Custom Widgets

Overview

- Why?
- User's Perspective
- Developer's Perspective

Why Design Custom Widgets?

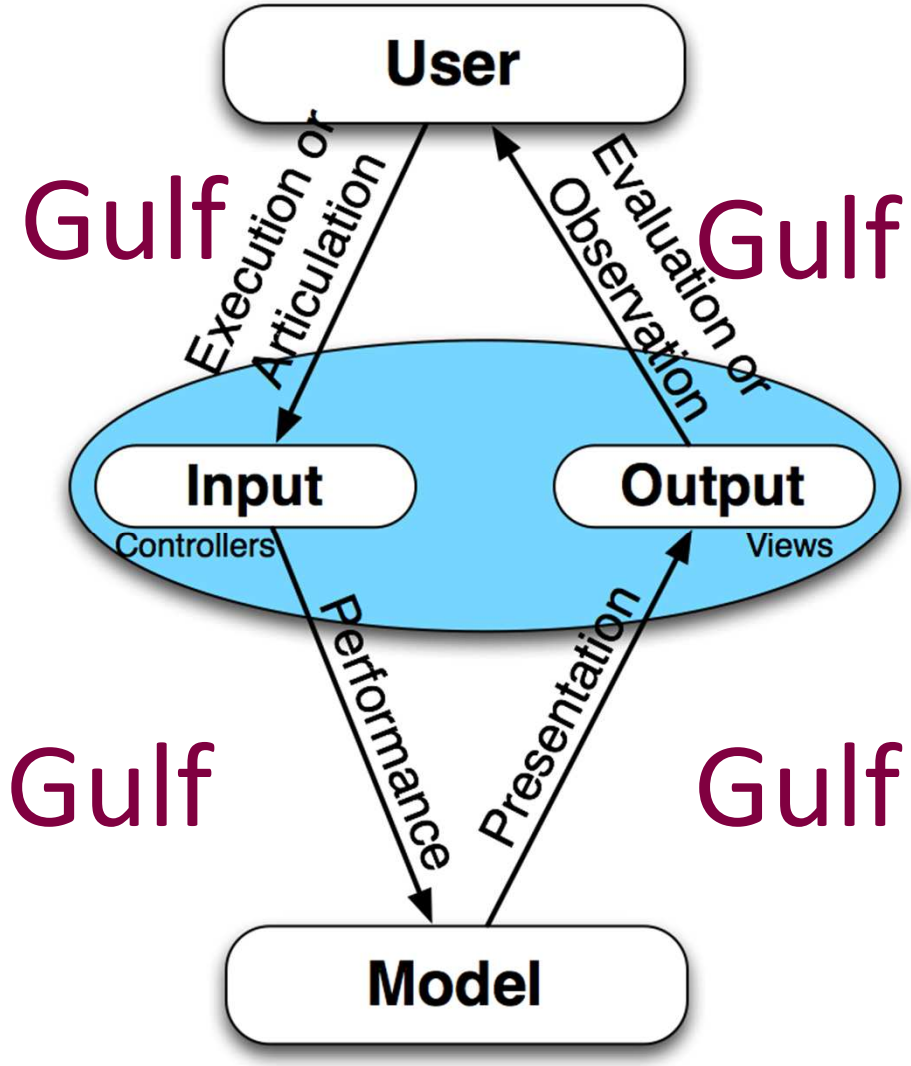
- You design new widgets to address a need (not addressed by existing widgets)
- Many examples of useless things
  - eg: the Segway
  - Make sure you need a custom control before you design one

Custom Widgets

- Custom widgets should:
  - Address a specific, identified 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 are "users" of the widget
  - Remember our interaction model...

Interaction Model; Gulfs



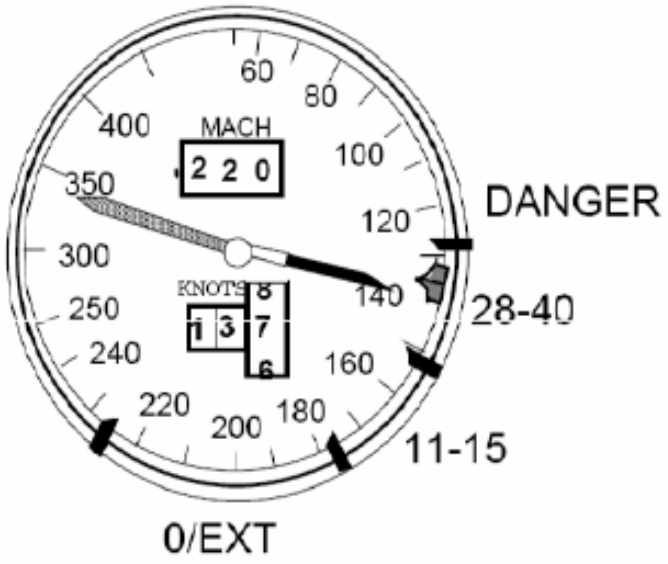
User's Perspective

- 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

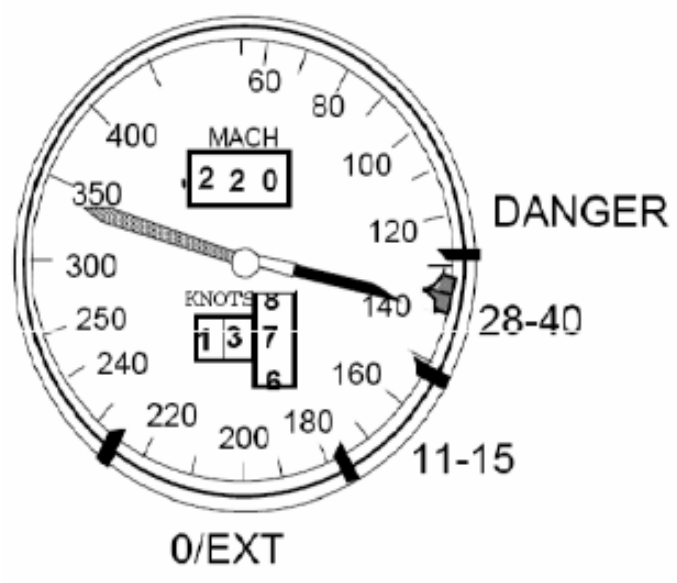
- As cockpits went "digital," analog controls were replaced with digital controls
- However, soon they began to see problems
- Why?



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 early digital airspeed indicators



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

Example: 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 and interviews help us uncover user's real needs and motivations
- Theoretical models are useful...
  - What are the gulfs that we need to attend to?

Designing for Users

- Physical models (prototypes of the widget) are useful...
  - Do 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
- Designing for users is a major focus of CS 489

- But designing for users is only 1/2 the story!

Developer's Perspective

- Goal is to define a self-contained widget that does one task well
- Developers should easily understand how to incorporate it into their project
  - Correct mental model of how it works
- Widget should be easy to to use
- Widget should be customizable
- Widget 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 widget's is based on its complexity
- Complexity is a function of both View and Model
  - View – painting.
  - Model – representing data.
- Custom widgets can range from the simple
  - A new style of button, for example
- To the complex
  - JTable

Simple Custom Widgets

- Recall behaviour of a typical button
  - mouseDown on button?
  - mouseUp on button?
- Demo simple custom Widget:
  - 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 Widgets

- Separation of concerns
  - MVC (within the widget!)
    - Functionality that can change should be factored out, delegated to separate classes
    - Create loose coupling between widget and other parts of interface
  - Design patterns help partition responsibilities, separate concerns
    - Observer, command, strategy, factory
    - Design patterns 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
- Examine Java models for inspiration

ListModel; 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()

getRownCount()

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 TreeNodes (another interface)
    - But also includes a DefaultMutableTreeNode
    - Convenience class to allow you to do basic implementation easily

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 for Other Developers

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

Summary: Two Users and a Thing

- End User:
  - Needs to understand what to do when faced with a widget and what effect widget will have on data
  - Has a mental model of how the widget should be used
- Developer:
  - Needs to understand how the widget can be used and how to integrate it with other UI elements
  - Has a mental model of how the widget should be used
- Widget:
  - Should afford experimentation
  - Should be consistent with the rest of the UI and its widgets