Best Practices in UI Design and Programming on Nokia Platforms

Srikanth Raju, Jarmo Lahtinen, Nitin Mittal

Forum Nokia
Nokia
http://www.forum.nokia.com

TS-1281

Presentation Goal

Learn the UI APIs and how to use them to develop effective applications on Nokia Series 40, S60, and Series 80 platforms

Agenda

Background

Overview of UI APIs

Series 40, S60, Series 80 Platform Notes

Best Practices in MIDP Programming

Summary

Demos

Agenda

Background

Overview of UI APIs

Series 40, S60, Series 80 Platform Notes

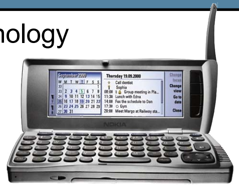
Best Practices in MIDP Programming

Summary

Demos

Platforms and Editions

Editions

	Series 40	S60	Series 80
First Edition	MIDP 1.0 CLDC 1.0 WMA 1.0 MMAPI 1.0 	MIDP 1.0/ CLDC 1.0 WMA 1.0 MMAPI 1.0 	PersonalJava™ technology JavaPhone™ API 
Second Edition	MIDP 2.0 CLDC 1.1 WMA 1.0 MMAPI 1.0 BTAPI (No OBEX) 	MIDP 2.0 CLDC 1.0 WMA 1.0 MMAPI BTAPI (No OBEX) 	MIDP 2.0/CLDC 1.1 CDC with Personal Profile WMA 1.0 MMAPI BTAPI (No OBEX) FC and PIM
Third Edition	MIDP 2.0/CLDC 1.1 Java™ Technology for the Wireless Industry WMA 1.1 MMAPI Bluetooth API (with OBEX API) Mobile 3D Graphics API FileConnection and PIM API 	MIDP 2.0/CLDC 1.1 Java Technology for the Wireless Industry Web Services API SATSA API Location API SIP API Mobile 3D Graphics API WMA 2.0 FileConnection and PIM API Bluetooth API (with OBEX API) 	

Agenda With Section Highlights

Background

Overview of UI APIs

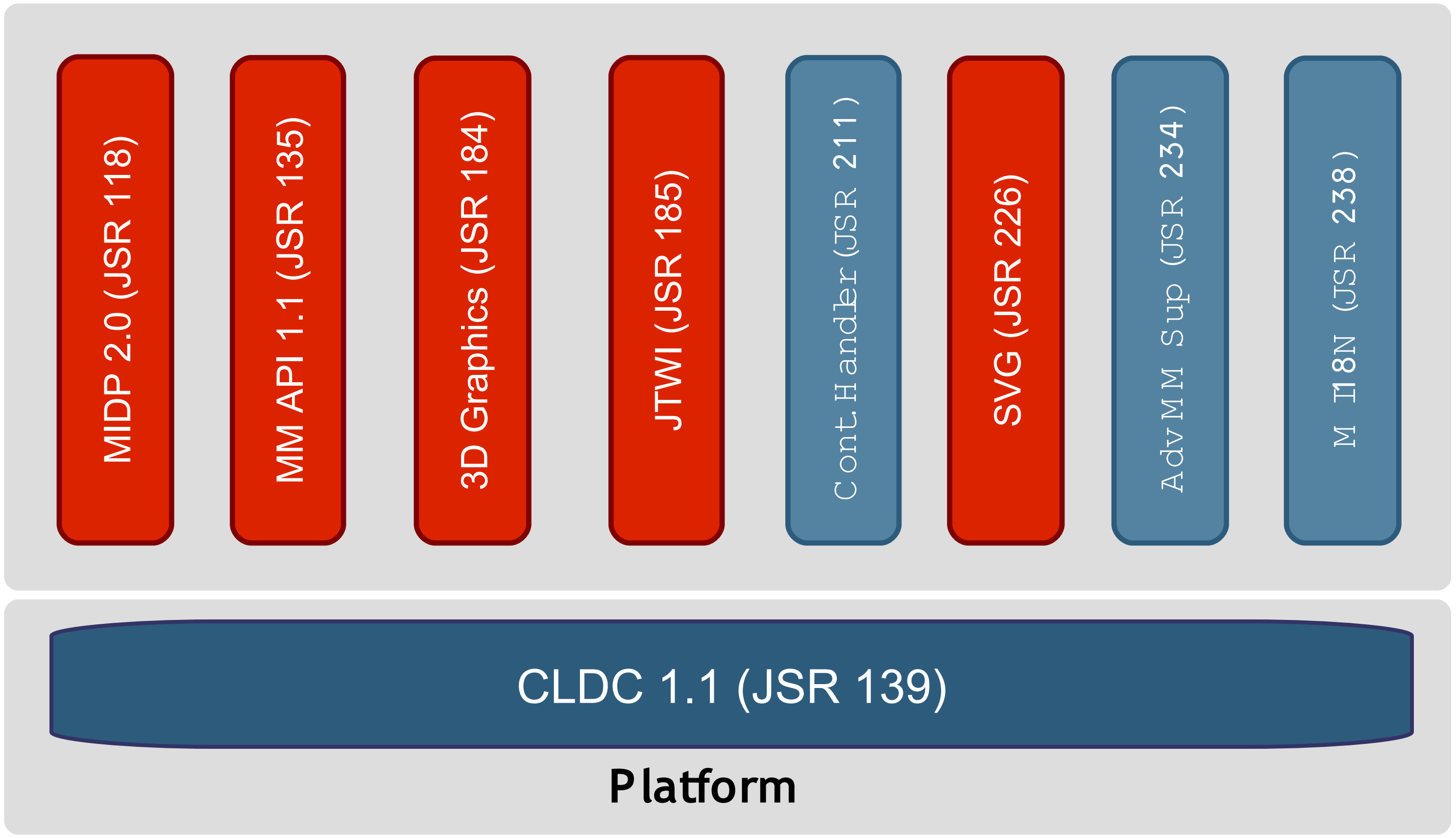
Series 40, S60, Series 80 Platform Notes

Best Practices in MIDP Programming

Summary

Demos

User Interface APIs—Today



🟥 Implemented in Nokia Devices 🟦 Not Yet Implemented in Nokia Devices

Java Specification Requests (JSR) With UI Relevance

- Mobile Information Device Profile 2.0 (JSR 118)
  - Enhanced game package with Sprite, TiledLayer classes, etc.
  - Enhanced `javax.microedition.lcdui` package
- Mobile Media API (JSR 135)
  - Ability to play/record media files—Both audio/video
  - Various codecs support
- Mobile 3D Graphics API for J2ME™ (JSR 184)
  - Designed similar to Java 3D™ API
  - OpenGL–ES-based
- Scalable 2D Vector Graphics API for J2ME™ (JSR 226)
  - Scalable vector graphics for resource constrained devices
  - Already implemented in Nokia series 40 device
- etc.

Agenda With Section Highlights

Background

Overview of UI APIs

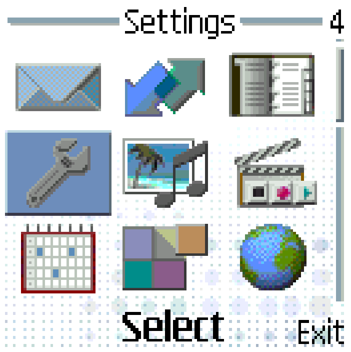
Series 40, S60, Series 80 Platform Notes

Best Practices in MIDP Programming

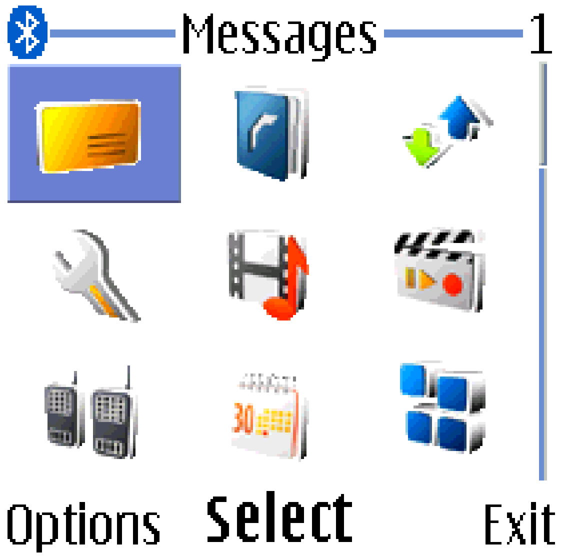
Summary

Demos

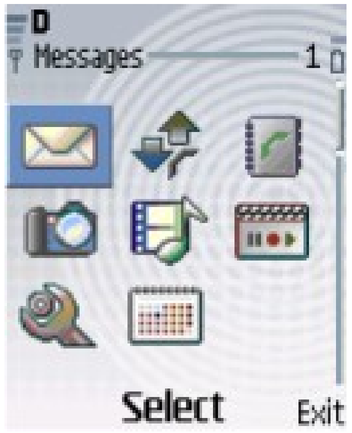
Series 40 UI Style



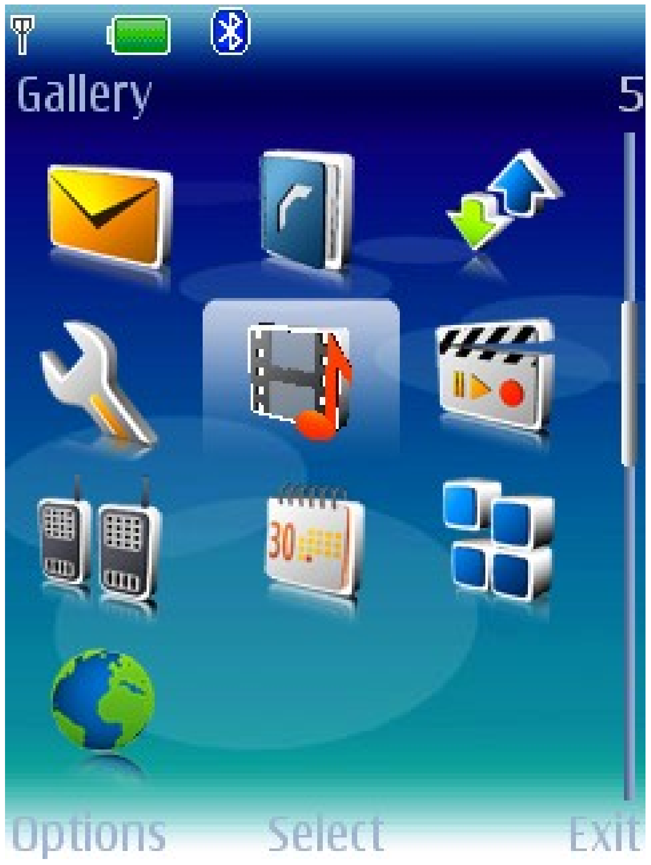
128x128



208x208

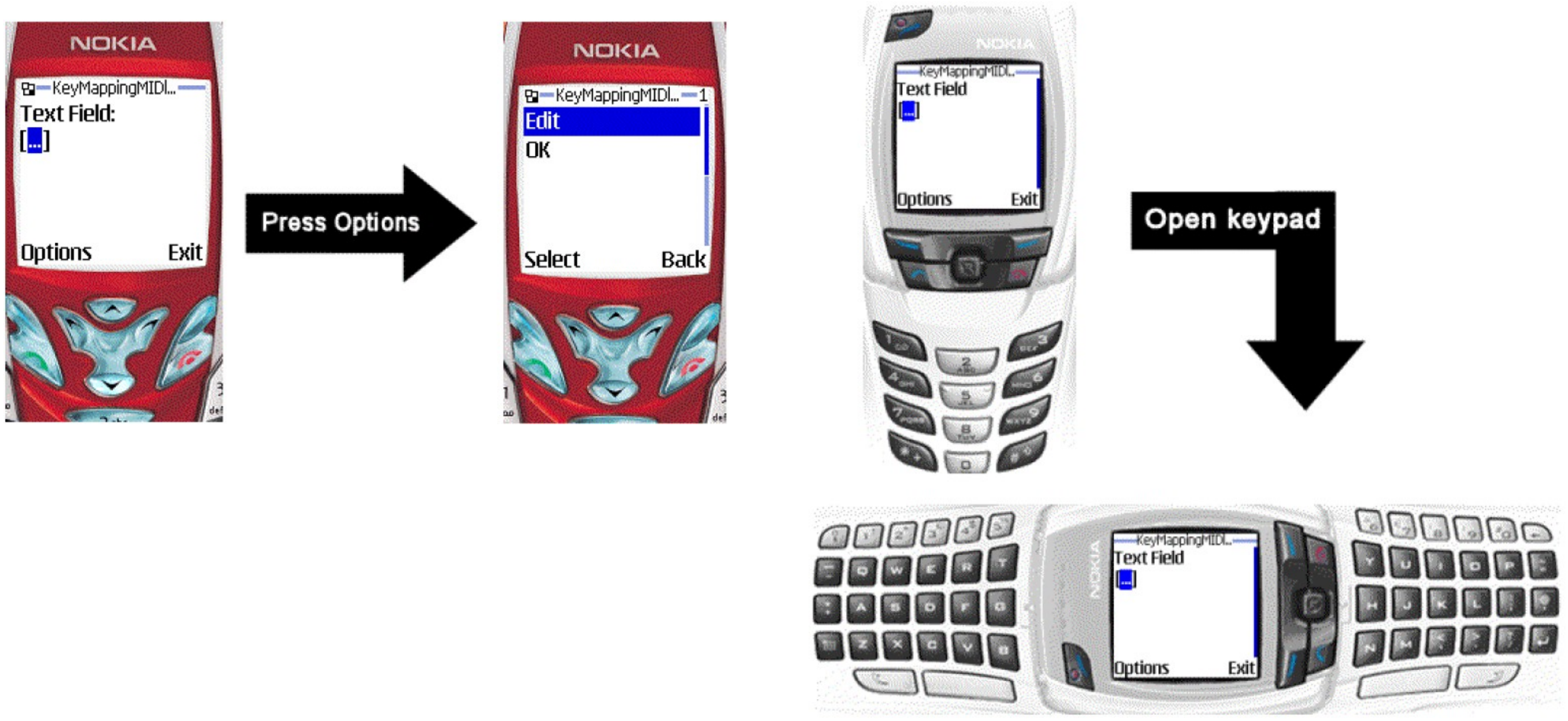


128x160

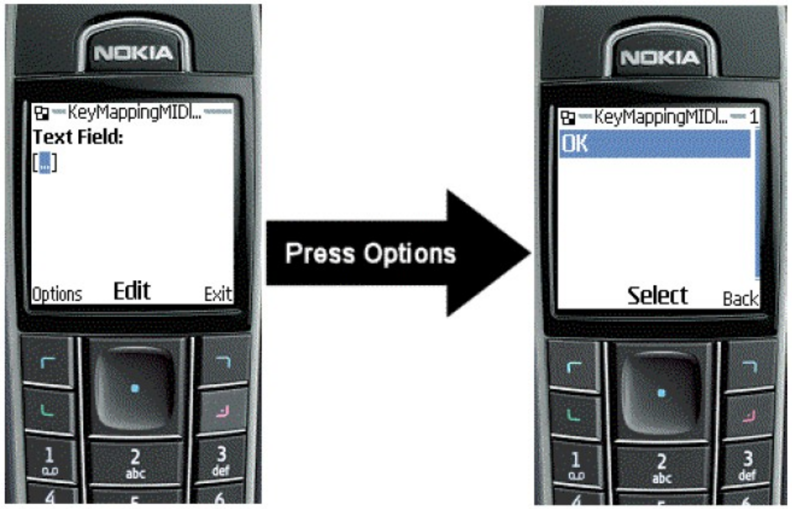


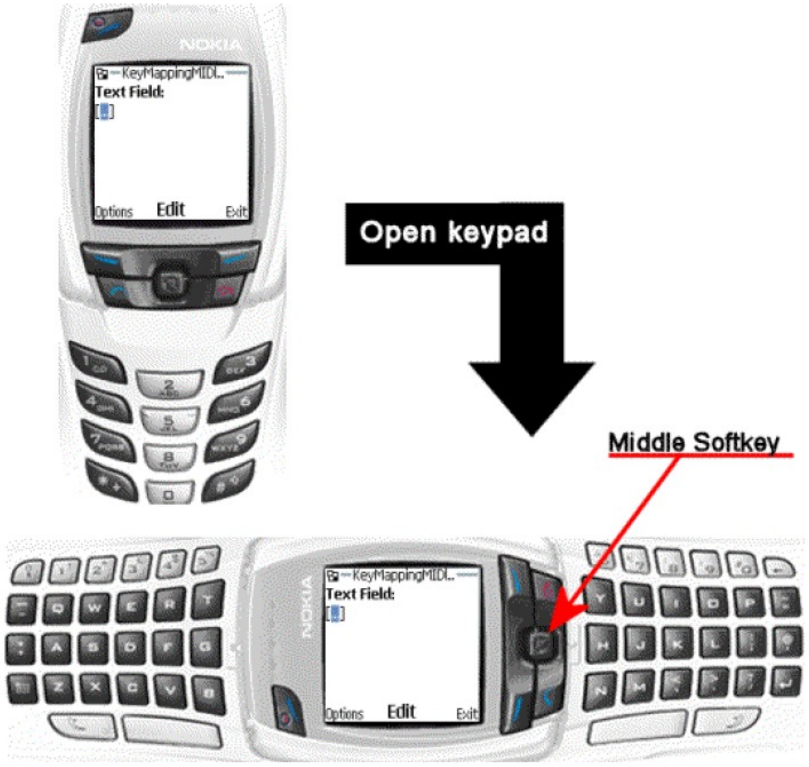
240x320

Two Soft Keys (Normal Pad and Foldout)

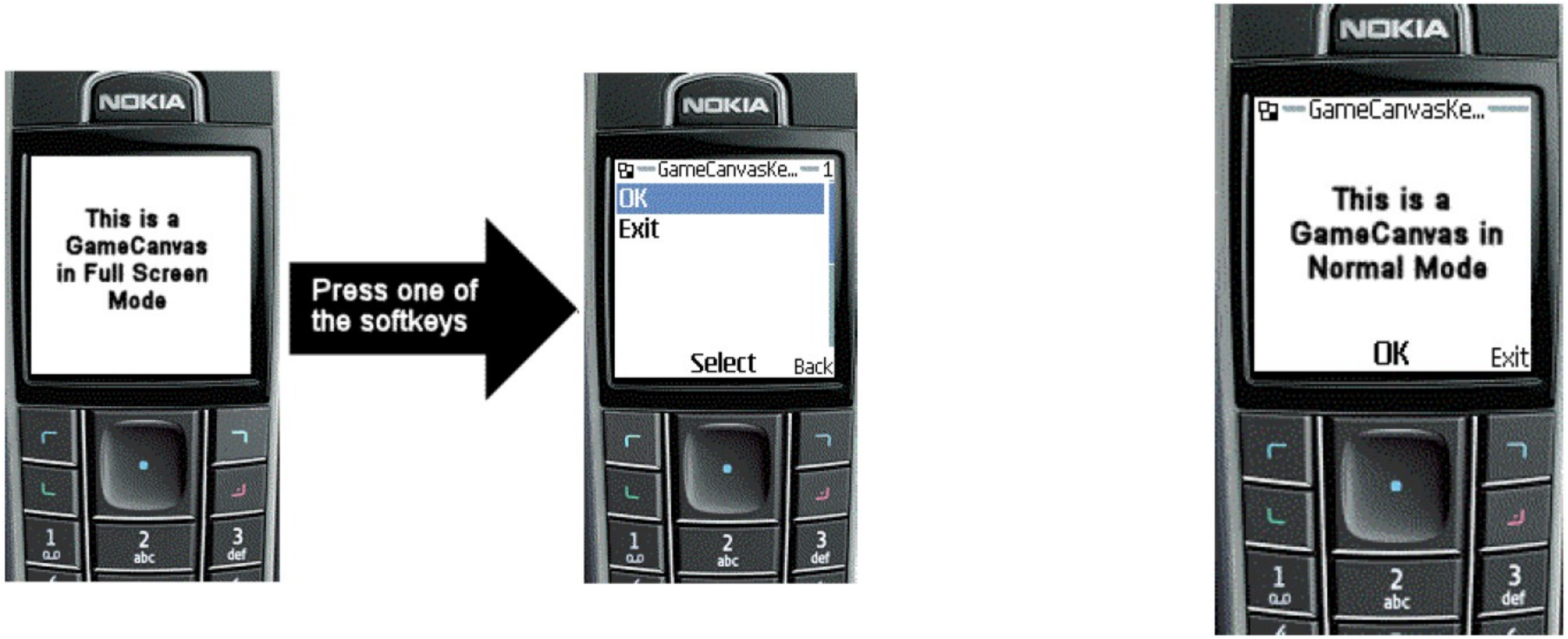


Three Soft Keys (Normal Pad and Foldout)





Softkeys in Full Screen and Normal Modes (`GameCanvas` in MIDP 2.0)

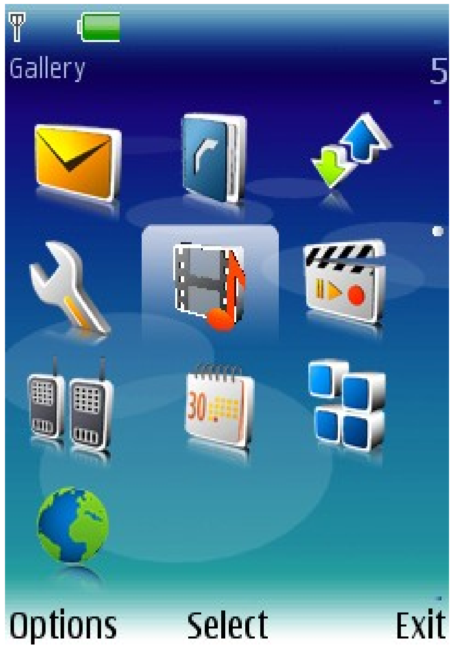

This is a GameCanvas in Full Screen Mode

Press one of the softkeys

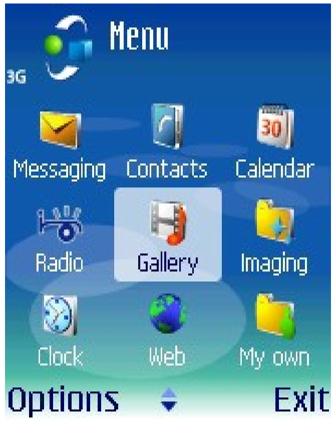
This is a GameCanvas in Normal Mode

Series 40 Third Edition: What's New

- New APIs: Mobile 3D graphics API for J2ME (JSR 184) and PDA optional packages for the J2ME platform (JSR 75)
- Series 40 third edition devices
  - N6136, N6165, N6265/N6265i, N6270, N6280, N7370, etc.
- Consistent UI with the latest Series 60 devices (Nseries devices)
  - Active standby buttons
- New screen size 240x320 pixels
  - Earlier sizes 128x128, 128x160 and 208x208 pixels
  - Canvas when FullScreenMode = false: 240x250 pixels
- MiniSD card support
- Maximum RecordStore size has been earlier 32k, now it has been increased
- MIDlets supported on memory card
- Automatic clearing of Canvas, when not in FullScreenMode



Series 40 Third Edition 320x 240 pix



S60 Second FP2 176 x 208 pix

S60 Platform Java Technology Capabilities

- S60 platform, first edition
  - Symbian OS 6.1
  - Available resolution 176x208 pixels
  - Java-based APIs
    - CLDC 1.0 and MIDP 1.0
    - Wireless messaging API (JSR 120)
    - Mobile media API (JSR 135)
- S60 platform, second edition
  - Symbian OS 7.0s
  - Available resolution 176x208 pixels
  - Java-based APIs:
    - CLDC 1.0/1.1 and MIDP 2.0
    - Wireless messaging API (JSR 120)
    - Mobile media API (JSR 135)
    - Java APIs for Bluetooth (JSR 82)

S60 Platform Java Technology Capabilities (Cont.)

- S60 Second Edition Feature Pack 1 (2.1)
  - New: Mobile Media API (MMA) 1.1
- S60 Second Edition Feature Pack 2 (2.6)
  - Symbian OS v8.0a
  - New: CLDC 1.1, FileConnection and PIM APIs, Mobile 3D Graphics API
- S60 Second Edition Feature Pack 3 (2.8)
  - Symbian 8.1a
  - New: WMA 1.1, MMAPI enhancements, JTWI, Web Services API, Bluetooth API with OBEX API
- Available resolutions: 176 x 208 and 352 x 416

S60 Platform Java Technology Capabilities (Cont.)

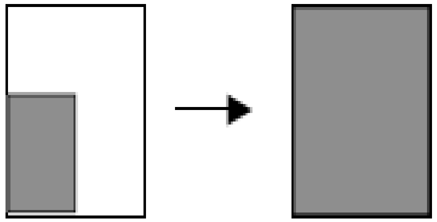
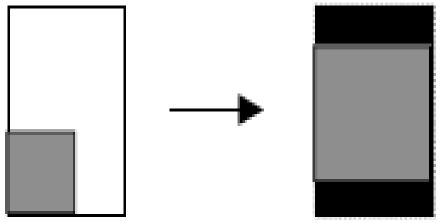
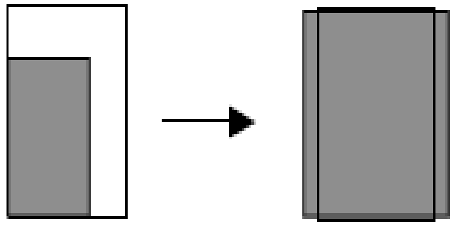
- S60 platform, third edition
  - Symbian OS 9.1
  - Java APIs:
    - CLDC 1.1
    - MIDP 2.0
    - Mobile Media API (JSR 135)
    - Java APIs for Bluetooth (JSR 82)
    - J2ME Web services specification (JSR 172)
    - Location API for J2ME (JSR 179)
    - Security and trust services API for J2ME (JSR 177)
    - Wireless messaging API 2.0 (JSR 205)
    - SIP API for J2ME (JSR 180)
  - Scalable UI with landscape/portrait orientations
  - Additional available resolution: QVGA (320 x 240)

S60 3.0 UI Changes

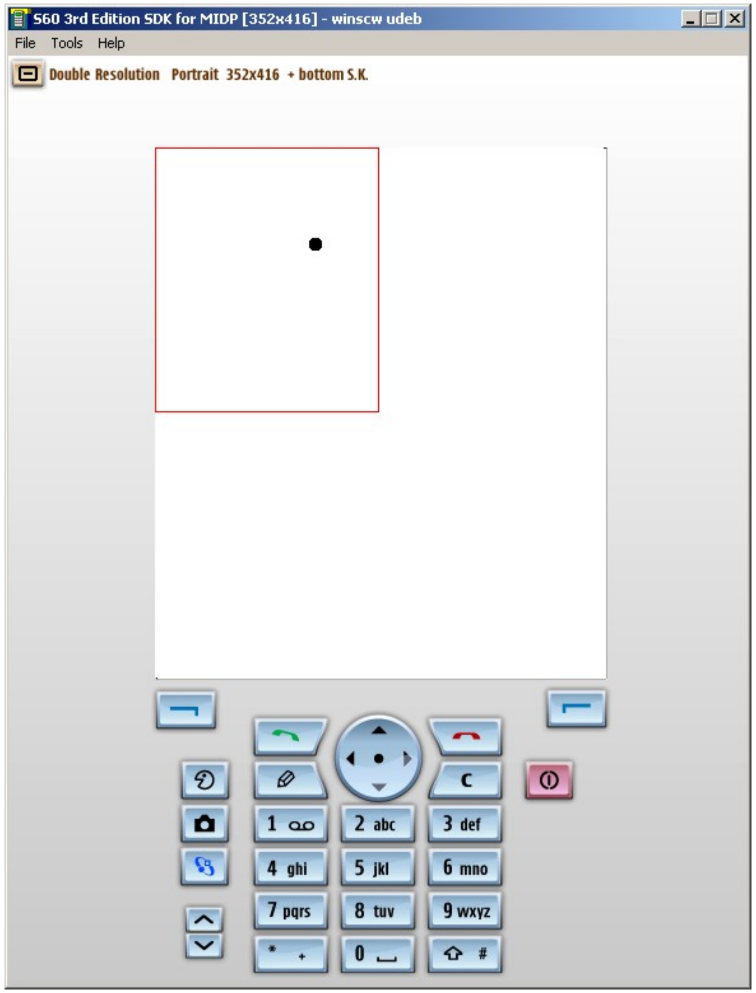
- The latest S60 devices have several display resolutions, therefore it is necessary to enable scaling of MIDlets
- This is done by using `Nokia-MIDlet-Original-Display-Size` and `Nokia-MIDlet-Target-Display-Size` parameters
- All pixel coordinates and sizes in all classes function as if the device’s display resolution were the resolution defined in the scaling attribute Nokia-MIDlet-Original-Display-Size

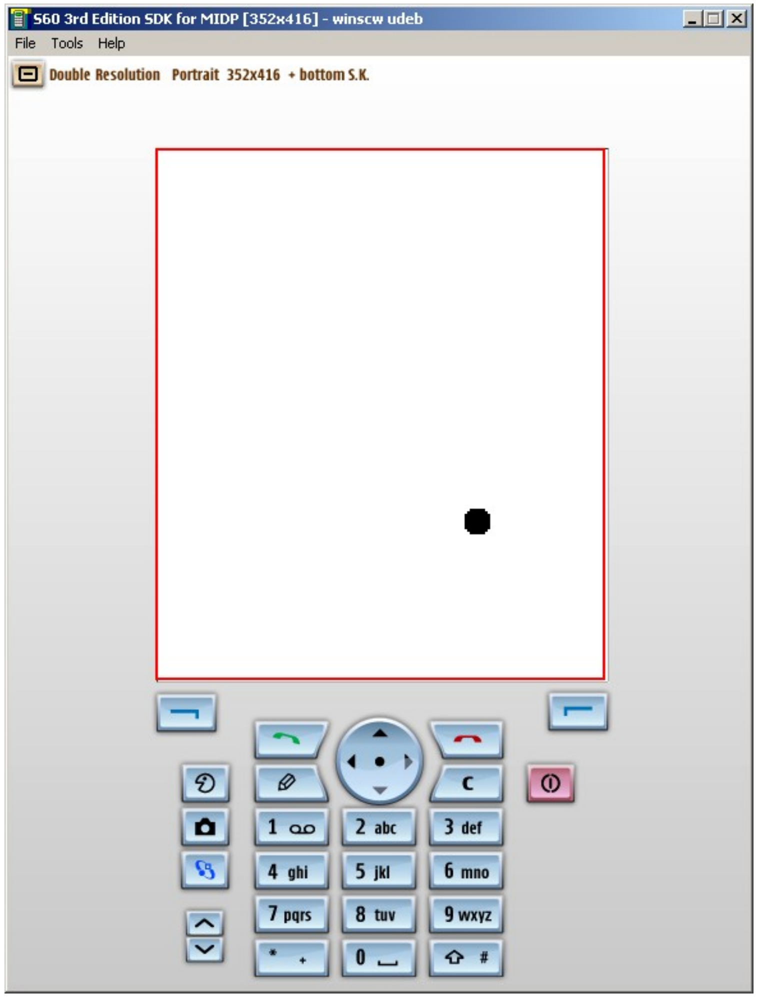
S60 3.0 UI Changes (Cont.)

Examples:

Use case	Attribute values	Referential result on the display
A 176x208 game is scaled to full-screen 352x416. *(Note that, the developer gets the same result even without defining the target display size.)*	Nokia-MIDlet-Original-Display-Size: 176,208 (Nokia-MIDlet-Target-Display-Size: 352,416)	
A 128x128 game is scaled to portrait QVGA 240x320, leaving black borders on top and bottom sides (40 pixels each). *(Note that, the developer gets the same result even without defining the target display size.)*	Nokia-MIDlet-Original-Display-Size: 128,128 (Nokia-MIDlet-Target-Display-Size: 240,320)	
A 176x208 game is scaled to full-screen portrait QVGA so that the aspect ratio is changed AND a few pixels are dropped from the left and right. The target resolution could be for example 256x320, so that eight pixels are lost from both sides.	Nokia-MIDlet-Original-Display-Size: 176,208 Nokia-MIDlet-Target-Display-Size: 256,320	

Scalable UI Example Application


S60 3rd Edition SDK for MIDP [352x416] - winscw udeb
File Tools Help
Double Resolution Portrait 352x416 + bottom S.K.


S60 3rd Edition SDK for MIDP [352x416] - winscw udeb
File Tools Help
Double Resolution Portrait 352x416 + bottom S.K.

DEMO

S60 3.0 Scalable UI Demo

Series 80 MIDP UI Differences

- LCDUI differences:
  - TextBox is always shown as a dialog with the only exception of a TextBox with a null title
  - Up to four Commands can be mapped to the Command Button Area’s buttons
    - Button 1 = positive (OK, ITEM, SCREEN)
    - Buttons 2 and 3 = neutral (ITEM, SCREEN)
    - Button 4 is negative (STOP, CANCEL, BACK, EXIT)
    - Note: some Displayables might have specific rules
  - All MIDlet Commands available at one given moment always appear in the “Actions” menu

Platform Porting Best Practices

- When porting from one platform to another, consider
  - Amount of heap memory
  - Application download time
  - Size of Java Archive (JAR) file
  - Display resolution
  - User interface
  - Processing power
  - Connectivity/Networking feature
  - Other device-specific features like Funshell API for game developers

Agenda With Section Highlights

Background

Overview of UI APIs

Series 40, S60, Series 80 Platform Notes

Best Practices in MIDP UI Programming

Summary

Demos

MIDP Application Development Phases

- Design time
  - Realize if the game is single- or multi-user
  - If multi-user:
    - Decide/Realize round-robin, turn-based, simultaneous
    - Is a game server needed or can things happen peer-to-peer
- Development time
  - Use the right tools and techniques
    - Example: built-in classes in MIDP2.0 vs. hand coding as discussed in this section
- Deployment time
  - Optimize foot print (discussed later)

Sprites—MIDP 1.0 Style

- No `Sprite` object in MIDP 1.0
- You can create a Sprite by loading an image from a resource—`Image.CreateImage(<.png>)`
- Need multiple images for animation
- Need to calculate collision detection manually
- Need to calculate movement manually

Sprites in MIDP 2.0

- MIDP 2.0 introduces a Sprite class that can be used as follows

```
Sprite sObj = new Sprite( image, 10, 10 );
sObj.move( -1, 0 );  // Simulate movement
sObj.paint( g ); // draw the sprite
sObj.collidesWith( otherObj, false );
```

Handling User Input

- In MIDP 1.0 user input is handled using `keyPressed(),getGameAction()` methods
- In MIDP 2.0, `GameCanvas.getKeyStates()` method returns an integer with each bit representing if the key was up or down
  - Latching behavior allows for catching rapid key presses

Game Keys Handling

```
// MIDP 1.0 or MIDP 2.0
class TetrisCanvas extends Canvas
{
    void init() {}

    protected void keyPressed(int keyCode)
{
        int action =
getGameAction(keyCode);

        switch (action) {
            case Canvas.LEFT:
                moveBlockLeft();
                break;
            case Canvas.RIGHT:
                moveBlockRight();
                break;
        }
    }
}
```

```
// Another approach for MIDP 2.0
Class TetrisCanvas extends
GameCanvas
{
     void  init() {}

  while(true) {
     int keyState = getKeyStates();
     if ((keyState & LEFT_PRESSED) !
= 0) {
          moveBlockLeft();
     }
    if ((keyState & RIGHT_PRESSED) !
= 0)  {
          moveBlockRight();
}
//Draw sprite
sprite.paint(g);
//Flush off-screen buffer
flushGraphics();
}
```

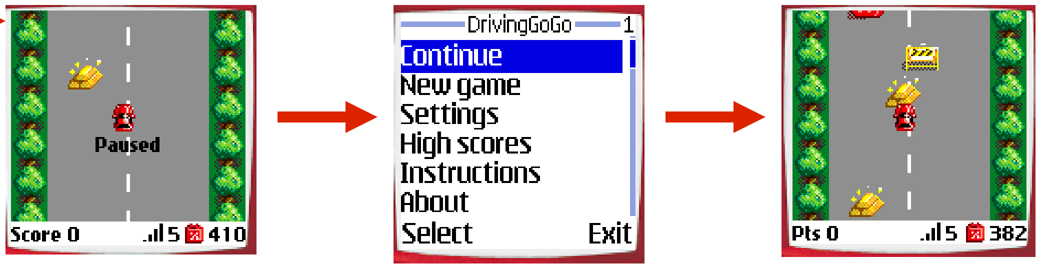
Synchronizing Game Speeds Across Different Devices

- MIDP devices exhibit different characteristics including processor speeds, available memory, and consequently different runtime performance
- Makes for running the game at the same speed across these devices a challenge
- Need to maintain a constant frame rate by:
    - Introducing a delay
    - Keeping frame rate constant by using `System.currentTimeMillis()`
      Example: `Thread.sleep(FRAME_TIME - (System.currentTimeMillis() - prevTimeMillis))`

Handling Game States

- Use `hideNotify()` in class Canvas to pause application
- `showNotify()` to “continue” application
- For Screen subclasses like `Form` use `isShown()` to test whether in foreground

Return From Phone Call



Note: From S60 second edition FP3 onwards MIDlet close upon pressing the End Call (Red) key

`END` Key Operation—Graceful Exit

- JAM calls MIDlet’s `destroyApp()` method
  - Provides a perfect place to implement auto-save
    - For accidental exit situations
    - For quickly exiting the MIDlet on purpose
  - Save any game-state-related data to RMS here
  - Save player’s nerves as, depending on the model, the `END` key can easily be pushed accidentally
- Player can later resume the game from the same state
- If the application shutdown is not completed in five seconds, JAM kills KVM immediately
  - Storing data to server or fetching something from server and storing to RMS could exceed this

END Key Operation—Auto-Save

Auto-save in MIDlet main class on exit:

```
public void destroyApp(boolean unconditional) {
       myForm.saveData();
}
```

Restore in myForm class on next start:

```
public void restoreData() {
  openRecordStore();
  try {
    if (rs.getNumRecords() > 0) {
      readData();
  }
  catch (Exception e) {
  }
  closeRecordStore();
}
```

Simultaneous Key Press Handling

Supported on Series 60 currently

```
// key1 = 1 only when KEY_NUM1 is pressed
static int key1 = 0;
// key2 = 1 only when KEY_NUM2 is pressed
static int key2  = 0;

//Key Repeated called when a key is held on for longer
protected void keyRepeated(int k){
     if (k ==KEY_NUM1){
         key1  = 1;
     }
     if (k == KEY_NUM2){
         key2  = 1;
     }
    if(key1 ==1 && key2 == 1){
   // Performs function when KEY_NUM1 and KEY_NUM2 are pressed at the
same time
      processRepeatedEvent(KEY_PRESSED_EVENT, 30); // Function written
in processRepeatedEvent
    }
}
```

Other UI/game Related Enhancements to MIDP Low Level UI APIs

- Class `Display` has a new `vibrate()` method
- Phone’s backlight can be accessed with a `flashBackLight()` method of the Display class
- Class Canvas has a new `setFullScreen()` method
- In MIDP1.0 transparent image support was optional
  - It is mandatory in MIDP2.0—Used in Sprites and TiledLayers

Optimized Graphics Programming

- Leveraging graphics primitives
- Effective use of clip regions
- Caching for performance
- Using PNG images efficiently
- Translation

Leveraging Graphics Primitives

- Graphics primitives are typically native
  - Execute very quickly compared to Java
- Use them to simplify Java code
  - Use `drawRect` instead four drawLines
  - Use `fillRect` instead of numerous drawLines

Effective Use of Clip Regions

- A clip region is always defined
  - Limits the area that can be painted
  - Indicates the area that needs to be painted
- Avoid executing complex rendering code that will be ignored
- Confine repaint requests to the area(s) that need to be updated

Effective Use of Clip Regions

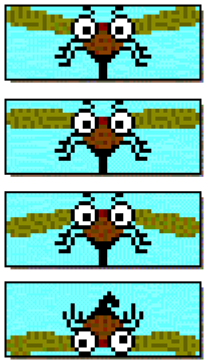
```
public void paint (Graphics g) {
   int colStart = g.getClipX() / gridSize;
   int colEnd = (g.getClipX() + g.getClipWidth()
                   + gridSize) / gridSize;
   int rowStart = g.getClipY() / gridSize;
   int rowEnd = (g.getClipY() + g.getClipHeight()
                   + gridSize) / gridSize;
   for(int row=rowStart;row<= rowEnd;row++){
   for(int col=colStart;col<=colEnd; col++){
       // Paint square for this row & col} }
}
```

Caching for Performance

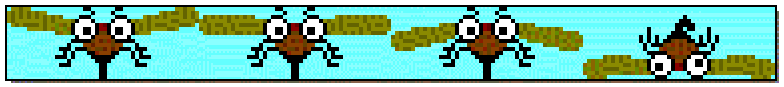
- Complex rendering operations may be cached in buffer images
- Ideal for graphical elements that change infrequently but are repainted often
- Also ideal for incremental updates of complex user interfaces

Using PNG Images Efficiently

- Combine small images into one large image
- Use clipping to render the desired portion of the larger image



Four PNG Images
1.6k



One PNG Image
0.7k

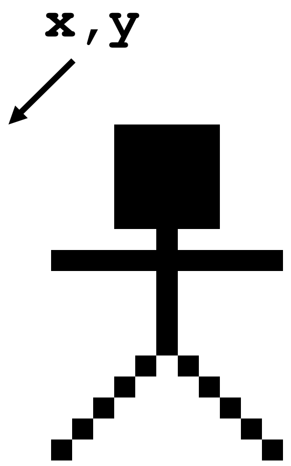
Translation

- Translation redefines the origin of the coordinate system
- Simplifies groups of rendering operations performed at an arbitrary location on the screen

Translation

- Without translation, the x and y location must be added to numerous coordinates

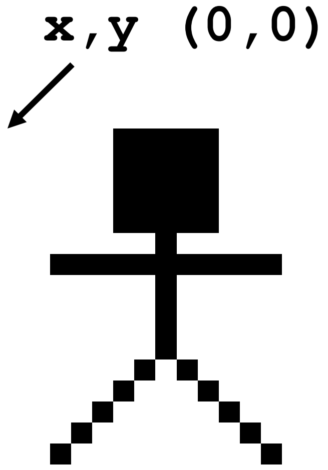
```
g.fillRectangle(5+x,0+y,5,5);
g.drawline(2+x,6+y,12+x,6+y);
g.drawline(8+x,5+y,8+x,10+y);
g.drawline(8+x,10+y,2+x,15+y);
g.drawline(8+x,10+y,12+x,15+y);
```



Translation

- Using translation, the coordinate computations are not needed

```
g.translate(x,y);
g.fillRectangle(5,0,5,5);
g.drawline(2,6,12,6);
g.drawline(8,5,8,10);
g.drawline(8,10,2,15);
g.drawline(8,10,12,15);
```



Double Buffering

- Do not double buffer if device already double buffered
  - Note: All current Nokia platform devices support double buffering
- Use `isDoubleBuffered()` to keep applications portable

```
if( isDoubleBuffered()) {
   // paint on screen
}
else {
   //paint on offscreen buffer flush buffer on the screen
}
```

UI handling and Network Connections

- Use a different thread for Connections (Ex: HTTP)
  - Keeps UI responsive
- Close connections and streams in finally block
- Do not “wait” on connection thread

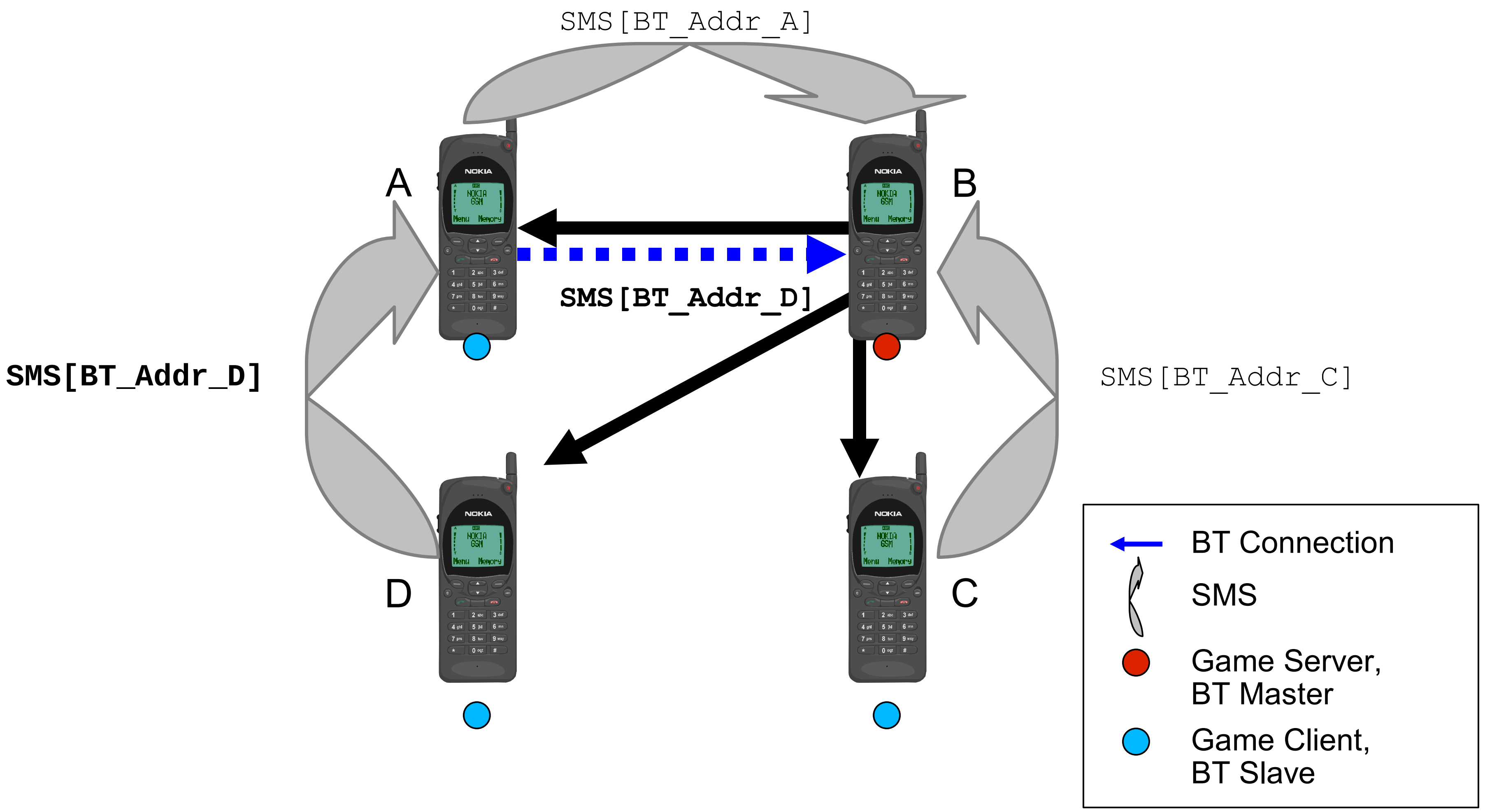
DEMO

Bluetooth Easy Connect

Java-Based Bluetooth Multiplayer Games

- Bluetooth multiplayer games are getting popular but…
  - Bluetooth multiplayer games are not end-user friendly due to the following
    - Bluetooth search
    - Bluetooth Connection strategy
  - Bluetooth multiplayer games don't use the operator network except during download or game or additional levels
- Bluetooth Easy Connect aims to target all of the issues mentioned above

Bluetooth Easy Connect @ Play



Agenda With Section Highlights

Background

Overview of UI APIs

Series 40, S60, Series 80 Platform Notes

Best Practices in MIDP Programming

Summary

Demos

MIDP Performance Encompasses…

- Application startup speed
- Programming callbacks
- Java language programming optimizations
- Graphics optimization
- Execution speed
- Use of resources
- User interface responsiveness
- Footprint reduction
- Implementation specific optimization

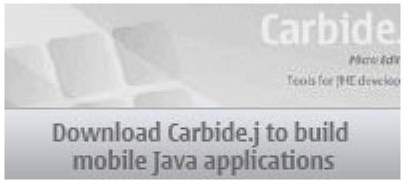
Final Words

- Code first, optimize later
- Profile, profile, profile
  - Remember the 80/20 rule
- Benchmark…
  (Irrespective of Heisenberg's uncertainty principle!)
- Yes, you can reduce footprint!

Summary

- Nokia provides comprehensive support for Java technology on the following platforms:
    - Series 40 platform
        - All Java, mass market, Nokia OS-based
        - Many latest JSRs
    - S60 platform
        - The smartphone platform, C++ and Java development platforms
        - Uptodate JSR implementation
        - Scalable UI
    - Series 80 platform
        - For enterprises; C++, Java (both CDC and CLDC)-based platforms
- Understanding Java-based implementation yields effective user experience
- Best practices help in designing applications with best UI that perform the best

For More Information



Nokia Resources

- Developer information: http://www.forum.nokia.com
- Nokia Java tools: http://www.forum.nokia.com/tools

Industry Resources

- JCP page: http://jcp.org
- Eclipse tools: http://eclipse.org
- Netbeans tools: http://netbeans.org

Q&A

Srikanth Raju, Nitin Mittal, Jarmo Lahtinen

Best Practices in UI Design and Programming on Nokia Platforms

Srikanth Raju, Jarmo Lahtinen, Nitin Mittal

Forum Nokia
Nokia
http://www.forum.nokia.com

TS-1281