Organization of User Interface Software
Administration

- Questions about assignments due and assignments assigned
What we will talk about

- Ways to organize UI code
- Different “models” of user interfaces as systems/programs
  - How they are structured and the parts that make them up
  - Conceptually and in practice
  - Separation of UI and rest of software = “semantics”
Semantic

- Functionality of system; what can be expressed
- What information is needed for each operation on object
- What errors can occur
- Semantic vs. UI is key issue in UI tools
- “Semantic Feedback”
  - Depends on meaning of items
  - Example: only appropriate items highlight during drag
Conceptual

- Key application concepts that must be understood by user
- User model
  - objects and classes of objects
  - Relationships among them
  - Operations on them
- E.g. text editor
  - Objects = characters, files, paragraphs
  - Relationships = files contain paragraphs contain characters
  - Operations = insert, delete, etc.
The User Interface

• Typically want to think of “UI” as only one component of an overall system
  – The part that “deals with the user”
  – Distinct from the “functional core” (AKA the “application”)
Separation of UI from “Appl”

- Really good reasons to want separation of UI 
  (in general: “separation of concerns”)
  - Modularity (good software design)
  - Different expertise needed
  - Don’t want to iterate the whole thing
Unfortunately this is typically very hard to do in practice

- More and more of interactive programs are tightly coupled to UI (in some cases everything)
  - Generally need to structure around user concepts
  - UI structure “sneaks into” application
Separation of concerns is a central theme of UI organization

• A continual challenge
• A continual tension and tradeoff
UI tasks

• So far have:

• Clearly more structure could be useful
UI tasks

- Basic parts of UI
UI tasks

• Basic flow
UI tasks

- Basic flow

Note relation to: Norman’s 7 stages
UI tasks

- Basic flow

Note relation to: Norman’s 7 stages
How do we connect these disparate parts into working whole

- Tempting to organize system modules around these boxes
  - One module for input, one for output, etc.
  - Has been tried

("Seeheim model" ~1983)
Didn’t work real well
Organizing UI as “3 big boxes” doesn’t work well because...

- Modern (“direct manipulation”) interfaces tend to be collections of quasi-independent agents
  - Each “object of interest” is separate (but still needs the 3 parts)
  - e.g. a button
    - has “button-like” screen appearance
    - acts on input in a “button-like” way
    - etc.
Leads to object-based organization
Object-oriented techniques

• Key features
  – Separation of “objects of interest” into encapsulated entities that implement that “object”
    • Store information about it
      – It’s “state” (“properties” in Flex)
    • Provide implementation of actions on that data (“methods”)
  – Combines data & action into one thing instead of traditional approach of data & procedures operating on it
Object-oriented techniques

• Key features
  – Abstract (& hide) the implementation details
    • Present “what” to outside world so that details of “how” can be changed w/o breaking other code
      – Classically no data access, only call methods
    • Reduces complexity by limiting dependencies
    • Example: Stack data structure
      – Just provide operations: push(), pop(), isEmpty()
      – Could be implemented with array or linked list
      – Can change implementation without breaking any code that uses stacks!
Object-oriented techniques

• Key features
  – Support reuse of code
    • Can base new code (new classes) on old code
  – Objects defined by a class
    • Represents of “type of thing”
    • Provides definition of methods appropriate to that type of thing
    • Provides implementation
Object-oriented techniques

• Key features
  – Object created as an “instance” of the class
    • Object gets own storage and uses methods provided by class
  – New classes can be created by specialization of a class (“inheritance”, “subclassing”)
    • Selectively replace (“override”) implementation of methods and other details “inherited” from another class (“superclass”, “base class”)
  • Substitutability: Object of subclass can be used anywhere object of superclass is expected
Object-oriented techniques

- Became popular along with GUIs, direct manipulation
- Buttons, sliders, icons, act like separate entities (→ objects)
  - Have internal state, persistence
  - React according to “what they are”
- OO was originally developed (SmallTalk) and became popular (C++) largely due to GUIs
Leads to object-based organization
Leads to object-based organization

- Each object implements each aspect
  - In a way that reflects what it is
Leads to object-based organization

- Objects organized hierarchically
  - Normally reflecting spatial containment relationships

⇒ “Component trees”
Component Trees

- Central concept for UI org
- Everything is done through this tree
  - Build an interface == build a tree
  - Change an interface == change a tree
Challenge: Separation of concerns

- Challenge is doing all this different stuff in a single object without creating a hopelessly large and complicated beast
One organizational approach

- **Model-View-Controller (MVC)**
  - Smalltalk ~1980
  - **Idea:** Separate out parts
    - output / presentation (View)
    - user input (Controller)
    - “semantics” / data (Model)

- **Goals**
  - Different kinds of views and controllers for same model
  - Create (subclass?) a new model, then re-use existing views and controllers
  - Multiple views (and controllers) for one model
MVC
MVC

View

Controller

Model

345
MVC

- **Model**
  - Can be simple as an integer for a counter, or string for an text entry box
  - Or as complex as a molecular simulator

- **View**
  - Everything graphical (output)
  - Layout, subviews, composites

- **Controller**
  - Schedule interactions with other VCs
MVC interaction cycle

- User operates input device
- Controller notifies model to change
- Model broadcasts change notifications to its dependent views
- Views schedules update of screen
  - May query model to get all details
MVC issues

• Views and controllers tightly coupled
  – Rarely implemented separately in practice

• What is in each part?

• Complexity when we have sub-parts
  – Sub-views, sub-controllers, sub-models
Exercise: MVC partitioning

- File picker
- MP3 Player
- Text editor
What do we have to help us implement UI systems?

• Layered set of tools…

(A different way to slice concepts)
### Layers of UI Software

<table>
<thead>
<tr>
<th>Application Program</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>“High-Level” Tools</td>
<td></td>
</tr>
<tr>
<td>Toolkit</td>
<td></td>
</tr>
<tr>
<td>Window System</td>
<td></td>
</tr>
<tr>
<td>OS</td>
<td></td>
</tr>
<tr>
<td>Hardware</td>
<td></td>
</tr>
</tbody>
</table>

Tools Supporting Each Layer
Layers of UI Software as They Tend to Occur in Commercial Systems...

<table>
<thead>
<tr>
<th>Application Program</th>
<th>“High-Level” Tools</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>OS</td>
</tr>
<tr>
<td></td>
<td>Hardware</td>
</tr>
</tbody>
</table>

Tools Supporting Each Layer
## Layers of UI Software

<table>
<thead>
<tr>
<th>Application Program</th>
<th>Tools Supporting Each Layer</th>
</tr>
</thead>
<tbody>
<tr>
<td>“High-Level” Tools</td>
<td></td>
</tr>
<tr>
<td>Toolkit</td>
<td></td>
</tr>
<tr>
<td>Window System</td>
<td></td>
</tr>
<tr>
<td>OS</td>
<td></td>
</tr>
<tr>
<td>Hardware</td>
<td></td>
</tr>
</tbody>
</table>
# Layers of UI Software

<table>
<thead>
<tr>
<th>Layer</th>
<th>Description</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application Program</td>
<td></td>
<td></td>
</tr>
<tr>
<td>“High-Level” Tools Toolkit</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Window System</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hardware</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Primary focus of this class**

Embodies organization

Each Layer

Use these if you can! (Built on TK layer)
Quick Look at the tools landscape

• Today’s tools are highly successful
  – Window Managers, Toolkits, Interface Builders are ubiquitous
  – Most software built using them
  – Are based on many years of HCI research
Window Systems

• Provides a virtual device abstraction
  – Each program can act as if it has a complete control over screen & input
  – Window system manages and controls multiple contexts, logically separated, but implemented together
• Analogous to OS management of CPU and memory
Window Managers (History)

- Multiple (tiled) windows in research systems of 1960’s: NLS, etc.
- Smalltalk (1974) at Xerox PARC
- Successful because multiple windows help users manage scarce resources:
  - Screen space and input devices
  - Attention of users
  - Affordances for reminding and finding other work
Windows, components

• “Window System”
  – Programming interface
  – Output graphics operations to draw clipped to a window
  – Input from mouse and keyboard to appropriate window

• “Window Manager”
  – User interface to windows themselves
  – Decorations on windows
  – Mouse and keyboard commands to control windows.
Windows, cont.

• Many systems combine Window System and Window Manager
  – SunTools, Macintosh, MS Windows, NeXT
• Others allow different WM on same WS
  – X, NeWS
  – Allows diversity and user preference
• Also different WS on same hardware
  – SunTools, X, NeWS on Unix machines
Window System: Output Model

- Graphics commands that the programs can use
- All must go through window system so they are always clipped
  - Usually can only draw with what the window system provides
Window System: Output Model

• Oldest systems (SunTools, etc.) simple primitives
• Later (Macintosh, X Windows) more sophisticated
  – Filled polygons, splines, colors, clipping
  – Still, all 2-D objects
  – Extensions for 3D
• Newer systems (e.g., Java Swing) have quite sophisticated output model
  – Fully scalable, transparency, …
Window System: Input Model

- How input from user is handled.
- Most only support keyboard and mouse
- All modern WS use similar model:
  - Events generated and passed to applications
    - “Event records” containing significant details of a user input action
      - type of input, x,y of mouse, time, etc.
    - Processed asynchronously (queued)
A model for input handling
Semantic-Syntactic-Lexical levels

• Comes from analogy to programming languages
  – Lexical:
    characters form symbols
    (keywords, operators, comments, etc.)
  – Syntactic:
    symbols organized by a grammar
    (into constructs: procedures etc.)
  – Semantic:
    meaning derived from constructs
    (so code can be generator or lang. interpreted)
A model for input handling
Semantic-Syntactic-Lexical levels

• For UI
  – Lexical: the basic inputs
    • Events: e.g., Mouse movements, button and key presses
    • Often consider interactions with basic interactors (e.g., button press, menu selection) to be at this level even though these may have more detailed syntax
  – Syntactic: what is current “state” of the system and what can happen next
    • In modern systems often expressed by showing certain dialogs or disabling menus, etc.
  – Semantic: translation to meaning in the form or actions carried out for the user

• Note: good conceptual model, not necessarily good implementation model
Toolkits

- A library of components that can be manipulated by application programs.
- A component is a graphical object which can be manipulated by the user to input a certain type of value.
  - Also called “widget”, “control”, “interactor”
  - Menus, scroll bars, text entry fields, buttons, etc.
- Infrastructure for implementing and organizing components
  - E.g., managing component trees, redraw, input distribution, etc.
  - Sometimes called “intrinsics”
- Used directly only by programmers
  - Only a procedural interface.
Toolkits (cont.)

• Interface to applications is most typically done with “callback procedures”
  – Application says: “when this happens” (e.g., this button pressed), “call this routine”

• Issues with callbacks:
  – Can be hundreds or thousands distributed around system
    • Modularization compromised
  – Hard to deal with undo, etc.
Toolkit Advantages

• Consistent Look and Feel
  – Key insight of Macintosh toolbox
  – Path of least resistance was to be consistent

• Structuring the task

• Re-use of code
  – Just flat out a lot less work to use the toolkit library than to recreate
But...

• Can be hard to use:
  – Very large libraries
    • Can end up as a complicated mess
    • Very large manuals
  – No help with when and how to call what
Higher Level Tools

• Since toolkits are hard to use, higher-level support is helpful
  – Graphical layout tools
  – Higher-level frameworks
  – Older tools called “User Interface Management Systems”

• Successful research ⇒ industry
Graphical / Interactive Tools

• Create parts of UI by laying out components with a mouse
  – Examples: Menulay (1983), Trillium (1986), Jean-Marie Hullot from INRIA to NeXT
Graphical Interactive Tools

• Significant Advantages
  – Graphical parts done in an appropriate, graphical way
  – Accessible to non-programmers
Component Architectures

• Create applications out of loosely coupled *components* which are separately developed and compiled
  – In UI software, each component controls an area of the screen
  – Example: drawing component handles picture inside a document

• Invented by Andrew research project at CMU (1988)

• Now: OLE, OpenDoc, Visual Basic Controls (VBX Controls), ActiveX, Java Beans
Higher Level Tools are Good

• Use them if you can
But a bit of a warning:

• Be aware of the path of least resistance

• Tools have Whorfian effects
  – Change the way you think
  – Change what is possible
  → Change what you design
Questions about the lecture or readings?