Mobile Information Device Profile - Java Programs in the Phone

Kari Systä
Nokia Research Center

NOKIA

Content

Background

- Java for devices
  - why
  - what are the issues
  - development of the technology
- Java for phones
- MIDP

NOKIA

Background

The boundary between a computer and consumer device is fading,

computers are for programs and applications developed by third parties.

NOKIA

If that happens

- A huge market for software and software-based services is developing
  - phones
  - set-top boxes and other types of digital-TV receivers
- The "information device" of individuals is a mobile phone
- Homes and families are using information services through TV

- If everyday services are going to Internet, is the current PC really the optimum device for ordinary families.

NOKIA

OK, there will application market for devices

- Should it be possible to install or download new applications after the devices has been sent out from the factory
  - YES
- Can and should we assume standardized CPU architectures
  - NO, so no binary standards
- Should we invite viruses
  - NO, built in security is essential
- Should we use something existing, instead of inventing a new language and platform
  - YES
- How many programmers are familiar with
  - Lisp, Wink, Telescript, OpenTV
- **Java sounds like a reasonable option**

NOKIA

Special features of "devices"

- HW constraints
  - CPU, memory, power consumption

  ⇒optimize and implement lower layers in native

  ⇒optimize, consider what is really needed

  ⇒interact with power management, do not spend too much CPU and memory
- Should not be booted several times a day

  ⇒the platform has to be robust
- Should not need maintenance & updates weekly

  ⇒the platform has to be carefully designed and then frozen

  ⇒the platform has to be robust
- Different UI concepts than Windows

  ⇒standardize on product category level

NOKIA

Challenges for portability

The real challenge

NOKIA

Resource usage of Java

Speed

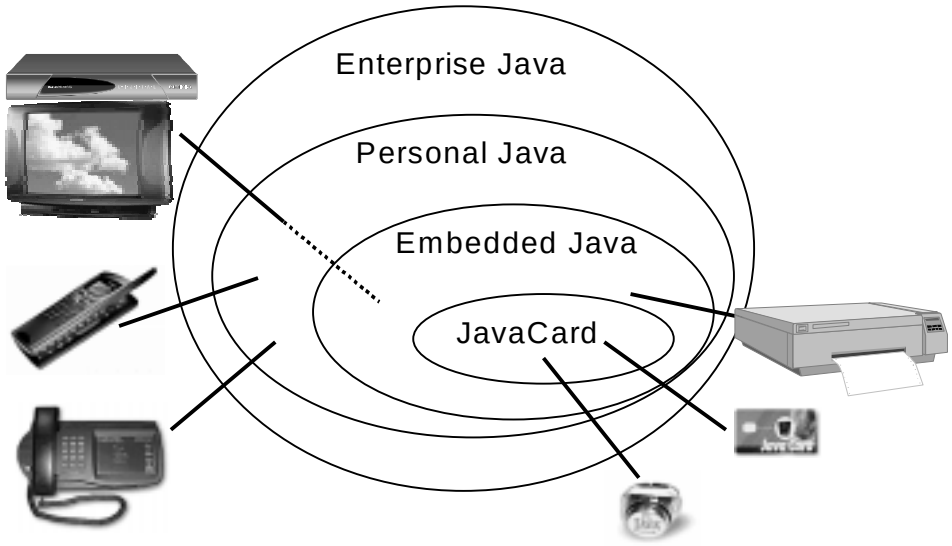
- Slower: YES
- Too slow: SELDOM
- User-noticeable slowness: downloading

Memory usage

- The platform classes are big
  - full JDK 1.2 is about 17MBytes
  - only 20 % is bytecode
- Several threads
  - several stacks - what are the bounds?
- Division of ROM and RAM need to be considered in devices

NOKIA

The first attempt to create Java “profiles”



NOKIA

JavaPhone

Additional API to be used with Personal Java

NOKIA

JavaPhone - content

- Datagram API
  - in GSM: SMS
- Access to PIM functions
  - address book
  - calendar
  - user profile
- Access to power management
  - allow application to listen and react
- Telephony API

NOKIA

JavaPhone - motivation

- Mobile Phones will not be just phones in future - rather "Wireless Information Devices"
  - communicators
  - smart phones
- An application platform is needed, that
  - provides portability over several models and manufactures,
  - easy easily accessed by software developers,
  - provides safety and robustness,
  - fit to special needs of Wireless Information Devices

NOKIA

However, JavaPhone is not ready for mass market

- Requires more than 2MBytes of ROM
- Assumes UI metaphor close to desktops

- New profiles that are fit-for purpose for small devices
  - CLDC - Connected, Limited Device Configuration
  - MIDP - Mobile Information Device Profile

- JavaPhone with (pJava => CDC) is viable for bigger devices

NOKIA

MIDP

NOKIA

MIDP Expert Group (MIDPEG)

- AOL
- DDI
- Ericsson
- Espial Group
- Fujitsu
- Hitachi
- J-Phone Tokyo Co.
- Matsushita
- Mitsubishi
- Motorola
- NEC
- Nokia
- NTT DoCoMo
- Palm
- RIM
- Samsung
- Sharp
- Siemens
- Sony
- Sun
- Symbian
- Telecordia

NOKIA

MIDP goals

- Create a common *application* development environment for two-way communication devices such as cellular phones.
- Simplicity, rather than completeness is main goal.
- Design to MID devices rather than aim at compatibility to Java2SE

NOKIA

MIDP: Goal

- Primary goals:
  - Size: must fit in “small” footprint (128K ROM)
  - Efficiency:
    - must run on low-end microprocessors
    - must run in limited heap size (32–128K RAM)
    - minimal creation of garbage
  - Time to market

NOKIA

MIDP: Scope

- Items not in MIDP scope:
  - How an application actually gets on the device
  - The end-to-end security model
  - Features specific to certain network
  - System- or OEM-specific application needs
  - Any particular implementation

NOKIA

MIDP - main components

- Important parts of java.lang, java.util
  - mainly as defined by CLDC
- timers
- simple API for persistent storage
- application of generic connection framework
- user interface

NOKIA

MIDP application framework

NOKIA

MIDlet (MIDP Application) inherits ideas from XLET

Well behaved

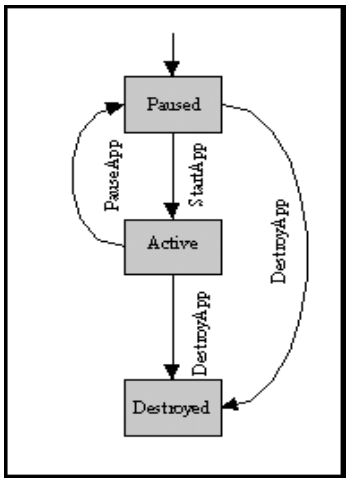
- Lives within resource constraints
- Runs and terminates when requested

MIDlet Lifecycle

- Start – acquire resources and start
- Pause – release resources and become quiescent
- Destroy – release all resources, destroy threads and end all activity

No main() .. exit()...

- Runtime.exit() and System.exit() will throw SecurityException



NOKIA

MID Application Suite

- Packaging (a Jar file)
  - Class files of the MIDlet(s)
  - Resource files
  - Manifest with application properties
- Application Descriptor
  - Configuration properties
  - Pre-download properties
    - Size, version, storage requirements

NOKIA

MIDlet Runtime Environment

- Execution Environment
  - Contents of the Jar file (MIDlet)
  - MIDP classes
  - CLDC classes
  - VM for the Java™ platform (“Java VM”)
- Application Management Software

NOKIA

Java Application Manager

- Specification recognizes that
  - devices
  - networks

  are different
- Many issues are left unspecified and up to the implementation of JAM:
  - Device specific mechanisms and policies
  - Install, removal of suite
    - Invoke individual MIDlet(s)
    - Suites are isolated from each other
    - MIDlets can share within a suite (jar file)
  - For example, loading via wireless network from web server.

NOKIA

Application programmer should know that

- Some implementations allow concurrently active MIDlets - some don't
- An application may loose:
  - display at any time
    - to native apps, other MIDlet, incoming phone call
  - CPU at any time
  - get paused
    - Application manager asks MIDlet to release resources, if it does not, it may be killed
- MIDlet may ask
  - display (foreground background)
  - paused, active

  but JAM dictates

NOKIA

A skeleton of a MIDlet

```
public class MyApp extends MIDlet {

  MyApp () {
     // good place for one-time initializations
  }

  startApp() {
     // prepare for execution,
     // take the released resources
     // this may be called several times
  }

  pauseApp() {
     // do your best to free resources
  }

  destroyApp() {
     // save state to persistent storage etc.
  }
```


NOKIA

Persistent storage

NOKIA

Persistent Storage

- Lightweight record oriented database (RMS)
  - Device independent API
  - Unique record Id for each record within a store
  - Records are arrays of bytes
  - Shared within MIDlet suite
  - Atomic update for single records
  - Support for enumeration, sorting and filtering
- Platform responsible for:
  - Integrity of data across reboots, battery changes, etc.
  - Storage in flash or other device memory

NOKIA

RMS Methods

- Record Store
  - openRecordStore, closeRecordStore, listRecordStore, deleteRecordStore, getRecordSize, getNumRecords
- Record Data
  - addRecord, deleteRecord, getRecord, setRecord, getRecordSize
- Record Selection
  - RecordEnumeration, RecordFilter, RecordCompare

Networking with Generic Connection

Networking

Client - Server

- HTTP following as per RFC 2616
- Usually connects to web servers
- Underlying protocol may be IP or non IP,
  but application does not need to know the difference

Can be supported across range of devices and wireless networks

Gateway provides transport to wired network

NOKIA

Networking Methods

- Extends CLDC Generic Connection Framework
- HttpConnection
  - get/setRequestProperty
  - get/setRequestMethod
  - getResponseCode, getResponseMessage
  - getHeaderField
    - Supports integers, dates, and strings
  - Parsed URL components
    - getURL, getHost, getPort, getFile, getQuery, getRef

NOKIA

And example

```
try {
  StreamConnection c =
     (StreamConnection)Connector.open(
                       theURL, Connector.READ_WRITE);
  InputStream is = c.openInputStream();
  DataInputStream dis = new DataInputStream(is);
  status = dis.readInt();
 } catch ...
```


NOKIA

TIMERS

NOKIA

Timer classes

- Applications can delay or schedule activities for a later time with **Timer** and **TimerTask** classes, including functions for:
  - one-time execution
  - repeated execution at regular intervals
- **Timer** handles queuing and delivery

```
public void schedule (TimerTask task,
                      Date time)
```

- abstract class **TimerTask** implements **Runnable**

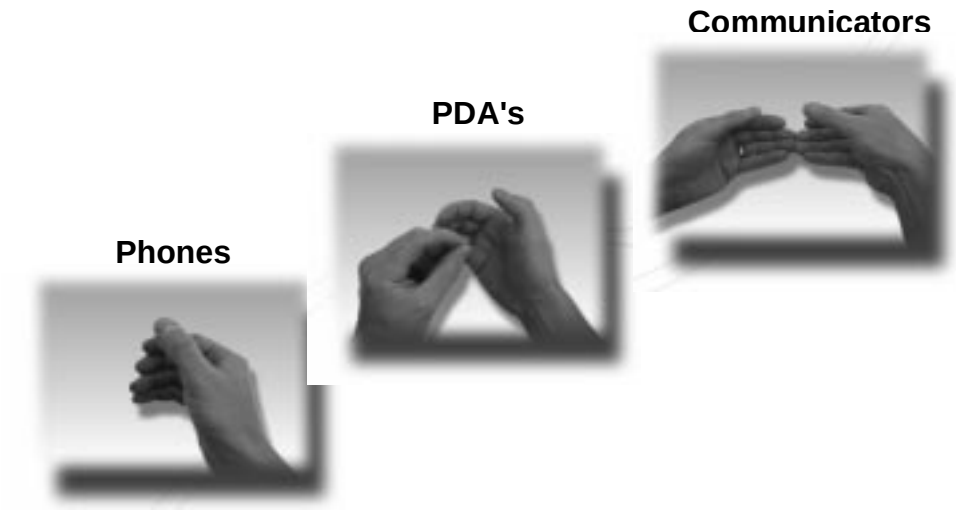
```
public void run() {
  if (System.currentTimeMillis() -
          scheduledExecutionTime()
      >= MAX_TARDINESS)
    return; // Too late; skip.
  // Perform the task
}
```


NOKIA

User interface

NOKIA

UI: Ergonomic foundations



NOKIA

Design principles

- Usable in all devices
  - Majority of wireless devices are one-hand operated
- Keep target devices in mind
  - Small screen (tens of pixels x tens of pixels)
  - Not all devices have a pointing device
- Think of end users:
  - Consumer products - not computers
  - Unified user interface
    - MIDP applications should behave consistently with resident functionality

NOKIA

Two layers

- High-level API for high portability
  - Applications should be runnable in all devices
  - Applications should be usable in all devices
  - No direct access to device features
    - Colors, screen sizes, input devices
- Low-level API for applications like games
  - Drawing primitives
  - Key events
  - Developers may compromise portability for better game-experience

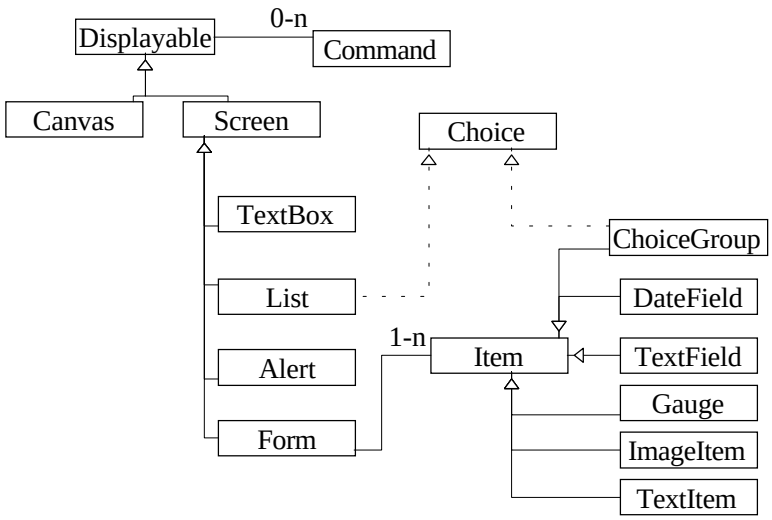
NOKIA

Screen-based design

- Applications should be based in simple screens
- A screen should contain minimum amount of information
  - Usually only one "thing"
- Simple interaction
  - No complex traversing, scrolling and selection
- Some screens are actually widgets (TextBox, List)

NOKIA

The class hierarchy (important classes only)



NOKIA

Input handling

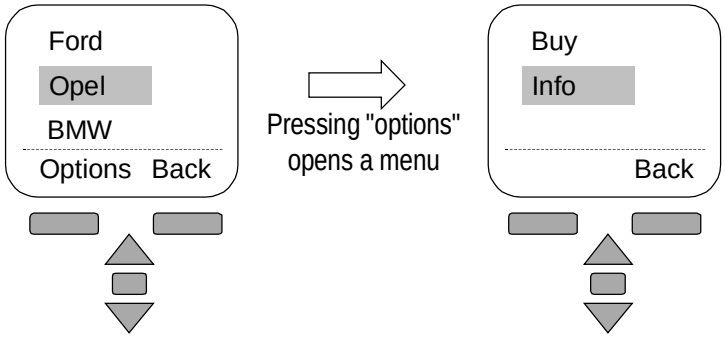
- Abstract commands in the high-level API
  - Instead of direct access to soft button (different devices may have different number of soft buttons)
  - Implementation maps the commands to soft buttons or menu items
  - Application can provide semantic hints (like back)
- Key events in the low-level API

NOKIA

Commands - example

- Consider commands:

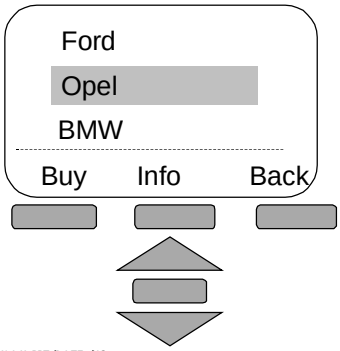
```
new Command("Buy", Command.SCREEN);
new Command("Info", Command.SCREEN);
new Command("Back", Command.BACK);
```



NOKIA

… in the case of three soft buttons

```
new Command(”Buy", Command.SCREEN);
new Command("Info", Command.SCREEN);
new Command("Back", Command.BACK);
```



NOKIA

Low-level events

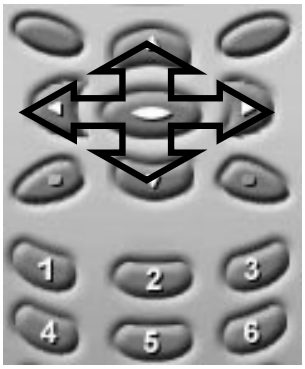
- Keys

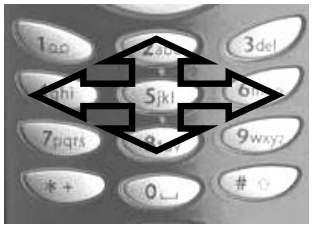
```
KEY_NUM0, KEY_NUM1, KEY_NUM2, KEY_NUM3,
KEY_NUM4, KEY_NUM5, KEY_NUM6, KEY_NUM7,
KEY_NUM8, KEY_NUM9, KEY_STAR, KEY_POUND
```

  - Devices may notify applications about other keys as well, but applications relying on those are not portable
- Abstract game events
  - Actions that the device can map either to 4-way scrolling keys or to numeric keys
  - **UP, DOWN, RIGHT, LEFT, FIRE, GAME_A, GAME_B, GAME_C, GAME_D**

NOKIA

Possible ways to map game actions





NOKIA

Example usage

```
class TetrisCanvas extends Canvas {
  int leftKey, rightKey, downKey, rotateKey;
   void init (){
      leftKey = getKeyCode(LEFT);
      rightKey = getKeyCode(RIGHT);
      downKey = getKeyCode(DOWN);
      rotateKey = getKeyCode(FIRE);
   }
   public void keyPressed(int keyCode) {
     if (keyCode == leftKey) {
        moveBlockLeft();
     } else if (keyCode = rightKey) {
      ...
   }
}
```


NOKIA

Repainting scheme in low-level API

- *Screens* are repainted automatically, but for *Canvas* a user-defined method *paint()* is called
  - paint() may have a clipping region set
- The application can request repainting by calling *repaint().*
  - Actual painting is done asynchronously and several repaint requests may be combined.
- Application may synchronize to repainting by calling *serviceRepaints()* or *callSerially().*

NOKIA

Examples of repaint-handling

```
// move coordinates of box
  x1 = box.x; box.x = x2;
  y1 = box.y; box.y = y2;
// ensure old region repainted (with background)
  canvas.repaint(x1,y1, wid, ht);
// make new region repainted
  canvas.repaint(x2,y2, wid, ht);
// force painting
// (note: application must not hold locks)
  canvas.serviceRepaints();
// application can now assume that screen
// is updated
```


NOKIA

Synchronization with callSerially

```
class Animation extends Canvas implements Callable {

  // paint the current frame
  void paint(Graphics g) { ... }
  void startAnimation() {
    // set up initial frame
      repaint();
      callSerially(this);
   }

  // called after previous repaint is finished
   void call() {
     if (/* there are more frames */){
       // update to the next frame
       repaint();
       callSerially(this);
     }
   }
}
```


NOKIA