Mobile Programming – Whence, Whither, How and Why?

”Mobiiliohjelmointi”-guest lecture 12.3.2009

Mikael Laine
Agenda

• Mobile software: what’s going on?
• Qt for S0
• Symbian C++ GUI programming
Mobile Platforms Now

• “Every once in a while, a revolutionary product comes along, that changes everything” - Steve Jobs on the iPhone, Macworld 2007
  • Multi-touch
  • Capacitive touchscreen
  • Web browsing
  • App Store
  • Usability!
• New emphasis on usability, looks and multimedia
• Competitors: requirement specification = iPhone
Mobile Apps: what's going on?

- iPhone SDK 2.0: an ecosystem of 3rd-party apps
  - Limited apps on a sophisticated platform
  - Easy SDK, fast development
  - App store for easy distribution
  - Many small but appealing utility apps and games
- Third-party apps a new priority and lucrative business!
  - New technologies
    - Touch
    - Accelerometer
    - Location-based services finally mainstream
    - Fast processors with 3d-acceleration
    - Cloud-based services
Application Distribution

- Apple: AppStore, July 2008
  - Huge success story
  - Demonstrated the importance of easy access
- Following:
  - Palm: App Catalog
  - Android Market
  - Nokia: OVI store, May 2009
  - Microsoft: “Bazaar” application (Windows Mobile 6.x) (not released)
The Now and the Future of Mobile Application Development

- Business apps
  - Built to a specific set of requirements
    - No "wow"-factor involved
- Consumer apps
  - Form over function: "may the sexiest app win" – looks matter!
  - Usability!
  - Distribution channels a big factor
  - Responding to culture
    - Music: iPod, "Zune"-sharing
    - Social networking (Facebook, Flickr, ...)
  - Trend-driven: today touch, inclinometer tomorrow?
  - Open source: the ability to harness and expand a platform’s capabilities
Mobile App Development Today: Technical Perspective

- Cloud-based services
  - Social networking
  - App store
  - Multimedia sharing
- Device hardware requirements
  - Games
  - Touch
  - Visual effects
- Software requirements
  - Usability
  - Touch
  - Visualization
Mobile App Development Today: Distribution

- App distribution methods
  - Free: just get the file delivered any way you can
    - Difficulties reaching the customer, advertising required
    - Easy and free of cost
  - Distribution only through specified outlet
    - Easy for customer and developer
    - May incur costs (e.g. 30% of sales)
Mobile App Development Today: Signing

- App package contents
  - Signing, and testing
    - Small cost involved
    - Puts pressure on the signing party; sometimes problems
  - Symbian
  - iPhone
  - J2ME
- No signing
  - Easy but with security drawbacks
  - Symbian
  - J2ME
Technical Solutions to New Requirements

- User experience design generates new requirements
- Whole new UI-platforms
  - iPhone
  - Android
  - S60 v. 5.0
  - ...
- 3d hardware-based applications
  - OpenGL present in many devices
- Graphics libraries etc.
  - On top of the old OS
  - Qt
  - Flash, Python
One Graphics Framework: ”Flexible UI” from Ixonos

- GUI framework built on OpenGL, but can be implemented to run on any graphical environment
- Redrawing: user-initiated, constant loop or both
- Best of Both: use the normal graphics context for simple operations, and OpenGL-drawing for fancy effects (switching is easy)
- Support for entirely customised primitives, just provide a bitmap. Also supports transparency.
- Simple event processing: no ”add to stack” needed, consume or don’t – events are available to all.
- Built-in bitmap-routines: just say where, and FUI will show what – nevermind how.
- Inclinometer and touch gestures can easily be used application-wide (centralise).
No Artificial Constraints – Freedom to Create!

- Graphical primitives may overlap
  - Powerfull visual representation possible, e.g. stacked photos, background animations and pointer trails on top of other content
  - Events (key, pointer) available to all (collaborative)
- Views may overlap
  - What is seen on the screen may consist of several “layers”, each represented by a view
  - Views may persist over applet switching and changes in other views (e.g. the navbar of Fig 2.)
- Primitives and views can easily be re-used
  - Just create the view or primitive, and add it to the list of drawable items (two lines of code, in most cases)

Fig. 2: A music store applet, consisting of several re-usable views including a navbar at the bottom. Startrail follows touch.
The Anatomy of an Applet: an Example

- **Applet “MusicStore”**
  - **View:** Background (re-used)
    - Primitives: Red orbs
  - **View:** StoreView (custom for this applet)
    - Primitives: Labels “Store”, “My Songs”
    - Primitive: List
      - a “compound” primitive, or control
      - Contains primitives for images, texts, buttons and background
  - **View:** “TrailingStars” (re-used)
    - Primitives: images of stars, trailing the pointer location (on a touchscreen).

- **Applet “ManagerApplet” (re-used)**
  - **View:** Navigation Bar (re-used)
    - Canvas background
    - Tab images
Applets, Views and Primitives

- **Applets** are small "applications" that solve a simple problem (e.g. a game, media viewers/players)
- Applets are comprised of **views**
- Views are comprised of (graphical) primitives, such as buttons, lists and bars.
- Primitives can also comprise other primitives (compound primitives)
The Anatomy of an Applet: an Example

• Applet “MusicStore”
  • View: Background
    • Primitives: Red orbs
  • View: StoreView
    • Primitives: Labels “Store”, “My Songs”
    • Primitive: List
      • a “compound” primitive, or control
      • Contains primitives for images, texts, buttons and background

• Applet “ManagerApplet”
  • View: Navigation Bar
    • Canvas background
    • Tab images
An Applet may Encompass Several Other Applets “Create Once, Use Anywhere”
Case Study: The Built-in "CLWButton" Primitive

- Specialisable: inherit for your own needs
- All elements optional: single texture or label will do
- Compound primitive
  - CLWLabel
  - CLWImage
    - Flipside images
    - Highlight image
    - Normal image

$1.99
Label
Icon mask
Icon highlight texture
Icon texture
"Flipside" icon highlight
"Flipside" icon
Application Development Platforms

- Symbian
  - Symbian C++, J2ME, Flash, Python, Qt,…
- Linux
  - GTK, …
- Android
  - Java, API not JDK
- iPhone
  - Objective C
- RIM
  - Java, with proprietary APIs
- Windows
  - .NET, …
Many Aspiring Mobile Platforms (Smartphone market shares)

- Symbian OS
  - S60
- iPhone OS X
- RIM Blackberry
- Windows Mobile
- Mobile Linux
  - Android
  - OpenMoko
  - ...
- Palm
- BREW
Symbian OS? Still Relevant?

• S60 is the biggest platform on Symbian OS
  • Well-designed internals
  • Many shortcomings in the bling-department
    • Limited UI components, lack of flexibility
  • Difficult and isolated platform for 3rd-party app development

• Big shifts going on at Nokia, the main endorser of Symbian OS
  • Answers sought to new “post-iPhone” requirements
  • Nokia owns Symbian since Dec. 2nd, 2008 → “Symbian Foundation”
    • Royalty-free Symbian platform available to members
    • Membership open to all organizations (annual fee US$1,500)
  • Nokia owns Trolltech since June 2008 (new name “Qt Software”)
    • Powerful and widely used application platform
Symbian - History

Psion founded

Psion Organizer

Psion Organizer

Psion Organizer

SIBO

EPOC

1980

1984

1990

1994

1998
Symbian Foundation

- Reduce Fragmentation (common platform)
- Expected to fuel the growth of multimedia-capable devices
- Builds on a large consumer base (over 200 million devices in use)
- Makes available the Symbian Foundation platform
  - Royalty-free licence
  - For members
- Application development free and open
  - Membership not required
  - Source code available
- Graphics and Apps
  - S60, UIQ and MOAP unified to create an unparalleled open software platform for the "mobile ecosystem"
The Future of Symbian OS: Speculation

Symbian C++ Applications
Qt Applications
Java, Python, etc. Applications

Symbian Foundation platform (Open Source)
Symbian OS (Open Source)
Qt [cute]

- Cross-platform application framework
- In wide use, with large developer base
- C++ programming language
  - Libraries
  - Preprocessor
    - Meta-object system (signals, introspection, internationalization)
    - New keywords, such as emit, signals, and macros like Q_OBJECT
    - Generates standard C++ code
- Webkit integration (S60 browser based on Webkit as well)
- Just what S60 needs
  - Looks nice, as the name implies
  - Will attract a plethora of developers
  - A new era of 3rd-party apps on S60 platforms!
Qt [cute] cont.

- Write once, compile anywhere – portability
  - Mac OS X
  - Windows (desktop, CE)
  - Linux/X11 (desktop, embedded)
  - S60 (not released yet)
- Offers a set of basic functionalities
  - OpenGL Graphics available through `QGLWidget`
  - I/O (file system, sockets, HTTP, XML,...)
  - Database interface
  - Container classes (vectors, lists, ...)
- Symbian C++ APIs can be used
Qt [cute] – A Little History

• 1991 Work started by Haavard Nord and Eirik Chambe-Eng
• 1994 The company TrollTech incorporated
• 1996 The release of Qt version 1.0
• 1996 The founding of the KDE project
  • Qt chosen ➔ important platform for Linux
• 1999 Qt 2.0
• 2001 Qt 3.0 Improved locale and unicode support
• 2005 Qt 4.0
• 2008 Trolltech bought by Nokia, new name ”Qt Software”
• 2008 S60 version announced
• 2009 S60 technology preview released
Qt: Large Userbase

- Adobe
- Wolfram Research (Mathematica)
- LucasFilm Ltd. (internal applications)
- Google earth
- Skype
- Walt Disney
- KDE
- Sony
Qt Designer

• Design your GUI easily
  • UI designer
Hello Qt – Example Qt Application

```cpp
#include <QApplication>
#include <QLabel>
int main(int argc, char *argv[]) {
    QApplication app(argc, argv);
    QLabel *label = new QLabel("Hello Qt!");
    label->show();
    return app.exec();
}
```

- Includes definition of QApplication
- Includes definition of QLabel
- Main function (starting point)
- QApplication object
- A Label `widget` to display the text
- Hidden by default, make visible here
- Control passed to the Qt framework; event loop started.
Qt – Events and Signals

• Qt uses a special concept called ”signals” to bind objects together
  • Unrelated to Unix signals
  • Signals: objects may ”emit” signals
  • Slots: objects may handle signals
    • A normal C++ function declared under the ”slots:”-keyword
  • Connected objects need no knowledge of each other, signals routed by the framework
• Signals are not events
  • Signals are for inter-object communications (e.g. using widgets)
  • Events are for handling lower-level events (e.g. creating widgets)
Qt: Signals

- Declare a signal function in the "signals:"-portion of your code
- Declare a slot function in the "slots:"-portion of your code
- connect()-function used to connect signals to slots
- Based on the *meta-object system* (moc) which provides two services
  - Signals-slots
  - Introspection
    - Functions declared by the Q_OBJECT macro allow for inspection of each object’s metadata
- Beautiful way of connecting objects
Qt: Signals example

class Employee : public QObject
{
    Q_OBJECT
public:
    Employee() { mySalary = 0; }  
    int salary() const { return mySalary; }  
public slots:
    void setSalary(int newSalary);
signals:
    void salaryChanged(int newSalary);
private:
    int mySalary;
};
Qt: Signals example (slot impl. & emit)

```cpp
void Employee::setSalary(int newSalary)
{
    if (newSalary != mySalary) {
        mySalary = newSalary;
        emit salaryChanged(mySalary);
    }
}
```

Implementation of the "slot" function: just a normal function.

Here we emit another signal. We can emit signals where ever we like.
Qt: Signals example (connecting)

- Slider value connected to an employee object’s `setValue`

```cpp
Employee *employee;
...
connect(slider, SIGNAL(valueChanged(int)),
       employee, SLOT(setValue(int)));
```

- Connection removed

```cpp
disablect(slider, SIGNAL(valueChanged(int)),
         employee, SLOT(setValue(int)));
```
Qt: Signals example (connecting, cont.)

- Signal has more arguments than the slot
  - Normally connected signals and slots have exactly the same argument types in the same order
  - Additional arguments ignored

```cpp
code {color: gray}connect(corporation, SIGNAL(salariesChanged(int, const QString& explanation)), employee, SLOT(setValue(int)));
```

- Many signals can be connected to the same slot

```cpp
code {color: gray}connect(lcd, SIGNAL(overflow()), this, SLOT(handleMathError()));
connect(calculator, SIGNAL(divisionByZero()), this, SLOT(handleMathError()));
```
Qt: localization

• Very simple from a developer’s point of view
  • Declare all literals within the \textit{tr()} function call
    • Only literals can be localized

\texttt{QString str1 = tr("Hello");}
\texttt{char *str = "hello";(QString translated = tr(str));}

• When localization required, call the \textit{lupdate} tool
  • Produces translation files for translation
Qt: Widgets

- The name for visual components in Qt
- Inherit from QWidget
- Drawing in paintEvent() event handler function
- May request a draw with repaint (immediate), or update (next event loop)
- May handle events in the event() function or some of the specialized functions
  - paintEvent()
  - resizeEvent()
  - mousePressEvent()
  - Drag’n drop functions
  - Clipboard functions
  - ...
  - keyPressEvent()
Qt: Using Widgets

- Top level (no parent) widget is treated as a window
- Several readymade widgets
  - Buttons (e.g. QPushButton)
  - Editors (e.g. QLineEdit)
  - Sliders (e.g. QSlider)
  - Tables (QTableView)
- Layout managers used
  - QLayout
    - QBoxLayout, QFormLayout, QGridLayout, QStackedLayout
Qt: Additional Information

- Qt homepage (www.qtsoftware.com)
  - Book as pdf (doc.trolltech.com/4.4/how-to-learn-qt.html)
  - Examples
  - Tutorials
- Qt for S60 Developer’s Library (forum.nokia.com)
- Qt for S60
  - Available from the Qt homepage
Qt Example: QtTouchy

- A simple touch-based game
Linux: Maemo

- Linux platform from Nokia
- Based on Linux and GTK
  - Hildon UI built on top of GNOME
- SDK available from forum.nokia
- Currently targets “Internet Tables” devices
  - (N800, N810)

- Short demo of the MAEMO SDK environment
Symbian - overview

• Symbian OS (The underlying operating system)
  • Microkernel architecture
  • Strongly object-oriented (almost all code in C++)
  • Multi-tasking, Memory optimization/protection
  • Modularity
  • Developed by Symbian Ltd. in London

• Add-on (User Interface) Layers
  • S60 (Nokia)
  • UIQ (Ericsson)
  • MOAP (DoCoMo)
Symbian - History

1980 Psion founded
1984
1990 SIBO
1994 EPOC
1998

© Ixonos Oyj 29.3.2007
Symbian UI Application Structure

**App Arc**
- CApaApplication
- CApaDocument

**Uikon**
- CEikApplication
- CEikDocument
- CEikAppUi

**Avkon**
- CAknApplication
- CAknDocument
- CAknAppUi

**Cone**
- CCoeAppUi
- CCoeControl
- CCoeEnv

**UI Variant layer (Nokia)**

**Application**
- CSomeApplication
- CSomeDocument
- CSomeAppUi
- CSomeView
Runtime Environment

- CONE
  - UI-framework
  - All UI-applications have an instance of this
  - Implements the client interface towards the window server
  - A handle to an instance of CCoeEnv is always available through a static function CCoeEnv::Static()
    - AppUI()
    - FsSession()
    - CreateGcL()
    - Window Server front-end
      - RunL (user events)

- UIKON
  - Implementations of UI-controls
  - CEikonEnv inherits from CCoeEnv
  - Replacement of EIKON (greek for “image”) starting with Symbian 5.1 (Unicode support)

- AVKON
  - A layer of S60-specific classes on top of UIKON
  - Similar layers: UIQ, CKON
Application Architecture Classes in Short

- Application, inherits from CApaApplication
  - Usually a rather dumb class
  - Creates an instance of the Document class
- Document, inherits from CApaDocument
  - Meant to store the internal state and data of the application
  - Creates an instance of the AppUI-class, around which the user interface is built.
Application Architecture Classes in Short

- Application UI (AppUI)
  - Inherits from CEikAppUi
  - Owns the UI-controls ("drawing surfaces") one way or another
  - Handles commands
  - Does not draw anything, but creates a view or several views
    - A single view of type CCoeControl
    - Several views of type CAknView, containing CCoeControls
  - AVKON adds another level to the inheritance, see for example CAknViewAppUi
Model-View-Controller (Simple)
Model-View-Controller (Multi-View)
Model-View-Controller (S60 View-Switching)
S60 View-Switching Architecture

- Sometimes more than one view is needed
- AppUi central: manages views and switching between them
- Adds one more layer: the view-layer
  - CAknViewAppUi
  - CAknViewApp
  - CAknView
Structure of Applications

• CONE/EIKON/UIKON Does not impose a “correct” place for application logic
• Drawing and graphics must be implemented in view- and/or container -classes due to inheritance from CCoeControl
• Usually it is a good idea to have the application logic in a separate “engine” component. Testing can be performed separately, and re-use is made easier. The engine component contains data structures and routines to manipulate the “model”
Structure of Applications, cont.

• The application controller is often implemented in AppUi, but for more complicated applications a separate “engine” is recommended.

• Most conventions of Object Oriented Programming are applicable to Symbian

• The structure of Symbian applications is rather free, but the general model steers development toward a certain model
S60 View-Switching Architecture

• Views accessed through CAknViewAppUi’s ActivateLocalViewL (TUid aViewId)

• Every view needs a window
  • Top-level control needs to be window owning
  • Subcontrols usually lodgers (not window owning)
  • Unnecessary window-owning controls should be avoided
    • infrastructure
    • demand on the Window Server
    • performance reduction

• Every view initializes its control(s)
  • At least one simple control is needed, otherwise nothing will be drawn
Event Handling: Overview

keyboard

Interrupt

Kernel

ISR

Window Server

Handle Key Event

Update Window

Application

Handle Key Event
Command Handling

- S60-applications have a menu, and “soft keys” (CBA: Common Button Array). Their appearance is defined in resource files
  - Every menu command has a numeric id
  - Selecting a command causes a call to AppUi::HandleCommandL
  - In a multi-view case, CAknViewAppUi can also implement HandleCommandL
  - View-specific commands (e.g. menu commands) can also be processed in CAknView::HandleCommandL()
Command Handling: Keyboard Events

- OfferKeyEventL()
- The Window Server routes key events to the foreground application
  - CCoeEnv::RunL() → CCoeAppUi::HandleWsEvent() → CCoeControlStack::OfferKeyL()
- Process key events in AppUI:
  - HandleKeyEventL()
    - Called if none of the controls consumes the event
- Process key events in a view:
  - OfferKeyEventL()
  - CCoeAppUi::AddToStackL() called first to add the control to the AppUi’s control stack
  - Compound controls should pass events to their child controls
Command Handling: Pointer events

- Events from a stylus (pencil), finger, mouse...
- The **Window Server** ensures events are sent to the right **window**
- The **Control Environment (CONE)** ensures events are sent to the right **Control**
  - `CCoeEnv::RunL \rightarrow CCoeAppUi::HandleWsEventL() \rightarrow CCoeControl::ProcessPointerEventL() \rightarrow HandlePointerEventL()`
- Implement `HandlePointerEventL` to receive pointer events
Resource Files

• Provide a separation between code and data
  • Textual descriptions translated into binary files that are used at run-time
• The most important statement within resource files is
  • RESOURCE struct_name [id] {member_list}
• Allowable statements in resource files are
  • NAME
  • STRUCT
  • RESOURCE
  • ENUM
  • CHARACTER_SET
Resource Files: Punctuation

• Member definitions with braces, beginning with "[name] =" must end in a semicolon
  titles =
  
  {
    MENU_TITLE
    
    {
      menu_pane = r_learningui_container_menu_pane;
    }
  }

• If a closing brace separates items in a list, put a comma between them
  MENU_ITEM {command = EsimpleappCommand1; txt = "Say Hey!";},
  MENU_ITEM {command = EAknSoftkeyExit; txt = "Exit";}

• Normally no commas or semicolons are needed
Example of a Menu Resource

RESOURCE MENU_BAR r_simpleapp_menubar
{
  titles =
  {
    MENU_TITLE {menu-pane = r_simpleapp_menu;}
  }
}

RESOURCE MENU_PANE r_simpleapp_menu
{
  items =
  {
    MENU_ITEM {command = EsimpleappCommand1; txt = "Say Hey!";},
    MENU_ITEM {command = EAknSoftkeyExit; txt = "Exit";}
  }
}
Example of a Dialog Resource

RESOURCE DIALOG r_aknexquery_confirmation_query
{
    flags = EGeneralQueryFlags;
    buttons = R_AVKON_SOFTKEYS_YES_NO;
    items =
    {
        DLG_LINE
        {
            type = EAknCtQuery;
            id = EGeneralQuery;
            control = AVKON_CONFIRMATION_QUERY
            {
                layout = EConfirmationQueryLayout;
                label = qtn_aknexquery_con_label_text;
            }
        }
    }
}
Resource files

- Why:
  - Data separation
  - Changes to the outlook and data can be made without touching the executables
  - Localization made easy

- What’s in a Resource File:
  - Character Strings for the UI
  - Menu definitions
  - Dialog definitions
  - Layout, geometry
  - Data structures
CBA Resource Example

RESOURCE CBA r_cba_turku_tampere
{
    buttons =
    {
        CBA_BUTTON
        {
            id = ESimpleApplicationCbaCommandTurku;
            txt = "Turku";
        },
        CBA_BUTTON
        {
            id = ESimpleApplicationCbaCommandTampere;
            txt = "Tampere";
        }
    }
}
Using CBA-resources

If ( aOutoTilanne )
{
   CEikButtonGroupContainer *cba = Cba();
   Cba->SetCommandSetL ( R_CBA_TURKU_TAMPERE );
   Cba->DrawDeferred();
}

-> HandleCommandL contains the following:
void CsimpleappAppUi::HandleCommandL(TInt aCommand)
{
   switch(aCommand)
   {
      case ESimpleApplicationCbaCommandTurku :
         ...
         break;
      case ESimpleApplicationCbaCommandTampere:
Resource Files, cont.

- Source code in:
  - Resource files *.rss
  - Resource headers *.rh
  - Resource constant headers *.hrh
- After compilation:
  - Binary resource file *.rsc
  - Compile-time files *.rsg and *.mbg (headers)

- The resource file’s name indicates the language code: sc=english, *.r01, *.r02
- BaflUtils::NearestLanguageFile()
  - Retrieves the resource file with the closest matching language id
- The framework automatically loads all the resource files mentioned in the mmp-file
- CCoeEnv::AddResourceFileL()
  - Loads more resource files to the list maintained by CCoeEnv if necessary
  - Also call DeleteResourceFile() afterwards
Resource Files, cont.

- Once the resource file is loaded, it can be read using CCoEEnv::ReadResource() or by instantiating TResourceReader
- For loading character strings, S60 has StringLoader, which has convenient static functions
- A Good practice is to store strings into a separate *.loc file, which is #included into the resource file

```
simpleapp.loc:
    #define SIMPLEAPP_CBA_TURKU “Turku”
    #define SIMPLEAPP_CBA_TRE “Tanperesteri”
```

```
simpleapp.rss:
RESOURCE CBA r_cba_turku_tampere
{
    buttons =
    {
        CBA_BUTTON
        {
            id = ESimpleApplicationCbaCommandTurku;
            txt = SIMPLEAPP_CBA_TURKU ;
        },
        CBA_BUTTON
        {
            id = ESimpleApplicationCbaCommandTampere;
            txt = SIMPLEAPP_CBA_TRE;
        };
    }
}
```
Character Strings in Source Code

- Storing strings in the executable binaries
  - In source code:
    - _LIT(KMyLine,"Good morning, Symbian!");
    - iLabel->SetTextL(KMyLine);
- Storing strings in resource files
  - In a resource file:
    - RESOURCE TBUF r_string { buf = "Good morning, Symbian!";}
  - In source code:
    - TBuf<22> infostring;
      iEikonEnv->ReadResource(infostring, R_STRING);
Localization in Practice

• All strings should be stored in localization files
  • Strings in resource files changed to references to strings in localization files
    • buf = “Hello” → buf = string_my_line
  • Named .lXX, where XX is the language code
    • Language codes defined in the enum TLanguage in “e32const.h”
• Change #include "LearningUI.loc" in your .rss file to
  • #if defined LANGUAGE_SC // default (U.K. English)
    #include "MyApp.LOC"
    #elif defined LANGUAGE_06 // Swedish
    #include "MyApp.l06"
    #endif
Localization in Practice (Installation file)

- Change the .pkg file:
  - "$(EPOCROOT)Epoc32\data\z\resource\apps\LearningUI.r01" - "!:\resource\apps\LearningUI.rsc"
  - -> {"$(EPOCROOT)Epoc32\data\z\resource\apps\LearningUI.r01", "$(EPOCROOT)Epoc32\data\z\resource\apps\LearningUI.r06"}
    - "!:\system\apps\LearningUI.rsc"
- Change all strings within .pkg to a list of strings
  - #{"LearningUI"},(0xE4886003),1,0,0
  - -> #{"LearningUI", "LearningUI"},(0xE4886003),1,0,0
- At install-time, the installation system will prompt for the language
  - The compiler does not know which language will be used
  - The language is independent of the executables
UI-Controls

- All drawing done in CCoeControl-derived classes
  - Dialogs
  - Buttons
  - Scrollbars
  - Editors
- Eikon, Uikon, Avkon, Qkon etc. all provide a set of UI-controls with a different look&feel
- CCoeControl-derived features include handling keyboard events, pointer events, and drawing on the screen
- CCoeControl may be composed of several controls (nesting)
- Often report their state through an observer interface
Compound Controls

- One control may "own" other controls (composition)
- Window ownership
  - Usually subcontrols use the owner control’s window
    - SetContainerWindow(CCoeControl*) must be called when instantiated
    - ConstructL(const TRect&, const CCoeControl*)
  - Every view must have a direct child control which is window owning
- Subcontrols may not overlap
  - Boundaries set using SetRect()
Managing Subcontrols

- Two methods
  - CCoeControlArray
    - New method since Symbian 9.1 (should be used)
  - Implementing the following functions:
    - CountComponentControls()  
    - ComponentControl() 
    - Older method, still works but is discouraged
Using CCocontrolArray

InitComponentArrayL();
CCoeControl* myComponent = new (ELeave) CComponent;
Components().AppendL(myComponent);
myComponent->ConstructL();
myComponent->SetThisAndThatL();
CleanupStack::Pop(myComponent)
CountComponentControls() and ComponentControl()

- CountComponentControls()
  - Returns the number of subcontrols contained in a compound control
- ComponentControl()
  - CCoeControl *ComponentControl(TInt aIndex)
  - Returns components of a compound control
UI-Controls, cont.

- CCoeControl-derived classes
  - Own a rectangular area, where they may draw themselves
  - Have a window on which to draw
    - Controls that are “lodgers” within the area of another control typically use the same window as their owner
    - Compound controls are “window owning”
  - Only one CCoeControl has focus at a time
    - Keyboard events are sent to this control
  - In case of compound controls, an event stack is used to offer key events to several controls in order
    - CCoeAppUi::AddToStackL()
    - RemoveFromStack
  - Key events handled in CCoeControl::OfferKeyEventL()
    - Has to be implemented in order to receive key events
CCoeControl

CCoeControl
class CCoeControl : public CBase
{
IMPORT_C virtual TKeyResponse OfferKeyEventL(const TKeyEvent& KeyEvent, TEventCode aType);
IMPORT_C virtual void HandlePointerEventL(const TPointerEvent& PointerEvent);
IMPORT_C virtual void SetContainerWindowL(const CCoeControl& Container);
IMPORT_C virtual void ConstructFromResourceL(TResourceReader& aReader);
IMPORT_C virtual void ActivateL(); IMPORT_C void DrawNow() const;
IMPORT_C virtual TInt CountComponentControls() const;
IMPORT_C virtual CCoeControl* ComponentControl(TInt aIndex) const;
IMPORT_C virtual void Draw(const TRect& aRect) const;
IMPORT_C void SetObserver(MCCoeControlObserver* aObserver);
IMPORT_C virtual TInt CountComponentControls() const;
IMPORT_C virtual CCoeControl* ComponentControl(TInt aIndex) const;
IMPORT_C virtual void Draw(const TRect& aRect) const;
};
+ many other functions
Drawing UI Controls

- CCoeControl draws itself when
  - The window server causes a call to Draw
  - Drawing is initiated from the code
- In case of compound controls, Draw is also called on the subcontrols
- CCoeControl::DrawNow()
  - Draw everything now
- CCoeControl::DrawDeferred()
  - Draw everything pretty soon
- RWindow::Invalidate(const TRect &aRect)
  - Window-server initiated redraw of a specific area

- If possible, use the rectangle parameter given in Draw(); only draw what is needed
- Notice the similarity to X11’s XExpose-event. The entire window-server architecture is somewhat similar
CCoeControl::Draw

- Draw() is a virtual, “const” function. This needs to be implemented in the case of your own controls
- Use The CWindowGc class for drawing
  - Much like Java’s Graphics-class
- Draw() may get called several times during execution. Should be snappy!
  - Bitmaps ready
  - Strings ready
  - Only drawing, not logic within Draw()
  - May not (contain code that causes a) leave
Readymade Controls

- CEikEdwin
- CEikTextListBox
- CEikCommandButton
- CEikDialog
- CEikScrollBar
- CEikLabel
- CEikBorderedControl
- CEikToolBar

- Documentation available at forum.nokia.com
  - resource definitions
  - code examples
Kiitos – Thank you

www.ixonos.com