devices. Additional features specific to a certain vertical market, device category or industry should be defined in a J2ME profile specification.

- Since the goal of the configuration is to guarantee portability and interoperability between various kinds of resource-constrained devices, the **configuration shall not define any optional features**. This limitation has a significant impact on what can be included in the configuration and what should not be included. The more domain-specific functionality must be defined in J2ME profiles or optional packages rather than in CLDC

J2ME Configuration - Requirements (cont.)

- A J2ME configuration specification shall generally **define a subset of the Java technology features and libraries provided by the Java 2 Platform, Standard Edition (J2SE)**. Consequently, rather than providing a complete description of all the supported features, the CLDC Specification shall only define the variances and differences compared to the full Java Language Specification (JLS) and Java Virtual Machine Specification (JVMS). If something is not explicitly specified in the CLDC Specification, then it is assumed that a virtual machine conforming to the CLDC Specification shall comply with the JLS and JVMS

J2ME – Configurations

- CLDC – Connected Limited Device Configuration
- "…standards-based technologies for developing applications that run on small mobile devices"
- Currently two versions: CLDC 1.0.4 (JSR 30), CLDC 1.1 (JSR 139)
    - 1.1 is backward compatible (1.0 Applications will run on 1.1 System)
- CDC - Connected Device Configuration CDC (JSR 36)
- "…a standards-based framework for building and delivering applications that can be shared across a range of network-connected consumer and embedded devices"

CLDC 1.0 Virtual Machine

- The first generation of Java technology-enabled wireless devices was based on the K virtual machine (KVM), a reference design that demonstrated how the CLDC specification could be implemented
- KVM (CLDC 1.0):
    - the smallest possible “complete” Java virtual machine
    - hence the name K, for kilobytes (40 kilobytes to 80 kilobytes)
    - Clean and highly portable
    - C-language
    - Modular and customizable

CLDC 1.1 Virtual Machine

- CLDC HotSpot Implementation 1.0 in mid 2002 as an optimized implementation that focuses on performance and footprint
- Hotspot VM (CLDC 1.1)
    - dynamic, adaptive compiler
        - most frequently used & time-critical parts
    - optimized interpreter
        - for infrequently executed methods
    - Assembly-language
        - optimized for ARM architectures
    - 8-10x faster than KVM

J2ME – CLDC compared to J2SE

- No floating point support
  - Byte code instructions removed
  - was reintroduced in CDLC 1.1
- No Java Native Interface (JNI, calling of native c-functions)
- No user defined class loaders (security)
  - Only allowed in a trusted environment, but not in applets and J2ME
- No reflection
- No Thread groups
  - Multithreading is supported, thread groups may be implemented by application developer

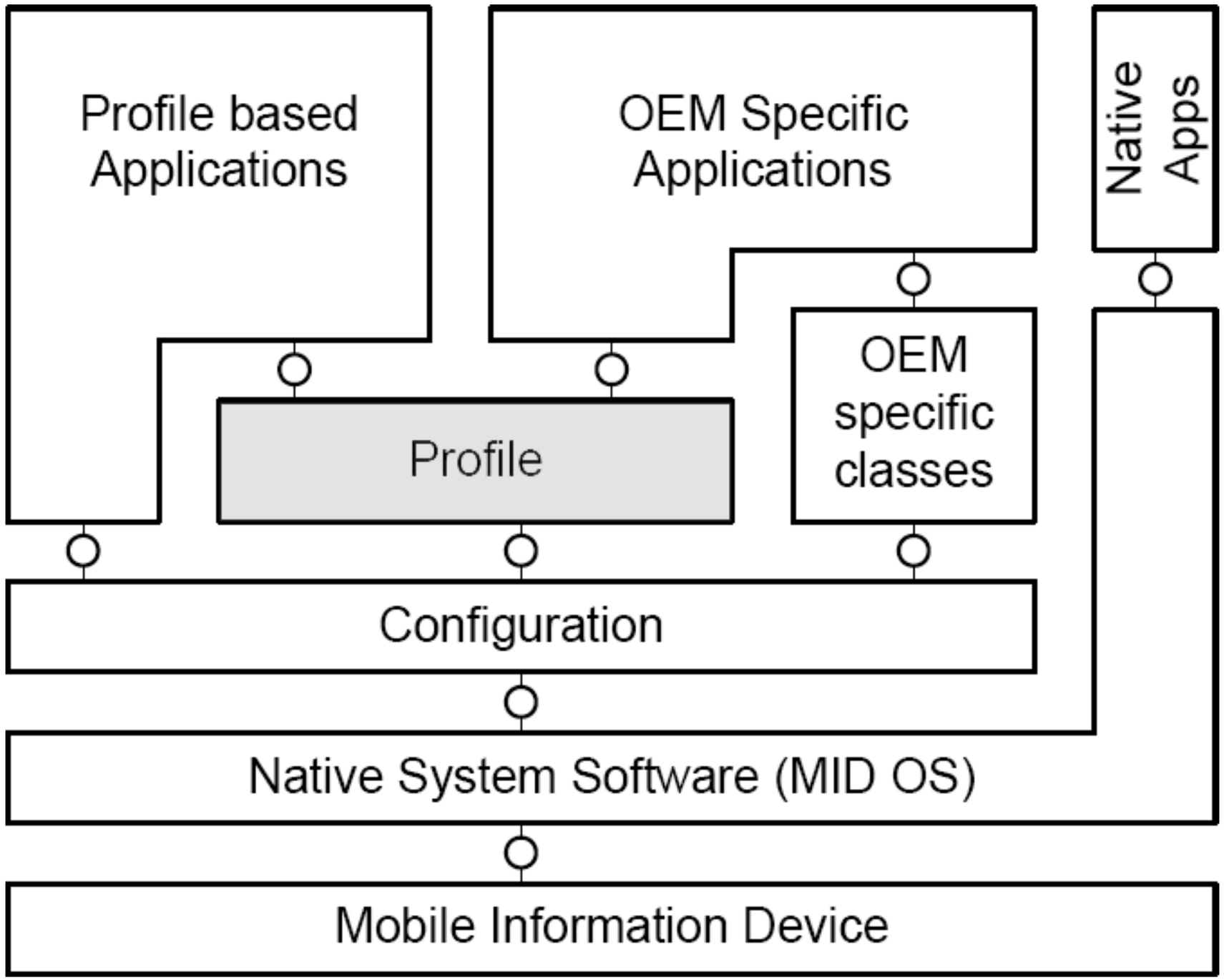
J2ME – CLDC compared to J2SE (cont.)

- No finalization
  - should be used with care anyway!
- No weak references
  - references not counting as references for Garbage collector
  - Reintroduced in CLDC 1.1
- Limited error handling - exceptions do exist though
  - Runtime errors are handled in an implementation-specific manner. The CLDC defines only three error classes: java.lang.Error, java.lang.OutOfMemoryError, and java.lang.VirtualMachineError. Non-runtime errors are handled in a device-dependent manner that involves terminating the application or resetting the device.

Packages of CLDC 1.1

- java.io
  - Provides classes for input and output through data streams.
- java.lang
  - Provides classes that are fundamental to the Java programming language.
- java.lang.ref
  - Provides support for weak references.
- java.util
  - Contains the collection classes, and the date and time facilities.
- javax.microedition.io
  - Classes for the Generic Connection framework.

J2ME - Profile



J2ME Profile

- Completes runtime environment specification
  - Specifies additional hardware constraints
  - Specifies API available additionally to that specified by Configuration
  - Intensely narrows down specification of target device
- Still even though target platform is narrow, a Profile is targeted at a whole class of devices, not at a specific platform
  - E.g. MIDP is used on all mobile phones supporting Java, even on some Smart Phones and/or PDAs (Mobile Information Devices)
- Example Profile: MIDP (Mobile Information Device Profile)

Packages contained in MIDP 2.0

- Mainly contained javax namespace
  - javax.microedition.io
  - javax.microedition.lcdui
  - javax.microedition.lcdui.game
  - javax.microedition.media
  - javax.microedition.media.control
  - javax.microedition.midlet
  - javax.microedition.pki
  - javax.microedition.rms

7 Steps to your J2ME application

Step 1: Design

- To summarize: Standard Software Engineering concepts apply to J2ME development as well, but in J2ME
    - we do not develop on the target platform. The development (e.g. PC) and target platforms (e.g. cell phone) differ significantly
    - we must take care of security constraints
        - “you do not want an application that secretly dials expensive numbers in the background”
- Although standardized, J2ME allows for vendor specific extensions and interpretations.
    - You often end up with different binaries of the same application for the different target devices. Even if it is Java!

Step 2: Create Project and Write Code

- Writing code is simple
  - All you need is your favorite text editor, Java SE, and a Java Wireless Toolkit for CLDC, e.g. one of
    - Sun Java Wireless Toolkit 2.5 for CLDC (Windows XP, Linux (alpha), not available for Mac OS)
    - Mpowerplayer SDK, platform independent (but limited)
    - Vendor specific Toolkits
- Sophisticated IDEs available: EclipseME, NetBeans + Mobility Pack
- Here we are using a plain text editor and Suns WTK 2.5

Step 3: Compile

- Using WTK or command line you can easily compile your project
  - The code will be compiled with Java SE compiler
  - Do not try to add SE libraries
  - Do not use compiler settings not suitable for J2ME
- Example: Compilation for CLDC 1.1 and MIDP 2.0 using the command line (from WTK root directory):

```
javac -bootclasspath ..\lib\cldcapi11.jar;..\lib\midpapi20.jar
<javasource file>
```

Step 4: Byte code verification

- As with any Java application the generated J2ME byte code needs to be verified
- Aim: prohibit illegal byte code or code violating type safety to be executed
- Idea: Split byte code verification into 2 phases
    - Pre-verification (after class generation) and verification on the actual device when loading the class
    - Introduction of pre-verification saves resources on the mobile device
        - due to injected attributes only a linear run to prove code integrity on device necessary
- Failure of code verification will yield security or integrity violations
- Pre-verification tool is part of WTK (bin directory)

Step 5: Package

- In contrast to J2SE applications a J2ME application consists of 2 parts:
  - The actual archive containing the class files (*.jar)
  - The application descriptor file (*.jad)
- Firstly, you create the Manifest file which describes the contents of the archive
- Secondly, The java archive is created similiar to J2SE
  - jar cvfm <yourapp.jar> yourManifest.mf .\<your package dir>
- Thirdly, you create the jad-File (e.g. using your text editor)

Example Manifest and JAD files

- MIDlet-Name: App
- MIDlet-Version: 1.0.0
- MIDlet-Vendor: SW WorldWide Inc

- MIDlet-1: App, , de.swww.AppMID
- MIDlet-Name: App
- MIDlet-Version: 1.0.0
- MIDlet-Vendor: SW WorldWide Inc
- MIDlet-Jar-URL: App.jar
- MIDlet-Jar-Size: 23454
- MicroEdition-Profile: MIDP-2.0
- MicroEdition-Configuration: CLDC-1.1

Step 6: Emulate and Test

- After the application has been correctly packed it can be locally tested using the emulator
- For testing vendor specific features or APIs you will need vendor specific emulators
- The device and the environment are emulated
  - Examples: Incoming phone call, Position update (Location API)
- Standard out / err is redirected to the WTK window
  - You see System.out.println and exception messages in the window

Step 7: Deploy

- Basically, there are 2 approaches:
  - Using a PC
    - USB cable, Bluetooth, Serial cable
    - Sometime vendor specific synchronization software necessary
  - Over the air (OTA)
    - Application will be downloaded from a web server
      - Firstly, the JAD file will be downloaded, then the actual JAR
      - Ensure, URI in JAD file and the mime-type of the JAD/JAR files are correct
    - OTA can be simulated in the WTK emulator

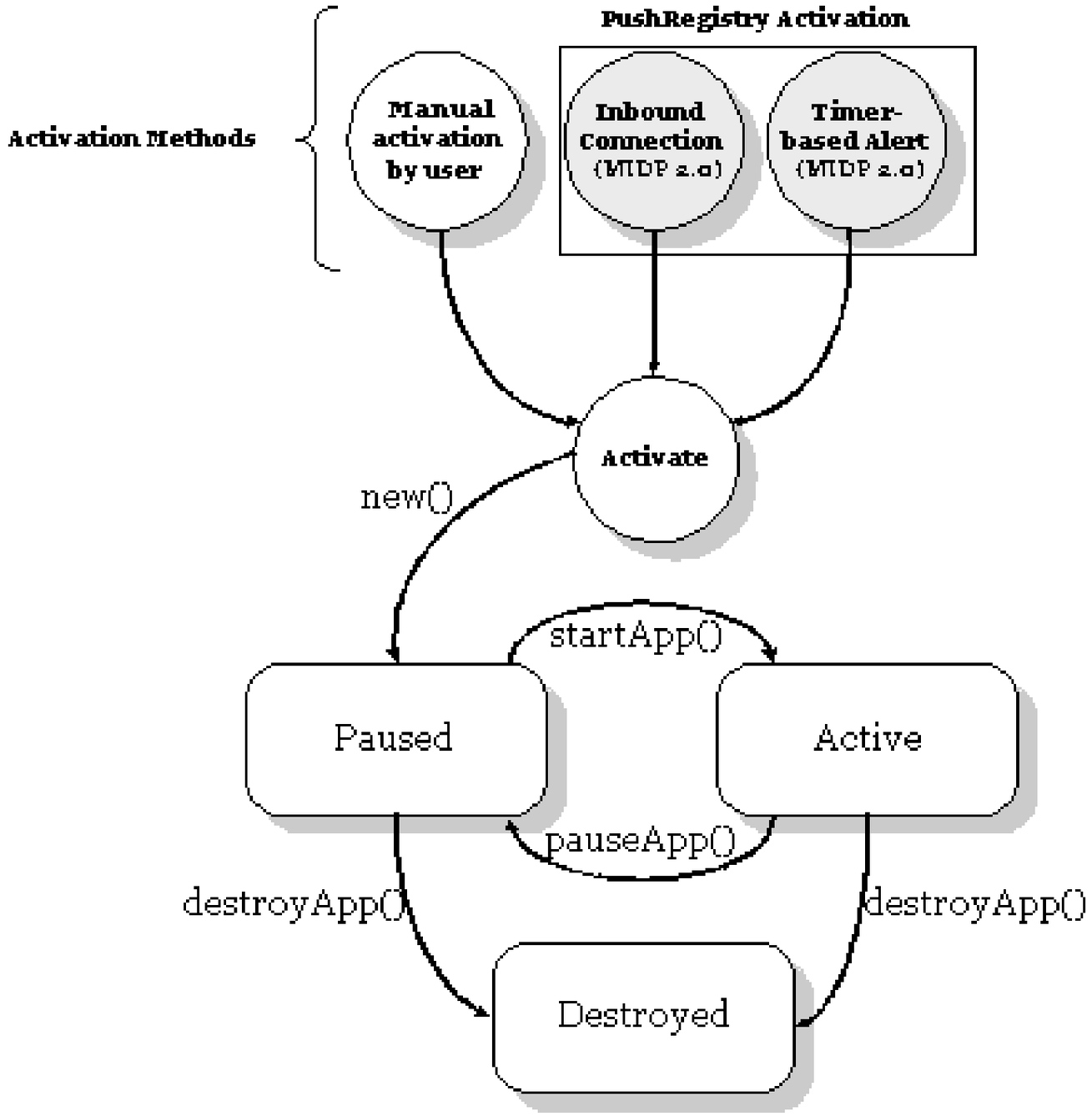
Why to sign a MIDlet suite?

- Some devices might restrict certain functionality to those applications with valid signatures.
- Some security policies reject the installation of any unsigned application.
- Other devices will warn users about an application being “untrusted” if it’s not signed.
- Security pop-ups on phones can get really annoying, and these can be lessened with signed applications.
- Ensures that no one but you can distribute his or her applications or updates to your application under your name.

Life cycle of a MIDlet

- A MIDlet is a managed application (in contrast to a regular J2SE application)
  - similar to an Java applet running in a web browser
- Each MIDlet's life-cycle is managed by the application management system (AMS), the software in the device that manages the download, installation, execution, and removal of applications and other resources on the device.
- Once the user selects the MIDlet for execution, the AMS instantiates it, which starts its life-cycle in the Paused state. The AMS then moves the MIDlet into the Active state and notifies it of this transition by invoking the MIDlet's startApp() method. Similarly, its pauseApp() and destroyApp() methods tell the MIDlet when it is paused or destroyed, respectively.

MIDLet Life Cycle (cont.)



MIDlet life cycle as a deterministic finite state machine (FSM)

- FSM = (∑, S, T, s, A), where:
- (∑) is the alphabet of symbols, application-specific, that are used by the transition function.
- (S) is the set of states. A state is a condition of the state machine at a certain time.
- (s∈S) is the start state.
- (T: sc x ∑ → sn) is the transition function. Based on the current state (sc), and an input symbol, it computes the next state (sn), and thus the transition to perform. The current state is defined as sc∈S. The next state is defined as sn∈S. The very first time, sn = s.
- (A⊆S) is the set of accept states.

Formal Definition of a MIDlet Life-Cycle Finite State Machine

- M = (∑, S, T, s, A) :
  - ∑ = {startApp, pauseApp, destroyApp}
  - S = {Paused, Active, Destroyed}
  - s = Paused. The very first time sn = s.
  - A = {Paused, Active, Destroyed}
- T: sc x ∑ → sn:
  - T (Paused, startApp) = Active
  - T (Paused, destroyApp) = Destroyed
  - T (Active, destroyApp) = Destroyed
  - T (Active, pauseApp) = Paused

The minimal J2ME application

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

public class Hello extends MIDlet{

     public Hello() {}

     public void pauseApp() {}

     public void destroyApp(boolean unconditional) {}

     public void startApp(){

        //only in WTK emulator window

       System.out.println("Hello World!");

     }

}
```

This Lecture

- Breymann, U., Mosemann, H.: Java ME – Anwendungsentwicklung für Handys, PDA und Co., Hanser, 2006, www.java-me.de
  - Chapter 1, 2, 3, 4
- The Java ME Platform
  - http://java.sun.com/javame/index.jsp
- Mobile Information Device Profile 2.0 Specification (JSR118)
  - http://jcp.org/aboutJava/communityprocess/final/jsr118/
- Connected Limited Device Configuration 1.1 (JSR 139)
  - http://jcp.org/aboutJava/communityprocess/final/jsr139/

Thank you!

Dr. Thilo Horstmann

e-mail: thilo.horstmann@gmail.com
blog: http://www.das-zentralorgan.de