Model View Controller and Event-Driven UI in Flash/Flex

CS160: User Interfaces
Maneesh Agrawala and Nicholas Kong

http://www.youtube.com/watch?v=WHxQU4RhyLk
- Most heavily used features directly mapped (volume, play/pause)
- Circular movements mapped to linear operations

Review: Metaphor
Review: Metaphor

Cognetics
- Ergonomics of the mind
- Study of “engineering scope of our mental abilities”

Cognitive Conscious/Unconscious
- What is the last letter in your first name?

Locus of Attention
- Idea/object/event which you are intently thinking about
- Focus implies volition; locus not always consciously controlled

Review: Cognition

Jef Raskin
Review: Modes

Noun-Verb VS Verb-Noun

Noun-Verb: Select object, then do action
- Emphasizes ‘nouns’ (visible objects) rather than ‘verbs’ (actions)

Advantages
- Closer to real world
- Modeless interaction
- Actions always within context of object
  - inappropriate ones can be hidden
- Generic commands
  - the same type of action can be performed on the object
  - e.g. drag ‘n drop:
Individual Programming Assignment  
(due Mar 2)

Design and Implementation Components
- Sketches of 3 alternatives, pick a favorite
- “Discount” user studies in section (Feb 25-26)
- Write up what you learned from the study
- Note how you changed your interface as a result
- Implement user interface

Application area: Project Management/To-Do List
- Items should have start and end date
- Traditional to-do list checklist view
- Timeline view

Individual Programming Assignment  
(due Mar 2)

Project Management/To-Do List
Tasks have the following properties:
  • Task Name
  • Percentage Completed (0-100%)
  • Start and End date
  • Priority
  • List of people assigned to the task
  • URL related to the task

Checklist view
• Include checkbox to automatically set completion percentage to 100%
• You should be able to see the completion percentage

Timeline view
What is a magic lens?

Timeline view

January February March

Magic Lens

January February March

2/10 - 3/5
Topics

Interactive application programming
  – Component Model
  – Event-Driven User Interfaces

Model-View-Controller
  – Architecture for interactive components
  – Why do we need it?
  – Changing the display

Interactive Application Programming
In the beginning…

http://www.cryptonomicon.com/beginning.html

The Xerox Alto (1973)
Event-Driven UIs

Old model (e.g., UNIX shell, DOS)
- Interaction controlled by system, user queried for input when needed by system

Event-Driven Interfaces (e.g., GUIs)
- Interaction controlled by user
- System waits for user actions and then reacts
- More complicated programming and architecture

Widgets
Widgets

Encapsulation and organization of interactive controls
  - Class hierarchy encapsulating widgets
  - Top-level “Component” class
    • Implements basic bounds management, and event processing

Drawn using underlying 2D graphics library

Input event processing and handling
  - Typically mouse and keyboard events

Bounds management (damage/redraw)
  - Only redraw areas in need of updating

Java Swing Widgets

![Java Swing Widgets](image)
Windows Vista Widgets

User Interface Components

Each component is an object with

- Bounding box
- Paint method for drawing itself
  - Drawn in the component's coordinate system
- Callbacks to process input events
  - Mouse clicks, typed keys

```java
public void paint(Graphics g) {
    g.fillRect(…); // interior
    g.drawString(…); // label
    g.drawRect(…); // outline
}
```
2D Graphics Model

Widget canvas and coordinate system
- Origin often at top-left, increasing down and to the right
- Units depend on output medium (e.g., pixels for screen)
- Rendering methods
  - Draw, fill shapes
  - Draw text strings
  - Draw images

Composing a User Interface

Label TextArea

Buttons

How might we instruct the computer to generate this layout?
Absolute Layout

But this is inflexible and doesn’t scale or resize well.

Containment Hierarchy
Component Layout

- Each container is responsible for allocating space for and positioning its contents

```
Component Layout

- Each container is responsible for allocating space for and positioning its contents

Border Layout (direct placement)

```

“Struts and Springs” (simple constraint-based layout)

```
Layout in Flash/Flex

```

```
Component Layout

- Each container is responsible for allocating space for and positioning its contents

Border Layout (direct placement)

```

“Struts and Springs” (simple constraint-based layout)

```

```
Layout in Flash/Flex

```
What are Flash and Flex?

Flex
- Framework for web applications
- Implemented using MXML and ActionScript
- Contains library of components
- Quickly prototype interfaces in MXML

Flash (actually, ActionScript)
- What Flash Player runs
- JavaScript-like syntax
- Object-oriented, procedural language
- Use to create custom components, event handling

Flex Widgets
Component Layout in Flex

One Flex Layout
One Flex Layout (XML)

```xml
<?xml version="1.0" encoding="utf-8"?>
<mx:Application
xmlns:mx="http://www.adobe.com/2006/mxml"
layout="absolute">

<mx:Canvas>
<mx:TabNavigator width="117" height="100">
<mx:Canvas label="Tab 1" width="100%" height="100">
<mx:Text x="17" y="21" width="81" height="25" text="This is tab 1"/>
</mx:Canvas>
<mx:Canvas label="Tab2" width="100%" height="100%">
<mx:Text x="17" y="21" width="81" height="25" text="This is tab 2"/>
</mx:Canvas>
</mx:TabNavigator>
</mx:Canvas>
</mx:Application>
```

Another Flex Layout

Form

Form

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem

FormItem
Flex Layout XML

```xml
<?xml version="1.0" encoding="utf-8"?>
<mx:Form xmlns:mx="http://www.adobe.com/2006/mxml"
    initialize="{init();}">
    <mx:FormItem label="Task" required="true">
        <mx:TextInput id="taskname" width="200"/>
    </mx:FormItem>
    <mx:FormItem label="Percentage" required="true">
        <mx:TextInput id="percentage" width="200"/>
    </mx:FormItem>
    <mx:FormItem label="Start Date (MM/DD/YYYY)" required="false">
        <mx:TextInput id="startDate" width="200"/>
    </mx:FormItem>
    <mx:FormItem>
        <!-- User clicks Button to trigger validation. -->
        <mx:Button id="submit" label="Submit" click="{addTask();}"/>
    </mx:FormItem>
</mx:Form>
```

Roll your own...
Roll your own…

Flex Event Handling

- Every component (i.e., objects that extend UIComponent) dispatch events corresponding to different interactions.
- Classes that extend EventDispatcher can dispatch and listen to events, pre- or user-defined

- Examples events include:
  - MouseEvent.MOUSE_MOVE, CLICK
  - KeyboardEvent.KEY_DOWN
  - FlexEvent.BUTTON_DOWN
Flex Event Handling

Three phases: Capturing, Targeting, Bubbling

Capturing
Flash Player traverses the display list from root to the target’s parent for event listeners.

Targeting
The event listener is called on the target.

Bubbling (certain events)
Flash Player traverses the display list from target to root.

Flex Event Handling

There are a few ways to specify event handlers in Flex. The code below shows inline specification in MXML. You can also use the addEventListener() function in ActionScript.

```xml
<?xml version="1.0" encoding="utf-8"?>
  <mx:Canvas scaleX="2" scaleY="2">
    <mx:Label x="0" y="0" id="lab" width="140" height="28" fontSize="20"/>
    <mx:Button x="0" y="26" label="Click Me!"
      buttonDown="lab.text='Hello';" click="lab.text='World!';"
      width="140" height="38"/>
  </mx:Canvas>
</mx:Application>
```
User input is modeled as “events” that must be handled by the system and applications.

Examples:
- Mouse input (and touch, pen, etc.)
  - Mouse entered, exited, moved, clicked, dragged
  - Inferred events: double-clicks, gestures
- Keyboard (key down, key up)
- Sensor inputs
- Window movement, resizing
Anatomy of an Event

Encapsulates info needed for handlers to react to input
- Event Type (mouse moved, key down, etc)
- Event Source (the input component)
- Timestamp (when did event occur)
- Modifiers (Ctrl, Shift, Alt, etc)
- Event Content
  - Mouse: x,y coordinates, button pressed, # clicks
  - Keyboard: which key was pressed

Abstracting Events

Level of abstraction may vary. Consider:

- Mouse down vs. double click vs. drag
- Pen move vs. gesture
Callbacks

```java
onMouseOver(Event e) {...}
onMouseUp(Event e) {...}
onMouseDown(Event e) {...}
onMouseClick(Event e) {...}
```

Slider

Event Dispatch Loop

- Mouse moved \((t_0, x, y)\)
- Event Queue
  - Queue of input events
- Event Loop (runs in dedicated thread)
  - Remove next event from queue
  - Determine event type
  - Find proper component(s)
  - Invoke callbacks on components
    - Repeat, or wait until event arrives
- Component
  - Invoked callback method
  - Update application state
  - Request repaint, if needed
Event Dispatch

Event Queue
- Mouse moved \((t_0,x,y)\)
- Mouse pressed \((t_1,x,y,1)\)
- Mouse dragged \((t_2,x,y,1)\)
- Key typed \((t_3, ‘F1’)\)
- ...

(queues and dispatches incoming events in a dedicated thread)

`/* callback for TextArea */
public void mouseMoved(e) {
    // process mouse moved event
}

Interactor Tree

Display Screen
- Outer Win [black]
  - Inner Win [green]
    - Result Win [tan]
    - Keypad [Teal]
      - = button
      - - button
      - + button
      - 0 button
Demo

- Walk through example code for layouts we saw earlier and the sample code for the first assignment
- Explore ActionScript’s event handling model