CS 160: User Interface Design

Video, Widgets, Events 02/28/11
Knowledge Navigator

Apple Knowledge Navigator, 1987

http://video.google.com/videoplay?docid=100196171226719096#
Graded on bSpace

Reading Responses for Jan.
1 pt [0, 0.5, 1, 1.15] each

Programming Assignment II

Heuristic Evaluation
Mini Assignment

Assigned Today,
finish by class on Wednesday:
Mid-semester feedback
CS160 Spring 2011 Midsemester Feedback

Please take a few minutes to review your experience of the course thus far. Your answers to the questions below will help us plan for the second half of the semester (and future offerings of CS160). Your answers are anonymous.

How worthwhile is this class so far?
- [ ] 1 - Not worthwhile at all
- [ ] 2
- [ ] 3
- [ ] 4
- [ ] 5
- [ ] 6
- [ ] 7 - Very worthwhile

How effective is your professor?
- [ ] 1 - Not effective at all
- [ ] 2
- [ ] 3
- [ ] 4
- [ ] 5
- [ ] 6
- [ ] 7 - Very effective
Topics for this week

Review of Video Prototyping

Interactive Application Programming
Component Model
Layout
Event-Driven User Interfaces

Wednesday: Model-View-Controller, Threading
Architecture for interactive components
Why do we need it?
Changing the display
Video Prototyping, Continued
Video Prototypes

Narrative: You control the story!
Use existing software & images of real settings
Narration optional (but required for your assignment!)
   Explain events while others move images/illustrate interaction

With good storyboards, should be able to create video prototype in a few hours
Creating a Video Prototype

1) Review field data about users & work practices
2) Create use scenario in words
3) Develop storyboard of each action/event with annotations explaining the scene. Put each element on a card. *This will save you a lot of time later.*
4) Shoot a video clip for each storyboard card
   
   Either Live action or UI Screen recording.
   
   Hold last frame of a section/shot for 1s
5) Use titles to separate clips (keep it onscreen for 3s)
CLUSTER
Video Prototype

Andy Hou, Kevin Chiu
Tips & Tricks

Add structure to better explain context
Begin with a title
Follow with an “establishing shot”
Switch between showing UI and showing live shots
At the end, connect back to the original motivation

Editing: Keep it simple!
Live video is most convincing to show context,
but still photos + narration also work
Don’t obsess about transitions and composition – just tell your story.
Tips & Tricks: Showing UI Interaction

Three options (mix and match for your project):

Link screens in Balsamiq and screencapture
Advantage: No other editing required
Disadvantage: Only discrete “clicks” are supported

Export images from Balsamiq and add interaction in PPT/Keynote/iMovie/…
Advantage: Can show continuous interaction
Disadvantage: More work

Print out Balsamiq designs and shoot live video
Advantage: Can show continuous interaction; faster than digital editing
Summary

Informal prototypes allow you to design (and test!) before writing code.

Rapid evolution and elimination of many problems happens in this phase.

Use informal paper + software prototypes for exploration (what should the design be?).

Once you have a compelling design, create a video for communication and persuasion.
Interactive Application Programming
In the beginning...
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
Do some work…
Prompt user for input
Wait for user input
Process user input…
Do some more work…
Exit
// Java Example:

Console console = System.console();
String name = console.readLine("Your name:");
System.out.println("You have entered: " + name);
String pass= console.readLine("Password:");
System.out.println("…");
Minimal “interactive” program

Do until a quit command: {
    wait for user input
    process it…
    (optionally) update display
}

Minimal “interactive” program

Do until a quit command: {
    wait for user input
    switch (input-cmd) {
        case insert: do-insert(...)
        case delete: do-delete(...)
        case backspace: ...
        (optionally) update display
    }
}
Minimal “interactive” program

Can’t use this (global) approach for window systems, because the result of a user command **depends on the active window** (and the active component within that window).

Too many possible combinations of input x target window, and window structure is dynamic.
GUI Toolkits

Most user interfaces today are written using toolkits (e.g., QT, Cocoa, Java Swing, GTK, Android SDK,…)
Toolkits come with libraries of interactive elements (widgets) and layouts
Frequently used interactive components

Toolkits also define an architecture:
A standard way how input and output are handled
Usually wrap main() – application programmer writes pieces of code that plug into the architecture
The architecture specifies how to write new widgets for the library
Widgets
Android Widgets
Java Swing Widgets
Windows Vista Widgets
# Mac Cocoa Widgets

## Interface Builder - Library

<table>
<thead>
<tr>
<th>Button</th>
<th>Slider</th>
<th>Progress Indicator</th>
<th>Tool Tip</th>
<th>Text</th>
<th>Combo</th>
<th>Check</th>
<th>Pop Up</th>
</tr>
</thead>
<tbody>
<tr>
<td>Radio Button</td>
<td>Toggle Button</td>
<td>Volume Control</td>
<td></td>
<td>Control Panel</td>
<td>Custom</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Label</td>
<td>Line Control</td>
<td>Optional Text Field</td>
<td></td>
<td>Multi-line Text Field</td>
<td>OpenGL</td>
<td>$</td>
<td><img src="https://example.com/logo.png" alt="" /></td>
</tr>
<tr>
<td>Multiline Label</td>
<td>Xcode</td>
<td>Text Field</td>
<td></td>
<td>Multi-line Text Field</td>
<td>Multi-line Text Field</td>
<td>2/12</td>
<td></td>
</tr>
</tbody>
</table>

![Image of various Mac Cocoa Widgets in Interface Builder library](https://example.com/image.png)
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
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:
```java
public void paint(Graphics g) {
    g.fillRect(...); // interior
    g.drawString(...); // label
    g.drawRect(...); // outline
}
```

Cocoa:
```cocoa
(void)drawRect:(NSRect)rect
```
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
Working with Widgets

Make the common case fast and the uncommon case possible.

Common case: assemble standard widgets into a layout
Uncommon case: write your own widget.

Custom Components in AndroidSDK:
• Extend View class
• Paint method: Override onDraw()
• Bounding box: Override onMeasure()
• Callbacks: Override onTouchEvent(), onKeyDown, ...

Composing a User Interface

How might we instruct the computer to generate this layout?
Absolute layout is inflexible and doesn’t scale or resize well.
(But: great for prototyping because it’s fast!)
Containment Hierarchy

- Window
  - Panel
    - Label
    - TextArea
    - Panel
  - Button
  - Button
Containment Hierarchy

Principle: Each container is responsible for allocating space and positioning its contents.
Common Hierarchical Layouts

1D Horizontal or Vertical List
2D Grid
Constraint-based Layout (Struts+Springs)
Example Declarative Layout (WPF)

```
<StackPanel>
  <Label>Enter Text:</Label>
  <TextBox TextWrapping="Wrap">…</TextBox>
  <StackPanel Orientation="Horizontal"
               HorizontalAlignment="Right">
    <Button>Ok</Button>
    <Button>Cancel</Button>
  </StackPanel>
</StackPanel>
```
Example Declarative Layout (WPF)

```xml
<StackPanel>
  <Label>Enter Text:</Label>
  <TextBox TextWrapping="Wrap">...</TextBox>
  <StackPanel Orientation="Horizontal"
               HorizontalAlignment="Right">
    <Button>Ok</Button>
    <Button>Cancel</Button>
  </StackPanel>
</StackPanel>
```
Example Declarative Layout (WPF)

```xml
<StackPanel>
  <Label>Enter Text:</Label>
  <TextBox TextWrapping="Wrap">…</TextBox>
  <StackPanel Orientation="Horizontal" HorizontalAlignment="Right">
    <Button>Ok</Button>
    <Button>Cancel</Button>
  </StackPanel>
</StackPanel>
```
**Android Layouts**

```
<LinearLayout orientation="horizontal">
    <TextView text="red" background="..."/>
    <TextView text="green" background="..."/>
    <TextView text="blue" background="..."/>
    <TextView text="yellow" background="..."/>
</LinearLayout>
```
Android Layouts

<LinearLayout orientation="vertical">
   <TextView text="row one" .../>
   <TextView text="row two" .../>
   <TextView text="row three" .../>
   <TextView text="row four" .../>
</LinearLayout>
In Android

```xml
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout orientation="vertical">
  <TextView text="Enter Text:"></TextView>
  <EditText text="lorem ipsum..."></EditText>
  <LinearLayout orientation="horizontal">
    <Button text="Ok"></Button>
    <Button text="Cancel"></Button>
  </LinearLayout>
</LinearLayout>
```
Layout in Cocoa: Springs + Struts

Interface Builder Demo
Specifying Layout

**Declarative**
e.g., HTML, XAML, MXML,…

**Procedural**
e.g., Java Swing

GUI Builders exist for either approach (but generating procedural code is brittle)

Is your UI layout determined statically or dynamically at runtime? If at runtime, may need procedural approach.

```
<StackPanel>
  <Label>Enter Text:</Label>
  <TextBox TextWrapping="Wrap">…</TextBox>
  <StackPanel Orientation="Horizontal"
                HorizontalAlignment="Right">
    <Button>Ok</Button>
    <Button>Cancel</Button>
  </StackPanel>
</StackPanel>
```
Specifying Layout

Declarative
e.g., HTML, XAML, MXML,…

Procedural
e.g., Java Swing

GUI Builders exist for either approach (but generating procedural code is brittle)

Is your UI layout determined statically or dynamically at runtime? If at runtime, may need procedural approach.

```java
public void init() {
    Container c = getContentPane();
    c.setLayout(new BorderLayout());
    c.add(new JButton("One"), BorderLayout.NORTH);
    c.add(new JButton("Two"), BorderLayout.WEST);
    c.add(new JButton("Three"), BorderLayout.CENTER);
}
```
Events
Events

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
Callbacks

Slider

- onMouseOver(Event e){...}
- onMouseDown(Event e){...}
- onMouseUp(Event e){...}
- onMouseClick(Event e){...}

mouse over

click

drag
Event Dispatch

Application

Cocoa

Carbon

Window Server

I/O Kit

Mouse

Keyboard

Tablet & stylus

Application environments

Application Services

Core Services

Kernel environment

Apple, Cocoa Event-Handling Guide
Event Dispatch Loop

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

Mouse moved \((t_0,x,y)\)
Event Dispatch Loop

1) Events from input devices enter here

2) Event is added to FIFO event queue

3) Main loop processes one event per iteration

Apple, Cocoa Event-Handling Guide
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
}
Mouse/Touch vs. Keyboard Events

Mouse Events are (usually) routed to the top-most (in z-order) visible component underneath the cursor using **hit testing**.

Exception: “captured” mouse events after beginning interaction

Keyboard events are (usually) routed to the component that has **key focus**.

Exceptions: keys that change focus, accelerator keys
Key Focus: Form Example

Contact Information
Please fill in all fields.

First name

Last name

Street address

City

Postal code

State

Business phone
ext.

Mobile phone

Home phone

E-mail address

1  2  3  4  5  6
Abstracting Events

Level of abstraction may vary. Consider:

**Mouse down** vs. **double click** vs. **drag**

**Pen move** vs. **gesture**
Single Tap vs. Double Tap (or Click)

How should the application be notified of events that have duration?

Graphics: Apple iPhone Programming Guide
Single Tap vs. Double Tap (or Click)

Option 1: Two separate events

Graphics: Apple iPhone Programming Guide
Single Tap vs. Double Tap (or Click)

Option 1: Two separate events

Graphics: Apple iPhone Programming Guide
Single Tap vs. Double Tap (or Click)

How do you prevent this?

Graphics: Apple iPhone Programming Guide
Single Tap vs. Double Tap (or Click)

Graphics: Apple iPhone Programming Guide
Single Tap vs. Double Tap (or Click)

Advantage: simple model for programmer
Disadvantage: every single tap incurs latency

Graphics: Apple iPhone Programming Guide
Single Tap vs. Double Tap (or Click)

Option 2: Let the programmer deal with it.

onTouchDown()  onTouchUp()  onTouchDown()  onTouchUp()
Single Tap vs. Double Tap (or Click)

Option 2: Let the programmer deal with it.

Graphics: Apple iPhone Programming Guide
Single Tap vs. Double Tap (or Click)

Option 2: Let the programmer deal with it.

If you know you don’t need double-taps, no latency.

onTouchDown()
tapCount = 1

onTouchUp()
tapCount = 1

handleTap()
Option 2: Let the programmer deal with it.

If you know you **do** need double-taps, emulate option 1.

```
tonTouchDown()
tapCount = 1
```

```
tonTouchUp()
tapCount = 1
```

Request single tap w/ delay:

```
handleTap()
```
Single Tap vs. Double Tap (or Click)

Option 2: Let the programmer deal with it.

- `onTouchDown()`
  - `tapCount = 1`

- `onTouchUp()`
  - `tapCount = 1`

- `onTouchDown()`
  - `tapCount = 2`

Request single tap w/ delay:

Handle double tap

Graphics: Apple iPhone Programming Guide
Single Tap vs. Double Tap (or Click)

Option 2: Let the programmer deal with it.

Handle double tap

onTouchDown() tapCount = 1
onTouchUp() tapCount = 1
onTouchDown() tapCount = 2
onTouchUp() tapCount = 2

Request single tap w/ delay:

Graphics: Apple iPhone Programming Guide
Detecting Gestures

Two different kinds of gestures:

**Continuous manipulation gestures:**
(e.g., pinch-to-zoom)

**Stroke recognition gestures**
(e.g., Handwriting recognition, Swype)

Android Gesture Search:
http://www.youtube.com/watch?v=umos1GZKbKw
Detecting Gestures

Most event architectures assume there is a single, “correct” response to a single input event. This model is not well suited to describing multitouch interactions. Why? Recognition, co-existence of different gesture types complicate the picture: input can match multiple possible interpretations. How to deal with uncertainty is still a research topic in HCI.
Next Time

Model-View-Controller, Threading in User Interfaces
Don’t forget to read and submit comment!

Continue work on Programming Assignment III, Prototype!