Model View Controller

Prof. James A. Landay
University of Washington
Winter 2007

January 12, 2007

Hall of Fame or Shame?

- eToys.com

Hall of Fame!

- Positives
  - clear navigation w/ top tabs & recs. on left
  - inverse pyramid style
  - clear value prop.
  - simple name & URL
  - privacy policy
- Negatives
  - non-links in blue

Outline

- Review
- Definitions of MVC
- Why do we need it?
- Changing the display
- Event flow
- Dragging at interactive speeds
- Adminstrivia
  - review this homework

Review Action Analysis & Automated Usability Testing

- GOMS
  - provides info about important UI properties
  - doesn't tell you everything you want to know about UI
  - ignores readability, memorizability of icons/commands, etc.
  - only gives performance for expert, error-free behavior
- hard to create model, but still easier than user testing
- changing later is much less work than initial generation
- Automated usability: pros & cons
  - faster than traditional techniques
  - can involve more participants → convincing data
  - ecological validity
  - easier to do comparisons across sites
  - tradeoff with losing observational data
**Model-View-Controller**

- Architecture for interactive apps
  - introduced by Smalltalk developers at PARC
- Partitions application so that it is
  - scalable
  - maintainable

---

**Example Application**

- Purple circles: 4
- Red squares: 2

---

**Model**

- Information application is manipulating
- Data representation of “real” world objects
  - “circuit” for a CAD program
    - logic gates & connections between them
  - shapes in a drawing program
    - geometry & color

---

**View**

- Implements visual display of model
- May have multiple views
  - e.g., circuit view & netlist view (wires & connections)
  - one of key advantages of MVC
  - When model is changed notify views
    - so view can change later (e.g., adding new item)

---

**Multiple Views**

- Purple circles: 4
- Red squares: 2

---

**View**

- Shape view
- Numerical View

---

**View**

- Implements visual display of model
- May have multiple views
  - e.g., shape view & numerical view
  - One of key advantages of MVC
  - When model is changed notify views
    - so view can change later (e.g., adding new item)
Controller

- Receiving all input events from the user
- Deciding what they mean and what to do

Model

View

Controller

\[ \text{Purple circles: 0} \]
\[ \text{Red squares: 0} \]

Controller Communication

- Communicates with the view
  - Determines which objects are being manipulated
    - e.g., which object was selected with a mouse click
- Calls model methods to make changes
  - Model makes the change and notifies the view to update

Relationship of View & Controller

“Pattern of behavior in response to user events (controller issues) is independent of visual geometry (view issues)”

$\rightarrow$ Controller must contact the view to interpret what user events mean (e.g., selection)

Combining View & Controller

- View and controller tightly intertwined
  - Lots of communication between the two
- Almost always occur in pairs
  - i.e., for each view, need a separate controller
- Many architectures combine in 1 class
Why MVC?

- Combining MVC into one class or using global variables will **not** scale. Why?
  - model may have more than one view
    - each different & needing update on model changes
- Separation eases maintenance. Why?
  - easy to add a new view later
    - may need new model info, but old views still work
  - can change a view later
    - e.g., draw shapes in 3-d
      - recall that the view handles selection

Adding Views Later

Purple circles: 4
Red squares: 2

Changing the Display

- How do we redraw when shape moves?

How Can We Move The Red Square?

Purple circles: 4
Red squares: 2

Ideas...?

Erase w/ Background Color & Redraw

Purple circles: 4
Red squares: 2

Works fine in this case

Moving Red Square

Purple circles: 4
Red squares: 2
**Changing the Display**

- **Erase & redraw fails**
  - using background color to erase leaves holes
  - drawing shape in new position loses ordering
  - how do we do it correctly?
- **Move in model & then redraw view**
  - change position of shapes in model
  - model keeps shapes in a desired order
  - tell all views to redraw themselves in order
  - drawbacks?
    - slow for large / complex drawings → flashing
    - solution?

**Damage / Redraw Method**

- **View informs windowing system of damage**
  - areas that need to be updated
  - view does not redraw them at this time
- **Windowing system**
  - batches updates
  - clips them to visible portions of window
- **Next time waiting for input**
  - windowing system calls Redraw method for window
    - passes region that needs to be updated

**Damage old, Change position in model, Damage new, Redraw**

**Event Flow**

- **Creating a new shape**
Event Flow (cont.)

- Assume purple circle selected

- Press mouse over tentative position
- Window sys. identifies proper window for event
- Drawing area controller gets mouse click event
- Checks mode & sees “circle”
- Calls model’s AddCircle method with position

Event Flow (cont.)

- AddCircle adds new obj. to model’s list of objs
- Model then notifies list of views of change
  - drawing area view
  - text summary view

- Views return to model
- Model returns to controller
- Controller returns to event handler

Event Flow (cont.)

- Views notify windowing system of damage
  - both views notify WS without making changes yet!

- Event handler notices damage requests pending & responds
  - if one of the views was obscured, it would be ignored

Event Flow (cont.)

- Event handler calls view’s Redraw methods with damaged area
- Views redraw all objects in model that are in damaged area

Dragging at Interactive Speeds

- Obvious method
  - damage old, move a few pixels, damage new
- Problems?
  - may be too slow
  - how fast must we do that to look smooth?
    - must take less than 200 ms
- Solutions?
  - don’t draw object, draw an outline (or dashed box)
    - use XOR to erase quickly
    - complement (problems w/ color)
  - save portion of frame buffer before dragging
    - restore afterwards to erase
  - less of an issue these days with FAST hardware
Summary

- The three parts of MVC are
  - model, which is?
    - underlying information that application is manipulating
  - view, which is?
    - visual display of the model information
  - controller, which does?
    - handles interaction with the user & decides what to do
- Which two pieces are often combined?
  - view & controller – they always occur in pairs anyways
- MVC partitions an application so that it is
  - scalable – may have multiple views for a single model
  - maintainable – add or change views later

Administrivia

- Look at project homework due Mon at 11 PM
- Questions?

Next Time

- Studio Task #1 – Gestalt Principles