Designing Interactive Systems II

Computer Science Graduate Programme SS 2010

Prof. Dr. Jan Borchers
RWTH Aachen University

http://hci.rwth-aachen.de

Review

• Web 2.0 in keywords
• GWT
• Cappuccino
• HTML5

Introduction

• Cross platform GUI Toolkit
  • Available for X11, Windows, Mac
  • Toolkit used by the KDE project
  • Managed by a company that provides official support
• Dual license
  • after pressure from open source community

http://qt.nokia.com/
**History**

- Started out in 1994 by Trolltech (Norwegian)
- Adopted by Matthias Ettrich for KDE (1996)
- Trolltech introduced Qtopia (2001)
  - Application platform for Linux based mobile devices
- Nokia bought Trolltech (2008)
  - Pushed Qtopia to be a new platform for Symbian, Windows CE / Mobile and Maemo

**Features**

- Extended C++
  - MOC files are meta-compiled into C++
- Custom widget behavior accomplished through signals and slots
- Plug-ins for mimicking look of other toolkits (Windows, Mac, Motif, etc...)
- UIDS creates XML files, which are meta-compiled into C++

**Signals & Slots Motivation**

- Disadvantages of Callbacks
  - Callbacks are strongly coupled to processing function
  - Callbacks are not type safe when using (void *)
    - Example: Button_CB(Fl_Widget *, void *)
• **Signals** are emitted by objects when they change their state in a way that may be interesting to the outside world.
• **Slots** can be used for receiving signals, but they are also normal member functions.
• **Advantages**
  • loosely coupled, anonymous communication
  • type safe
• **Similarities to bindings** in Cocoa

### Signals & Slots Example

```cpp
class Hello : public QWidget
{
    Q_OBJECT
public:
    Hello( const char *text, QWidget );
signals:
    void clicked();
};

int main( int argc, char **argv )
{
    QApplication a(argc, argv);
    Hello h("hello world");
    QObject::connect( &h, SIGNAL(clicked()), &a, SLOT(quit()) );
}
```

```cpp
class Q_EXPORT QApplication : public QObject
{
    Q_OBJECT
public:
    QApplication( int &argc, char **argv );
public slots:
    void quit();
};
```

### Demo

```cpp
class Q_EXPORT QApplication : public QObject
{
    Q_OBJECT
public:
    QApplication( int &argc, char **argv );
public slots:
    void quit();
};
```

```cpp
int main( int argc, char **argv )
{
    QApplication a(argc, argv);
    Hello h("hello world");
    QObject::connect( &h, SIGNAL(clicked()), &a, SLOT(quit()) );
}
```
Advanced Features

- Supports Phonon multimedia framework
- Adheres to MVC paradigm since v4.0 (InterView)
- OpenGL accelerated 2D rendering and transformations (even on active widgets)
- Extremely sophisticated parallel processing (multi-threading and IPC) capabilities (e.g., QFuture)
- Qt is one of the most well-documented UITKs (check out http://doc.trolltech.com)

Qt Embedded / Qtopia Core

- Qt for Linux based mobile devices
  - Replaced X by Linux framebuffer
- Has the same API as Qt Desktop
  - Learn one API, target multiple platforms (Windows, X11, Mac OS X, embedded Linux)

Evaluation

- Availability: high
  - free for GPL use on X11, Mac, and Windows
  - $3000/license for commercial use
- Productivity: high with Qt Creator
- Performance: signals & slots mechanism adds some extra overhead, but not a lot
- Graphics Model: rasterop and vector (since v4.0)

Evaluation

- Adaptability: mimic various other toolkit, define your own ‘stylesheet’
- Extensibility: pretty high - free to modify source code
- Resource Sharing: yes
Java History

- Java 1.0 (1995): 6-week version of AWT
- Java 1.1: Listeners event model, localization
- Java 2, v.1.2: JFC (Swing, Java2D, Accessibility, Drag&Drop), audio playback
- Java 2, v.1.3: audio in, MIDI, Timer (for UI, animations, etc.)
- Java 2, v.1.4 (2002): full-screen mode, scrollwheels, Preferences API
- Java 2, v. 5.0 (a.k.a. J2SE 1.5) (2005): Java 2D, improved internationalization, Java Sound
- Java SE 6 (2006): Scripting host, dynamic compilation, JDB4

What is AWT?

- Abstract Window Toolkit
- OO UI toolkit for the Java platform
- Maps to native widgets of the host platform
- First version of AWT was developed in only 6 weeks!
## AWT Architecture

- **Apps**
- **UITK**
- **WM**
- **BWS**
- **GEL**
- **HW**

### AWT
- Java is not a complete OS
- No own window manager
- Applications use AWT for graphics
- AWT works on top of the Java Virtual Machine (JVM)

### JVM

## AWT overview

- Component as top level object
- Containers can contain multiple widgets
- Layout Managers handle the positioning
- Events are being handled with Listeners
- One window per widget (heavyweight)

## Applets vs Applications

- Java offers two kinds of UI programs:
  - **Applets**
    - run inside a web browser (or Applet Viewer)
    - embedded in HTML source
    - restricted access to underlying OS
  - **Applications**
    - run as standalone, (almost) full OS access
    - subclasses of Frame

## Hello AWT

```java
import java.awt.*;

public class Hello extends Frame {
    public static void main(String argv[]) {
        new Hello();
    }

    Hello() {
        Label hello = new Label("Hello World");
        add(hello, "Center");
        setSize(200, 200);
        setVisible(true);
    }
}
```
The Component Class

- Parent class for all things to see and interact with onscreen (except for menus: MenuComponent)
- Over 150 methods
  - from getWidth() to addMouseMotionListener()

Events in Java 1.0

- Component class has an action() method
- Public boolean action (Event E, Object o);
- All events belonging to that Component go to action()
- Problem: huge action() methods with lots of if statements

import java.awt.*;

public class OldEvents extends Frame {
    public static void main(String argv[]) {
        new OldEvents();
    }
    OldEvents() {
        Button button = new Button("Click me");
        add(button, "Center");
        setSize(200, 200);
        setVisible(true);
    }
    public boolean action (Event e, Object o) {
        String caption = (String)o;
        if (e.target instanceof Button)
            if (caption == "Click me")
                System.out.println("Button clicked");
        return true;
    }
}
import java.awt.*;
import java.awt.event.*;

public class NewEvents extends Frame implements ActionListener {
    public static void main(String argv[]) {
        new NewEvents();
    }
    NewEvents() {
        Button button = new Button("Click me");
        add(button, "Center");
        button.addActionListener(this);
        setSize(200, 200);
        setVisible(true);
    }
    public void actionPerformed(ActionEvent event) {
        System.out.println("Button pressed");
    }
}

Layout managers

- Widgets are dynamically positioned
- Container widgets have child widgets
- Layout managers are attached to containers
- Various types: GridBagLayout, BorderLayout, FlowLayout, ...
- No (pixel-) absolute positioning

Pros

- Advantages of AWT
  - Speed: use of native peers can speed up component performance
  - Applet Portability: most web browsers support AWT classes by default
  - Look and Feel: AWT components more closely reflect the look and feel of the OS they run on

Cons

- Disadvantages of AWT:
  - high overhead (one window per widget)
  - only few widgets (common denominator)
  - hard to port (platform specific limitations)
  - not very extensible
Java Swing
it's spelled JFC

• Derived from Netscape’s IFC
• Swing is a “lightweight” UI toolkit for Java
• Four times as many widgets as AWT (trees, ...)
• Pluggable look and feel
• Runs on Java 1.1.5+, included with Java 1.2+
• JFC (Java Foundation Classes) include Swing, drag and drop, clipboard support, etc

Java
pluggable look-and-feel
DEMO
The Swing solution

- Swing is implemented in "100% pure" Java
- Using AWT only for root-level widgets
- Providing AWT-like API
- Offers advanced widgets on all platforms
- Pluggable look and feel - can mimic host platform or be a custom theme
- Supports MVC

MVC in Swing

- View and controller combined into delegate
- Interfaces for Model and View (e.g. ButtonModel, ButtonUI)
- Delegates implement ComponentUI
- Allows customization of UIs

Hello, Swing

import javax.swing.*;

public class Hello extends JFrame {
    public static void main(String argv[]) {
        new Hello();
    }

    Hello() {
        JLabel hello = new JLabel("Hello World");
        getContentPane().add(hello, "Center");
        setSize(200, 200);
        setVisible(true);
    }
}

Other toolkits for Java

- SWT (http://www.eclipse.org/)
  - Written in Java, but using native widgets through JNI
- subArctic (http://www.cc.gatech.edu/gvu/ui/sub_arctic/)
  - animation, snapping, dragging, etc
- Piccolo (http://www.cs.umd.edu/hcil/piccolo/)
  - Toolkit for zoomable UIs
  - bindings for Cocoa (discontinued), WinForms, wxWidgets, gtk, etc
Java: Evaluation

- **Availability**: high (binary portability)
- **Productivity**: medium with AWT, high with Swing
- **Parallelism**: external yes, internal depends on OS
- **Performance**: medium (bytecode interpretation), memory and performance tradeoffs between AWT and Swing

Java: Evaluation

- **Graphics model**: RasterOp, Vector based
  - Java2D offers vectors, uses GPU for acceleration
- **Style**: native with AWT, pluggable-simulated with Swing
- **Extensibility**: high
  - It's open source...

Java: Evaluation

- **Adaptability**: fairly high (Swing)
  - custom look and feels, can be switched at runtime
  - ResourceBundles can store resources (like text and icons for different languages)
    - but no human-readable format for all languages (properties files limited to ISO-8859-1)
  - Resource sharing: depends on core OS
  - Distribution: depends on core OS

Java: Evaluation

- **API structure**: OO
- **API comfort**: high with Swing
- **Independence**: high, Swing has support for MVC
- **Communication**: Clipboard and drag and drop with Swing (improved with J2SE6)