- Provides visual access to tools
- Supports recognition over recall
- Animations disrupt interaction
- Paging makes some items hard to find
Model-View-Controller and Event Driven UI

CS 160: User Interfaces
Wesley Willett

Includes slides based on those of James Landay & Jeffrey Heer

Review: Modes
Review Depicting Processes & Storyboarding

Due Today (before class)

- Contextual Inquiry and Task Analysis
Assignment (Due Feb. 26)

- Android Intro Application
  - Build a simple application for searching and browsing Flickr photos using Android
  - Individual assignment
  - Requires significant work – get started early

- Emphasis on:
  - Designing a UI for searching / browsing
  - Creating appropriate Activity & Intent objects
  - Handling Activity lifecycle

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
Component/ Widget Model

Encapsulation and organization of interactive components ("widgets")
- Typically using a class hierarchy with a top-level "Component" type implementing 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

What are Some Examples of Components?
What are Some Examples of Components?

- Windows
- Layout panels
- Drawing panes
- Buttons
- Sliders
- Scrollbars
- Images
- Dropdown boxes
- Toolbars
- Menus
- Dialogue Boxes
- Progress indicators
- Video
- Icons
- Links
- Checkboxes
- Radio buttons
- Etc.

Periodic Table of Motif Widgets
Java Swing Widgets

Windows Vista Widgets
Android Widgets

• 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

User Interface Components

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

- Every component is a clipped drawing canvas with a coordinate system
  - Origin typically at top-left, increasing down and to the right
  - Units depend on the output medium (e.g., pixels for screen)
- Rendering methods
  - Draw, fill shapes
  - Draw text strings
  - Draw images

Composing a User Interface

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 in Android

View – Stores layout and content for a rectangular region. Handles measuring and layout, drawing, focus change, scrolling, and input for that region.

Viewgroup – Contains and manages other views. Base class for ‘layouts’.
One Android Layout

Component Layout in Android

Children must specify layout parameters appropriate for their parent.
Another Android Layout

Table Layout

Android Layout XML

```xml
<TableLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:stretchColumns="1">
    <TableRow>
        <TextView
            android:text="User" android:textAlign="end"
            android:padding="3dip" />
        <EditText
            android:id="@+id/username" android:padding="3dip" />
    </TableRow>
    <TableRow>
        <TextView
            android:text="Password" android:textAlign="end"
            android:padding="3dip" />
        <EditText
            android:id="@+id/pass" android:password="true" android:padding="3dip" />
    </TableRow>
    <TableRow android:gravity="right">
        <Button
            android:id="@+id/cancel" android:text="@string/table_cancel" />
        <Button
            android:id="@+id/login" android:text="@string/table_login" />
    </TableRow>
</TableLayout>
```
Another Android Layout

Some Android Layouts

```xml
<RelativeLayout ...>
    <TextView android:id="@+id/label"
    .../>
    <EditText android:id="@+id/entry"
        android:layout_below="@id/label"
    .../>
    <Button android:id="@+id/ok"
        android:layout_below="@id/entry"
        android:layout_alignParentRight="true"
        android:layout_marginLeft="10dp"
    .../>
    <Button android:layout_toLeft="@id/ok"
        android:layout_alignTop="@id/ok"
    .../>
</RelativeLayout>
```
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

An event encapsulates the information needed for handlers to react to the 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

Events

Level of abstraction may vary. Consider:

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

Slider

- `onMouseOver(EventArgs){…}`
- `onMouseUp(EventArgs){…}`
- `onMouseDown(EventArgs){…}`
- `onMouseClick(EventArgs){…}`
- `onMouseClick(EventArgs){…}`

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
}

Demo

Explore Java’s event handling model
Use debugger to walk into Swing internals
Need source from Sun, provided w/ JDK
Some classes expose callback methods which can be overridden with custom handlers.

Examples include:
- `Activity.onKeyDown(int keyCode, KeyEvent event)`
- `View.onWindowFocusChanged(boolean hasWindowFocus)`

Some important callbacks aren’t exposed and external listener classes must be attached. (e.g. handling button clicks)

```java
public class SendResult extends Activity{
    protected void onCreate(Bundle savedInstanceState){
        ...
        // Listen for button clicks.
        Button button = (Button)findViewById(R.id.myButton);
        button.setOnClickListener(myButtonListener);
    }
    private OnClickListener myButtonListener = new OnClickListener(){
        public void onClick(View v){
            setResult(RESULT_OK, "My Button Was Clicked!");
            finish();
        }
    };
}
```
Model-View-Controller Architecture

Architecture for interactive apps
- introduced by Smalltalk developers at PARC
Partitions application in a way that is
- scalable
- maintainable
Example Application

Blue circles: 4
Cardinal squares: 2

Model

Information the app is trying to manipulate
Representation of real world objects
- circuit for a CAD program
  • logic gates and wires connecting them
- shapes in a drawing program
  • geometry and color
View

Implements a visual display of the model
May have multiple views
- e.g., shape view and numerical view

Multiple Views

Blue circles: 4
Cardinal squares: 2
**View**

- Implements a visual display of the model
- May have multiple views
  - e.g., shape view and numerical view
- Any time the model is changed, each view must be notified so that it can change *later*
  - e.g., adding a new shape

**Controller**

- Receives all input events from the user
- Decides what they mean and what to do
  - communicates with view to determine the objects being manipulated (e.g., selection)
  - calls model methods to make changes on objects
    - model makes change and notifies views to update
Controller

Blue circles: 3
Cardinal squares: 2

Click!

Controller

Blue circles: 4
Cardinal squares: 2
Relationship of View & Controller

“pattern of behavior in response to user events (controller issues) is independent of visual geometry (view issues)” – Olsen, Chapter 5.2

Controller must contact view to interpret what user events mean (e.g., selection)
Combining View & Controller

View and controller are tightly intertwined
- lots of communication between the two
Almost always occur in pairs
- i.e., for each view, need a separate controller
Many architectures combine into a single class

Why MVC?
Combining MVC into one class will not scale
- model may have more than one view
  • each is different and needs update when model changes
Separation eases maintenance and extensibility
- easy to add a new view later
- model info can be extended, but old views still work
- can change a view later, e.g., draw shapes in 3-d
  (recall, view handles selection)
- flexibility of changing input handling when using separate controllers
Adding Views Later

Blue circles: 4
Cardinal squares: 2

Changing the Display

How do we redraw when shape moves?
Moving Cardinal Square

Erase w/ Background Color and Redraw
Changing the Display

Erase and redraw
- using background color to erase fails
- drawing shape in new position loses ordering

Move in model and then redraw view
- change position of shapes in model
- model keeps shapes in a desired order
- tell all views to redraw themselves in order
- slow for large / complex drawings
  * flashing! (can solve w/ double buffering)

Damage / Redraw Method

View informs windowing system of areas that need to be updated (i.e., damaged)
- does not redraw them at this time...

Windowing system
- batches updates
- clips them to visible portions of window

Next time waiting for input
- windowing system calls Repaint method
  * passes region that needs to be updated
Damage old, Change position in model, Damage new

Blue circles: 4
Cardinal squares: 2

Event Flow

Creating a new shape
Event Flow (cont.)

Assume blue circle selected

Event Flow (cont.)

Press mouse over tentative position
Windowing system identifies proper window for event
Controller for drawing area gets mouse click event
Checks mode and sees “circle”
Calls model’s AddCircle method with new position
AddCircle adds new circle to model's list of objects
Model then notifies list of views of change
  - drawing area view and text summary view
Views notifies windowing system of damage
  - both views notify WS without making changes yet!
    • model may override

Views return to model, which returns to controller
Controller returns to event handler
Event handler notices damage requests pending and responds
If one of the views was obscured, it would be ignored
Event Flow (cont.)

Event handler calls views’ Repaint methods with damaged areas.
Views redraw all objects in model that are in damaged area.

Dragging at Interactive Speeds

Damage old, move, damage new method may be too slow:
- must take less than ~100 ms to be smooth.

Solutions:
- don’t draw object, draw an outline (cartoon)
  • use XOR to erase fast (problems w/ color)
- save portion of frame buffer before dragging
  • draw bitmap rather than redraw the component
- modern hardware often alleviates the problem
Review

Event-Driven Interfaces
- Hierarchy of components or widgets
- Input events dispatched to components
- Components process events with callback methods

Model-View-Controller
- Break up a component into
  - **Model** of the data backing the widget(s)
  - **View** determining the look of the widget
  - **Controller** for handling input events
- Provides scalability and extensibility

Looking forward

- Containment hierarchy model is now over 20 years old, designed in a context of significantly less processing and graphics power.
- Dominant model in use today, and still quite useful, but in many cases limiting.
- Limitations include:
  - Assumes rectangular components
  - Limited support for animation
  - Level of extensibility (varies by toolkit)
- Suitability for next-generation interfaces?
For Next Time

- Readings:

- Get started on Android programming assignment!