Architectures for UI software

- Why is interactive software hard?
- Event based systems
- Event loops
- Frameworks

Interactive software is hard to build

- “UI” accounts for more than 50 percent of:
  (Wise, 1997)
  - Design effort
  - Implementation effort
  - Maintenance effort
  - Code size

Why is interactive software hard to build (process reasons)?

- Continuous changes (user-centered design)
- Multiple, collaborating disciplines
  - Interaction designer
  - Programmer
  - Graphic designer
  - Domain specialist

Why is interactive software hard to build (technology reasons)?

- Multiple sources of input
- Need to provide user control
  - Program must respond to any kind of input at any time
- Tight response time required for feedback
  - Continuous motor action (i.e., pointing): 0.1 s
  - Basic input task (i.e., opening dialog box): 1 s
  - Complex tasks: 10 s

How can tools help with problems of the process?

- Separation of tools for roles:
  - Screen painter: Interaction designer
  - Application framework: Programmer
  - Style sheet: Graphic designer
  - Example: Gould et al.’s ITS
- Separate volatile and non-volatile portions:
  - Volatile: Text strings, dialog layout
  - Nonvolatile: Dialog box processing

How can tools help with problems of the technology?

- Standardized internal architectures
- Prewritten code
- Incremental modification
- Facilitate reusable designs
- We’ll focus on these this term
Architectures for interactive software: 1970s

- Architecture mirrored sequential structure of the task:
  - Ask how many widgets
  - Enter number
  - Ask cash or credit
  - Enter A or R
  - if (A) then do_cash
  - else do_credit
  - Ask for address
  - ...

The event loop: early 1980s

- Want user to be in control!
- New interface paradigm: direct manipulation of the virtual world, design software around a conceptual model
- User presented with a collection of operations, can do any at any time

Basic event loop structure

- Basic structure
  - program initializes
  - user initializes input event
  - while event not “quit”
  - change program state
  - user initiates input event
  - program shuts down
- But it gets complex quickly!

Structure of an event

- Types: Mouse move, mouse click, keystroke, timer alarms, sometimes OS events, sometimes interprocess communication
- Event record:
  - Type (MOUSE_MOVE, KEY, …)
  - Time (usually)
  - Mouse location
  - Flags

Dispatching events

- What we do with an event depends upon
  - The kind of event
  - The location of the mouse (mouse events) or input focus
  - The state of the program
- Picking the correct response to an event is complex!

Examples

- (Programs 2-5 and 2-8 from Chernicoff’s Macintosh Revealed, 2nd Ed.)
### Prewritten code: Widgets

- Libraries of predefined basic interactions
- Screen display plus underlying code
- Examples: Scrollbars, Buttons, Menus, Standard file dialogs, …
- Pass events to the widget, it handles details
- Ensures *compatibility* as well

### Prewritten code: Application frameworks

- Widgets don’t handle high-level dispatching
  - programmer must decide an event is appropriate to a widget
- Application frameworks are prewritten event loops
  - Dispatching has already been coded
- The framework passes the correct event to the correct object

### Incremental modification

- Examples: AWT, MFC, Apple Application Kit (MacOS X), Motif
- Programmer *overrides* default behavior:
  - When a window needs to be redrawn, MFC calls `OnDraw`
  - Default `OnDraw` does nothing
  - Programmer overrides `OnDraw` to draw a specific kind of window

### Reusability

- Model-View-Controller architecture
- Rarely implemented completely
- Model: Persistent data
- View: Display of data
- Isolates app-specific code in Controllers
- Increases reusability of Model and View