Interactive Systems and MVC Pattern

Dr. Isaac Griffith Idaho State University

Outcomes

After today's lecture you will be able to:

- Understand the basic idea of an architectural pattern
- Understand the structure and operation of the Model-View-Controller Pattern
- Understand the shift in design thinking when working with a framework such as MVC

Introduction

- Our general approach used so far has worked well for the examples and simple systems we have encountered.
- But, for larger systems this approach seldom leads to efficient designs
  - Instead, we should rely upon the experience of other engineers
  - To handle more complex problems we utilize this experience in the form of frameworks/structures in which we operate
- For Software Systems, these structures are provided by choosing a **Software Architecture**
- Specifically, this week we will be working with the **Model-View-Controller (MVC) architectural pattern**

Model-View-Controller (MVC)

CS 2263

Model-View-Controller

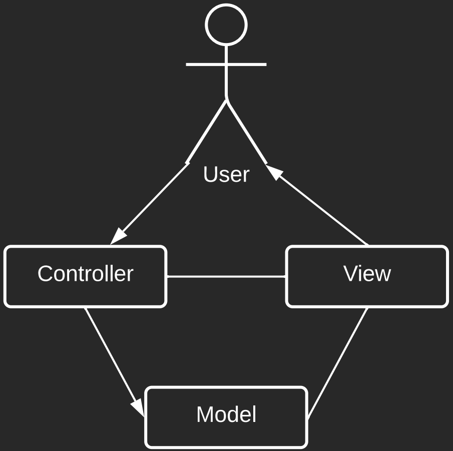
- MVC is an older pattern, first identified in the Smalltalk language
- It divides the system into three subsystems:
  - **Model:** Application or object data
    - Any object may play this role
    - Tend to be passive data storage
  - **View:** Rendering of the data for the user
    - Renders the data into a specific format (suitable for interaction)
    - Updates the rendering as the model changes
  - **Controller:** Means by which the user manipulates the data
    - captures user input
    - issues method calls on the model to modify stored data

Model-View-Controller

- The pattern has several advantages:
  - produces highly cohesive modules with low coupling
  - facilitates greater flexibility and reuse
  - supports multiple presentations of the same information

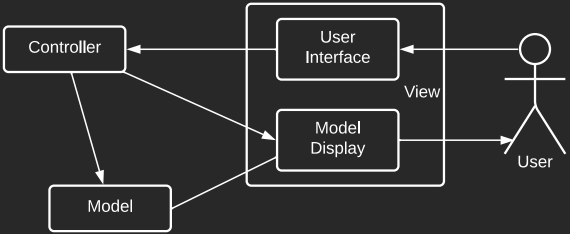
Model-View-Controller

- Typical operation:
  - Model only changes when user input causes the controller to inform the model of the changes
  - The view then must be notified of the model changes
  - The controller is provided with variables for both the model and view to which it is connected
  - The view communicates with the model, thus it has an instance of the model
  - The controller and view communicate to the user via the UI
    - UI components are used by both
    - The view is not the UI



Model-View-Controller

- For our purposes we will refer to the common subsystem which contains **UI components** and the **model display** as the "View"
  - Responsible for all the Look and Feel issues
  - User input may induce a change in the model (via the controller) or the view, or both
    - Ex: user deletes or adds text, the controller captures the changes and notifies the model
    - The view then refreshes its display
    - Thus both the view and model were updated by user input



Model View Controller

- More than one view-controller pair may be associated with a given model
- Models may be changed by other mechanisms external to the controller
  - The view still needs to be updated
- View-Model relationship is that of Subject-Observer
  - model will automatically notify all its associated views of changes
  - **Guiding Principle:** each view is a faithful rendering of the model

Examples

- Library System
  - Model will include information about books
  - A view would allow a user to view number of copies, holds on books, and to place a hold
  - Simultaneously, a staff member may add copies of a book
  - Both views are accessing the same information from the book model
- Graphs
  - Model: Data set of (x,y) pairs
  - Views: bar graphs, line graphs, pie charts, etc.
    - changing format allows changing the view
  - Controller: add/remove data

Implementation

- For any architecture, we must have a clear understanding of how the responsibilities are shared between subsystems.
- In the MVC the roles are clearly defined as follows:
  - The **view** is responsible for all presentation issues
  - The **model** holds the application object
  - The **controller** takes care of the response strategy

Model Definition

```java
public class Model extends Observable {
  // code

  public void changeData() {
    // code to update data
    setChanged();
    notifyObservers(changeInfo);
  }
}
```

View Definition

```java
public class View implements Observer {
  // code

  public void update(Observable model, Object data) {
    // refresh view using data
  }
}
```

Note: Due to the deprecation of both `Observable` and `Observer` in Java 9+, I would suggest implementing your own implementations of these components rather than relying upon those provided in the JDK

Implementation

- As we develop features, we start with the detailed list of specifications
    - We need them to be detailed enough so that we can classify them according to which category they fall into: Model, View, Controller
- Often there is entanglement making it difficult to separate out the different components
    - Separating the view from the data is tricky if there is a close relationship between the data and methods for rendering it
    - Separating which components belong to the controller and which belong to the view
- The latter issue can be addressed by creating a UI with functionality to serve both the view and controller
    - Display components are available to the view
    - Input capture components are available to pass data to the controller

Benefits of MVC

- **Cohesive Modules:** We separate code (display and data) to form more cohesive modules
- **Flexibility:** Model is unaware of the exact nature of the view or controller, adding flexibility
- **Low Coupling:** Design modularity improves chances of the components to be swapped in and out as the programmer or user desires.
  - Promotes parallel development, easier debugging, and easier maintenance
- **Adaptable Modules:** Allows components to be changed with little interference to the rest of the system
- **Distributed Systems:** due to the separation of the modules, the systems could be physically separated

Drawing Program

CS 2263

A Simple Drawing Program

- We will apply MVC by designing a simple program that creates and labels figures
    - This demonstrates how to design with an architecture in mind
    - This allows us to understand how the MVC architecture is employed

The Requirements

Initial Requirements

- Draw lines and circles
- Place labels at various points on the figure
  - labels are strings
  - font and font size will be selected by a separate command
- Save the complete figure to a file
- Open a file to edit it
- Backtrack our drawing process by undoing recent operations

‘Look and Feel’ Requirements

- A Simple frame containing:
  - display panel on which the figure is displayed
    - Listen to mouse-clicks employed by user to specify points
  - command panel containing the following buttons
    - `Draw Line`
    - `Draw Circle`
    - `Add Label`
    - etc.
- Display Panel will have
  - cross-hair cursor for specifying points while drawing
  - a _ for showing the character insertion point for labels
  - default cursor is an arrow

‘Look and Feel’ Requirements

- Drawing
  - Lines require user to specify end points with mouse clicks
  - Circles require the user to specify two diametrically opposite points on the perimeter
    - center will be marked with a black square for reference
  - Labels require user to specify starting point with mouse click

Drawing a Line

Actions performed by the actor	Responses from the system
1. The user clicks on the `Draw Line` button in the command panel	
	2. The system changes the cursor to a cross-hair
3. The user clicks first on one end point and then on the other end point of the line to be drawn	
	4. The system adds a line segment with the two specified end points to the figure being create. The cursor changes to the default

Adding a Label

Actions performed by the actor	Responses from the system
1. The user clicks on the `Add Label` button in the command panel	
	2. The system changes the cursor to a cross-hair cursor
3. The user clicks at the left end point of the intended label	
	4. The system places a _ at the clicked location
5. The user types a character or clicks the mouse at another location	
	6. If the character is not a carriage return the system displays the typed character followed by a _, and the user continues with Step 5; in case of a mouse-click, it goes to Step 4; otherwise it goes to the default state

Change Font

Actions performed by the actor	Responses from the system
1. The user clicks on the `Change Font` button in the command panel	
	2. The system displays a list of all the fonts available
3. The user clicks on the desired font	
	4. The system changes to the specified font and displays a message to that effect

Select an Item

Action performed by the actor	Responses from the system
1. The user clicks on the `Select` button in the command panel	
	2. The system changes the display to the `selection mode`
3. The user clicks the mouse on the drawing	
	4. If the click falls on an item, the system adds the item to its collection of selected items and updates the display to reflect the addition. The systems returns the display to the default mode

System Design

CS 2263

ROAR

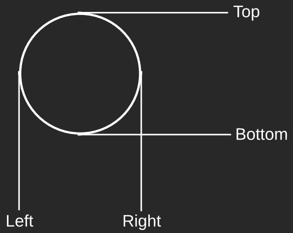
Designing the System

- The process for designing a system when we have selected a base architecture is slightly different
- The difference stems from the fact that we have a set of subsystems within which we need to work
- The key is to define how the roles of these subsystems fit within the context of our problem
- The first step is then to evaluate our use cases and determine how to allocate the system responsibilities
- Then we look into designing each subsystem

Defining the Model

For the drawing program our model is a collection of three items:

- **Lines**
  - represented by the two endpoints
- **Circles**
  - represented by:
    - the X-coordinates of the leftmost and rightmost points
    - the Y-coordinates of the topmost and bottommost points
- **Labels**
  - represented by:
    - coordinate's starting position
    - the text
    - style and size of the characters

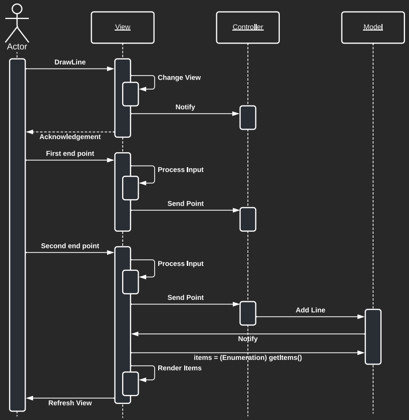


- The collection then is accessed by the view when it is to rendered
- The collection also requirements methods to access and modify it
  - `addItem(Item)`
  - `getItems()`

Defining the Controller

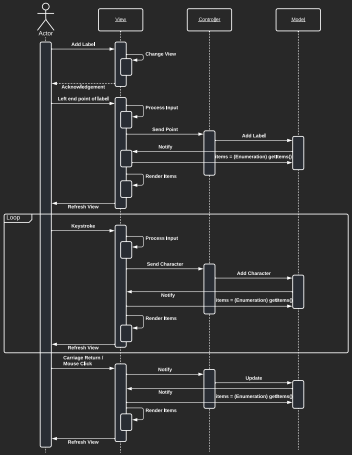
- The controller subsystem orchestrates the who show and thus its definition is critical
  - When the user executes an operation, input is received by the view
  - This information is then conveyed to the controller via public methods
  - The controller then communicates to model

Drawing a Line / Circle



Adding a circle is similar but with additional processing to map the input the the model.

Adding a Label



Selection and Deletion

- We are able to delete lines, circles, or labels by first selecting them and then invoking the `delete` operation
  - Lines are selected by clicking on an end point
  - Circles by clicking on the center
  - Labels by clicking the label
- This is implemented as follows:
  1. user give the command through a button click, then clicks to specify the item
     - this is detected in the view and communicated to the controller
  2. model must have a means to indicate an item is selected
     - view recognizing the change in selection updates the view (marks selected items a specific color)
  3. we find the appropriate item by looping through all components and detecting if the selected point is contained within them
  4. Model notifies the view, which renders all items appropriately
- Deletion adds the additional behavior that the delete command was invoked and the selected items are then removed from the collection and the view is re-rendered

Saving and Retrieving Drawings

The use cases for saving and retrieving data are simple

- The user requests a save/retrieve operation, the system asks for a file name which the user provides and the system completes the task.

This can be partitioned across the subsystems as follows:

1. View: receives the initial request from the user and prompts for a file name
2. View: invokes the appropriate method on the controller, passing the file name as a parameter
3. Controller: takes care of clean-up (i.e., unselects all items), then invokes appropriate method in the model
4. Model: serializes relevant objects to the file.

For Next Time

- Review Chapter 11.1 - 11.4
- Review this lecture
- Read Chapter 11.5 - 11.6
- Watch Lecture 31

Are there any questions?

ROAR