Lecture 3: UI Software Architecture

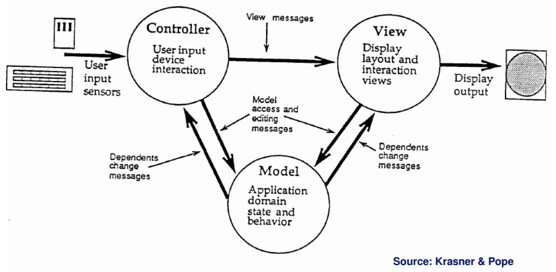
Today's Topics

- Model-view-controller
- View hierarchy
- Observer

Model-View-Controller Pattern

- Separation of responsibilities
  - Model: application state
    - Maintains application state (data fields)
    - Implements state-changing behavior
    - Notifies dependent views/controllers when changes occur (observer pattern)
  - View: output
    - Occupies screen extent (position, size)
    - Draws on the screen
    - Listens for changes to the model
    - Queries the model to draw it
  - Controller: input
    - Listens for keyboard & mouse events
    - Tells the model or the view to change accordingly
- Decoupling
  - Can have multiple views/controllers for same model
  - Can reuse views/controllers for other models

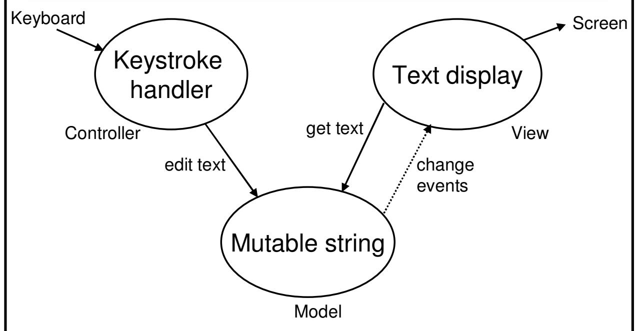
MVC Diagram



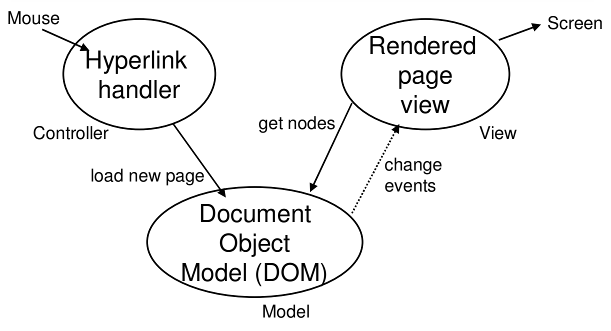
Controller
User input device interaction
View messages
View
Display layout and interaction views
Display output
User input sensors
Model access and editing messages
Dependents change messages
Dependents change messages
Model
Application domain state and behavior

Source: Krasner & Pope

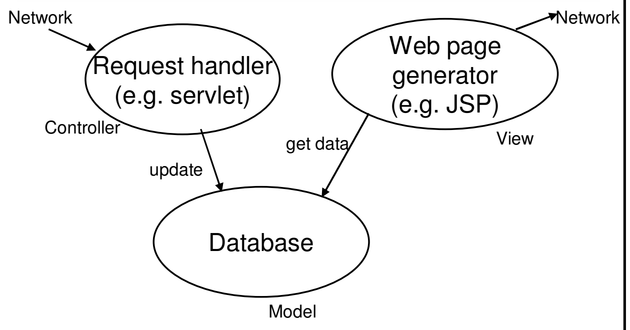
Example: Text Field



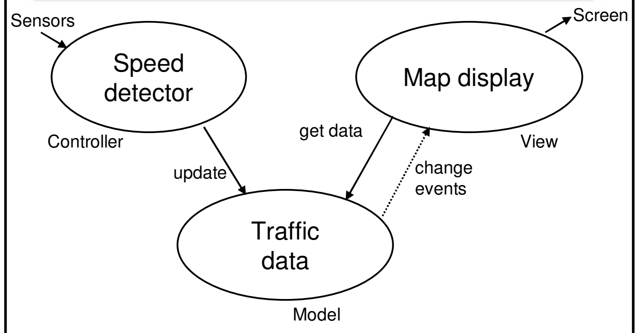
Example: Web Browser



Example: Database-Backed Web Server



Example: Traffic Visualizer



Model Granularity

- How fine-grained are the observable parts of the model?
  - getText() vs. getPartOfText(start, end)
- How fine-grained are the change descriptions (events)?
  - "The string has changed somehow" vs. "Insertion between offsets 3 and 5"
- How fine-grained are event registrations (the events the listener actually sees)?
  - "Tell me about every change" vs. "Tell me about changes between offsets 3 and 5"

Hard to Separate Controller and View

- Controller often needs output
  - View must provide **affordances** for controller (e.g. scrollbar thumb)
  - View must also provide **feedback** about controller state (e.g., depressed button)
- State shared between controller and view: Who manages the selection?
  - Must be displayed by the view (as blinking text cursor or highlight)
  - Must be updated and used by the controller
  - Should selection be in model?
    - Generally not
    - Some views need independent selections (e.g. two windows on the same document)
    - Other views need synchronized selections (e.g. table view & chart view)

Reality: Tightly Coupled View & Controller

- MVC has largely been superseded by MV (Model-View)
- A reusable view manages both output and input
  - Also called widget or component
- Examples: scrollbar, button, menubar

View Hierarchy

- Views are arranged into a hierarchy
- Containers
  - Window, panel, rich text widget
- Components
  - Canvas, button, label, textbox
  - Containers are also components
- Every GUI system has a view hierarchy, and the hierarchy is used in lots of ways
  - Output
  - Input
  - Layout

View Hierarchy: Output

- Drawing
  - Draw requests are passed top-down through the hierarchy
- Clipping
  - Parent container prevents its child components from drawing outside its extent
- Z-order
  - Children are (usually) drawn on top of parents
  - Child order dictates drawing order between siblings
- Coordinate system
  - Every container has its own coordinate system (origin usually at the top left)
  - Child positions are expressed in terms of parent coordinates

View Hierarchy: Input

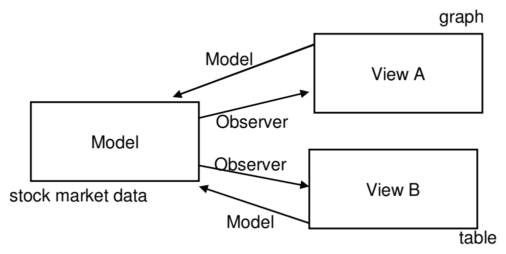
- Event dispatch and propagation
  - Raw input events (key presses, mouse movements, mouse clicks) are sent to lowest component
  - Event propagates up the hierarchy until some component handles it
- Keyboard focus
  - One component in the hierarchy has the focus (implicitly, its ancestors do too)

View Hierarchy: Layout

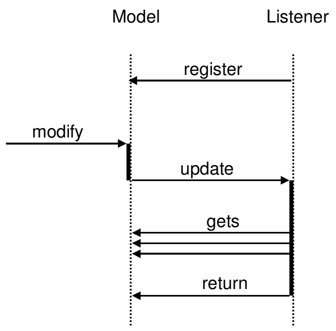
- Automatic layout: children are positioned and sized within parent
  - Allows window resizing
  - Smoothly deals with internationalization and platform differences (e.g. fonts or widget sizes)
  - Lifts burden of maintaining sizes and positions from the programmer
    - Although actually just raises the level of abstraction, because you still want to get the graphic design (alignment & spacing) right

Observer Pattern

- Observer pattern is used to decouple model from views



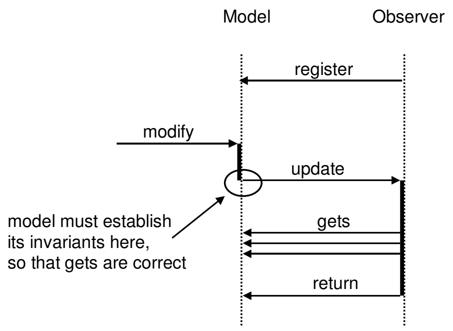
Basic Interaction



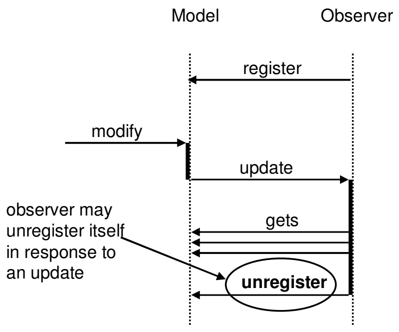
```
interface Model {
    void register(Observer)
    void unregister(Observer)
    Object get()
    void modify()
}

interface Observer {
    void update(Event)
}
```

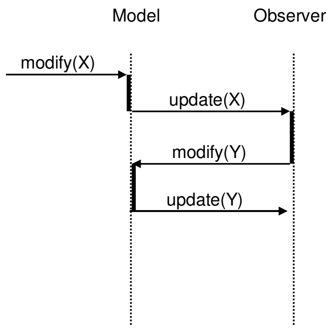
Model Must Be Consistent Before Update

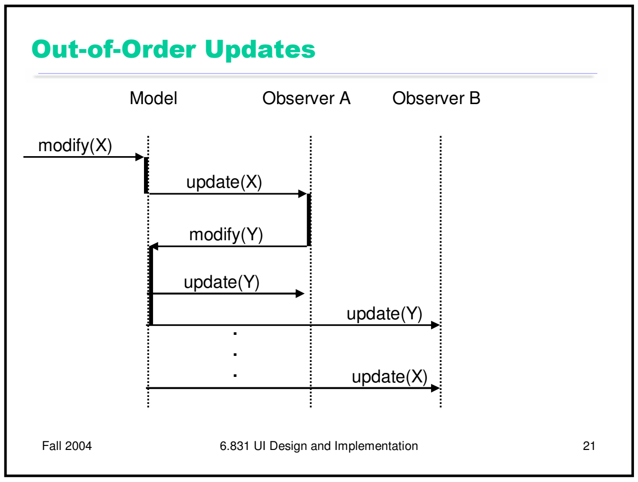


Registration Changes During Update

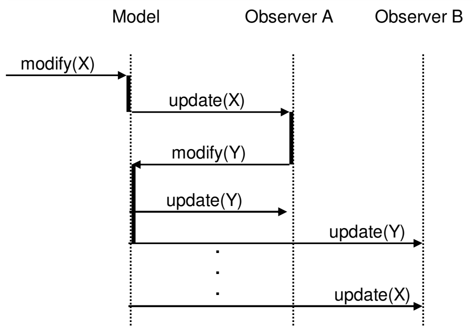


Update Triggers A Modify





Out-of-Order Updates



Fall 2004 6.831 UI Design and Implementation 21