

10 Custom Components

CS349 Winter 2012

Why Design Custom Controls?

- You design new controls to address a need
- Many examples of useless things
 - The segway
- Make sure you need a custom control before you design one



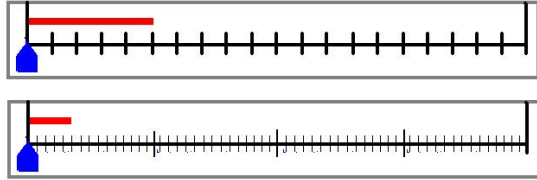
Custom Controls

- Custom component should:
 - Address a very specific need
 - Do one, well-defined task better than existing methods
 - Be reusable, customizable across applications
- To meet these goals, we need to consider two perspectives:
 - User's perspective
 - Developer's perspective
 - Both "users" of the component



Perspectives

- Designing Custom Components
 - User's perspective
 - Developers perspective
 - Design of simple controls

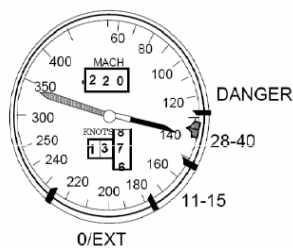


User's Perspective

- What problem is the user trying to solve?
- How does the user conceptualize the problem?
- How is the user currently solving the problem?
 - What tools and/or information does s/he use?
- How do we find the answers to the above questions?

Example: Cockpit Design

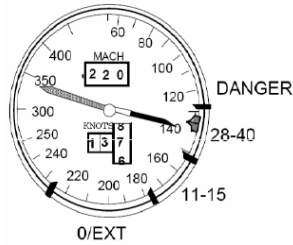
- As cockpits went “digital,” analog controls were replaced with digital controls
- However, soon they began to see problems
- Why?



From Hutchins' "How a Cockpit Remembers Its Speeds" (1995)

Example: Cockpit Design

- Pilots don't rely on speed numbers
- They set "speed bugs" to indicate minimum speeds at different aircraft weights
- They use spatial relationships to assess the situation
- Not possible with a digital airspeed indicator



From Hutchins' "How a Cockpit Remembers Its Speeds" (1995)

Undo/Redo in Photoshop

- Users sometimes rapidly execute undo/redo
 - But are not fixing a mistake
- Instead, they are assessing the result of their last action
- Need to look beyond what is being done, and ask why it is being done
- What purpose is the activity serving?
 - User may not be able to tell you
 - Why not?

Designing for Users

- Observations, interviews help us uncover user's real needs and motivations
- In designing new components and interfaces, we must understand users and the larger context of their work to be successful
 - CS 489
- First step is to understand how users approach interaction in interfaces
 - Models of user interaction

Models of Interaction

- Interaction styles vary extensively
 - From batch input to direct manipulation/virtual reality
- Interaction involves 2 participants
 - Human and computer
 - Interface translates between them
- Several models; we'll examine 2:
 - execution evaluation cycle
 - interaction framework



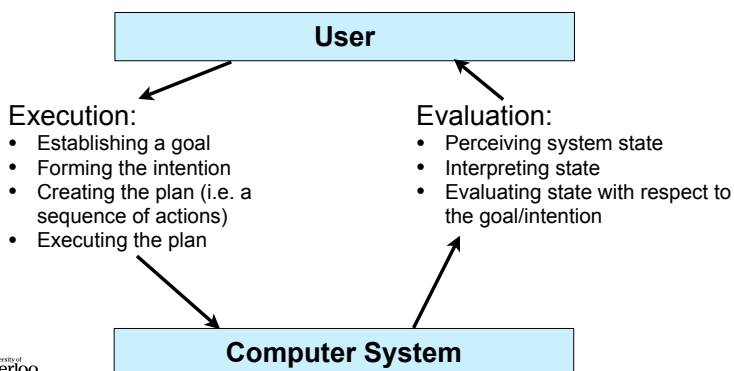
Terms of Interaction

- Some definitions:
 - Domain: An area of expertise and knowledge in some real-world activity
 - Tasks: Operations that manipulate concepts in the domain
 - Goal: Desired output from a task
 - Core language: The computational attributes of the domain relative to the system state
 - Task language: The psychological attributes of the domain relative to the user state.



Execution-Evaluation Cycle

- 2 stages with 7 steps
- Developed by Norman (1980)



What's Interesting

- Norman's interaction is very naïve
 - Execution involves:
 - Establishing a goal
 - Forming the intention
 - Creating the plan (i.e. a sequence of actions)
 - Executing the plan
 - Evaluation involves:
 - Perceiving system state
 - Interpreting state
 - Evaluating state wrt goal/intention
- However, Norman uses it to identify where user errors occur and thus where we can focus improvements



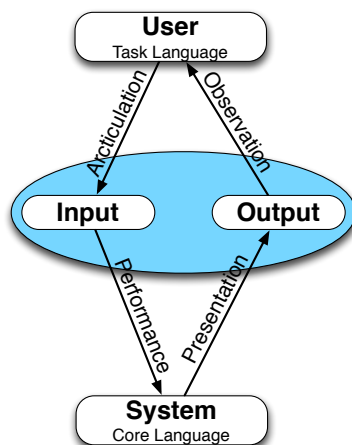
Types of User Error

- Norman identifies gulfs of execution and gulfs of evaluation
 - Gulf of execution: Difficulty in translating the user's intentions into actions allowed by system. Can the user carry out their intentions directly?
 - Gulf of evaluation: Difficulty in interpreting the state of the system to determine whether the expectations (goals) have been met.
- Problems
 - Only considers system as far as interface
 - Focuses on user's view of interaction
 - What about designer and their model?



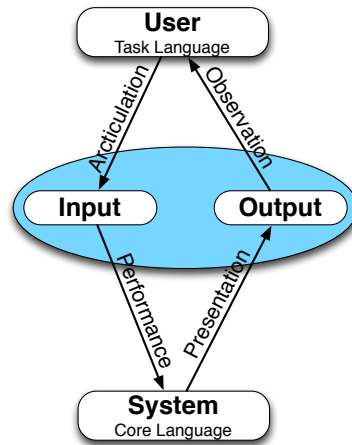
Interaction Framework

- Extends Norman's model:
 - Includes system state explicitly
- Four nodes:
 - System, User, Input and Output
 - Each node has own language:
 - System language = core language
 - User language = task language
 - Input and Output languages form the interface
 - Translates between core and task language



Note: Can Combine Models

- Articulation involves:
 - Establishing a goal
 - Forming the intention
 - Creating the plan (i.e. a sequence of actions)
 - Executing the plan
- Observation involves:
 - Perceiving system state
 - Interpreting state
 - Evaluating state wrt goal/intention



Designing Custom Controls

- Theoretical models are useful
 - What are the pieces we need to worry about in our design?
- Physical models (of the control) are also useful
 - Walk throughs with potential users
 - What can the user do? How will they do it? Any other actions that need support?
 - Address problems with gulf of execution
 - How does the control display changes in state? How will users understand those changes?
 - Address problems with gulf of evaluation
- BUT user is only ½ the story

3-Feb Announcements

- A01 marked
 - most marks are available but... jfalbanese to nmemond
 - demos at end of class
- Office hours
 - by appointment

Developer's Perspective

- Goal is to define a self-contained component that does one task well
- Developers should easily understand how to incorporate it into their project
 - Correct mental model of how it works
- Component should be easy to use
- Component should be customizable
- Component architecture should suggest its uses and how it can be extended so other designers can reuse it



Continuum of Complexity

- Developers primary concept of a control is based on control's complexity
- Complexity is a function of both View and Model
 - View – painting.
 - Model – representing data.
- Custom Components can range from the simple
 - A new style of button, for example
- To the complex
 - JTable



Simple Custom Components

- Recall behaviour of a typical button
 - mouseDown on button?
 - mouseUp on button?
- Demo simple custom component:
 - OnPressButton
 - TestButton



Some Notes

- Note that “addActionListener” methods are not included in JComponent
- Create your own
- Need to understand the EventListenerList listenerList attribute of JComponent
 - In Java, a Vector with paired entries
 - First the class of the listener, then the listener itself
- fireActionPerformed method also not present
 - Implement so that it parses the listenerList firing all actionListeners



Complex Components

- Separation of concerns
 - MVC split
 - Functionality that can change should be factored out, delegated to separate classes
 - Create loose coupling between component and other parts of interface
 - Design patterns help partition responsibilities, separate concerns
 - Observer, command, strategy, factory
 - Also provide a common language to increase understanding between developers



The View

- In Java, generally work within the javax.swing package rather than java.awt
- Lightweight components just make more sense
 - You are implementing a lightweight component
- Would probably use a Canvas object if implementing heavyweight custom component



The View

- Typical strategy is to derive a class from JPanel or JComponent
 - Similar, but they imply different uses
 - JComponent is a “thing”
 - JPanel is a container, or a collection of “things”
- Override paintComponent(Graphics g) to do display the custom view.
 - Much of the task is similar to XWindows programming
 - Graphics object, ability to paint and draw strings, etc.
 - g is actually a (more capable) subclass of Graphics, Graphics2D



The View

- Implement and attach listeners
 - The “controller” part of MVC
 - Coordinates the view and model given user input
- “Hide” interaction listeners from public interface
 - E.g. in OnPressButton, the MouseAdapter is an interaction listener
 - Just modifies view based on mousing events, no interaction with model



The Model

- Reuse existing models if they make sense
 - If you are creating a new renderer for a list, use the ListModel rather than creating your own model
 - Same with table or tree
 - AbstractModels exist and can be incorporated easily by users
 - Remember users are UI builders
- If you need to construct a model, make it an interface
 - Allows UI builders to build an adapter for their data model using your interface



The Model

- Provide a listener interface to notify others when the model changes
 - Reuse existing listener interfaces where appropriate
 - `PropertyChangeListener / PropertyChangeEvent` is a very flexible mechanism for this
- Consider granularity of listener updates
- Model should be completely independent from any GUI code
 - Should be able to test it by itself
- Consider Java models to understand how to build



ListModel versus TableModel

Four methods in `ListModel`

- `addListDataListener (ListDataListener l)`
Listener added (notified each change to data model).
- `removeListDataListener (ListDataListener l)`
Removes listener
- `getElementAt(int index)`
Returns the value at the specified index.
- `getSize()`
Returns the length of the list.

Nine methods in `TableModel`

- `addTableModelListener`
- `removeTableModelListener`
Similar
- `getValueAt(int rowIndex, int columnIndex)`
Returns the value for the cell at `columnIndex` and `rowIndex`.
- `getColumnCount()`
- `getRowCount()`
Similar again
- `getColumnName(int columnIndex)`
- `getColumnClass(int columnIndex)`
- `setValueAt(Object aValue, int rowIndex, int columnIndex)`
- `isCellEditable(int rowIndex, int columnIndex)`



Convenience Classes

- Good idea to add basic convenience classes
- Java's `DefaultListModel`, `DefaultTableModel`, `DefaultTreeModel`
 - `DefaultListModel` is a wrapper for a `Vector`
 - `DefaultTableModel` is a wrapper for a `Vector` of `Vectors`
 - `DefaultTreeModel` creates a basic tree of `TreeNode`s (another interface)
 - But also includes a `DefaultMutableTreeNode`
 - Convenience class to allow you to do basic implementation



Integrating Model and View

- Provide methods to get and set model of the custom component
- In setModel method:
 - Unregister listeners from old model
 - Register listeners with new model
 - Repaint the view to present the new model
- Much of this is automatic with JTable and DefaultTableModel
 - Benefit of convenience classes.
- You may need to handle some of this with your own models



Designing Custom Controls for Other Developers

- BE CONSISTENT
- It should be obvious that Java has a pattern for custom controls
- FOLLOW IT if you are building a control in Java



Designing Controls

- Users need to understand how control works
- Developers need to understand how control can be used
- Designers of controls need to consider these two “ideas” or models when building a new control
- The goal is to align models
 - Designer – Developer
 - Designer – End User



UI Design

- Custom controls are a special case
 - Complex design task because two different users
- More generally, you are building a user interface using standard components for an end user
- Consider overall design issues from a user vs developer/designer perspective

Component Design

- User should be able to:
 - Recognize what to do when faced with a component and what effect component will have on data
- Component should:
 - Be consistent with rest of interface and its components
 - Afford experimentation
- Think about user-computer dialog, the models of interaction
- Primary goal is to align user's with designer's mental model of control