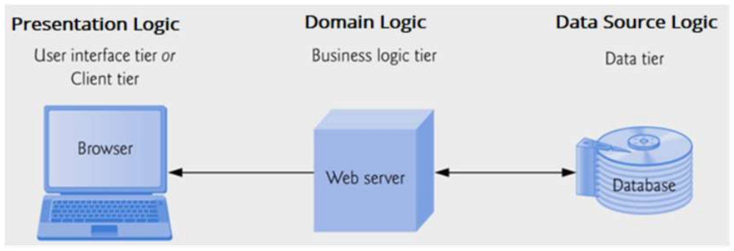
SOEN 387: Web-Based Enterprise Application Design

Chapter 4. Web Presentation

Content

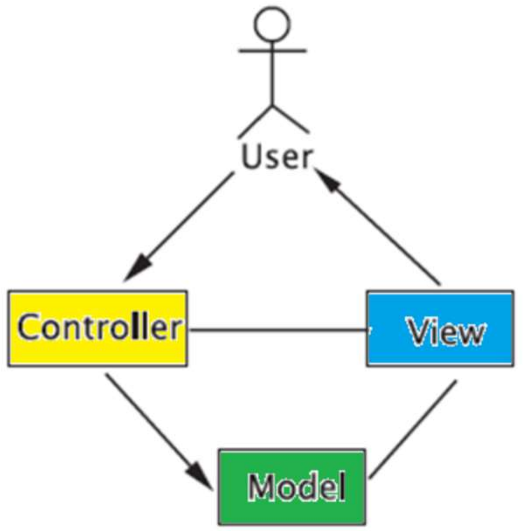
- Model-View-Controller
- MVC Example
- Implementation
- Benefits of the MVC pattern



Model-View-Controller

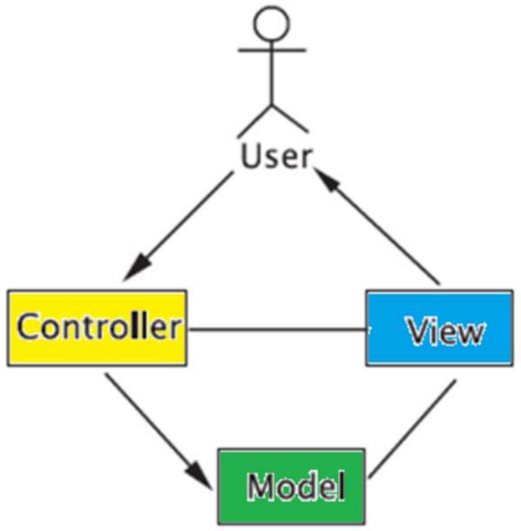
- The pattern divides the application into three subsystems: model, view, and controller.
- The model, which is a relatively passive object, stores the data.
- Any object can play the role of model. The view renders the model into a specified format, typically something that is suitable for interaction with the end user.

  For instance, if the model stores information about bank accounts, a certain view may display only the number of accounts and the total of the account balances.
- The controller captures user input and when necessary, issues method calls on the model to modify the stored data. When the model changes, the view responds by appropriately modifying the display.



The model–view–controller architecture

- The view must be notified when the model changes.
- Instance variables in the controller refer to the model and the view.
- The view must communicate with the model, so it has an instance variable that points to the model object. Both the controller and the view communicate with the user through the UI.
- This means that some components of the UI are used by the controller to receive input; others are used by the view to appropriately display the model and some can serve both purposes



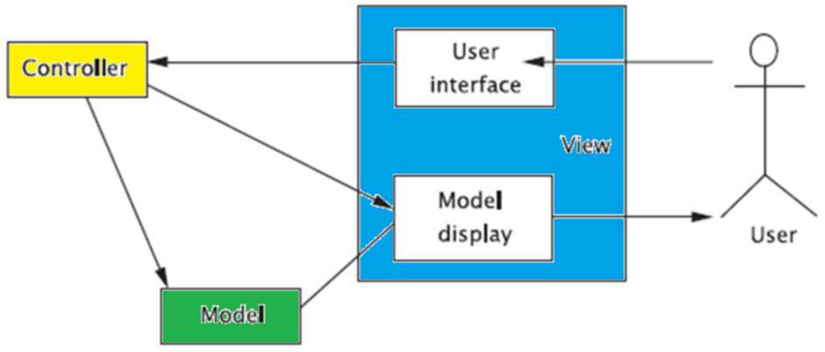
The model–view–controller architecture

Alternate view of the MVC architecture

- This user interface must include components that are part of the controller (e.g., buttons for giving commands).
- The view subsystem is therefore responsible for all the look and feel issues, whether they arise from a human–computer interaction perspective (e.g., kinds of buttons being used) or from issues relating to how we render the model.
- User-generated events may cause a controller to change the model, or view, or both.

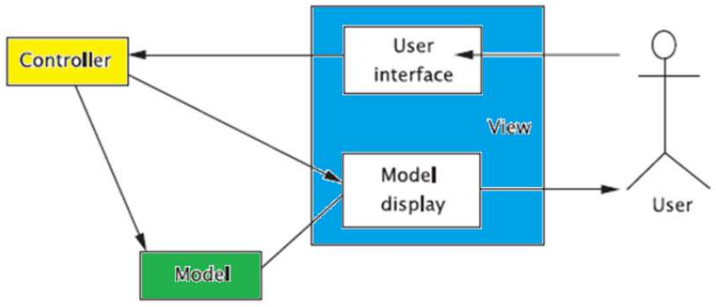
For example, suppose that the model stored the text that is being edited by the end-user. When the user deletes or adds text, the controller captures the changes and notifies the model. The view, which observes the model, then refreshes its display, with the result that the end-user sees the changes he/she made to the data. In this case, user-input caused a change to both the model and the view.

On the other hand, consider a user scrolling the data. Since no changes are made to the data itself, the model does not change and need not be notified. But the view now needs to display previously-hidden data, which makes it necessary for the view to contact the model and retrieve information.



An alternate view of the the MVC architecture

- More than one view–controller pair may be associated with a model.
- Whenever user input causes one of the controllers to notify changes to the model, all associated views are automatically updated.
- It could also be the case that the model is changed not via one of the controllers, but through some other mechanism. In this case, the model must notify all associated views of the changes.
- The view–model relationship is that of a subject–observer. The model, as the subject, maintains references to all of the views that are interested in observing it.
- Whenever an action that changes the model occurs, the model automatically notifies all of these views.
- The views then refresh their displays.



An alternate view of the the MVC architecture

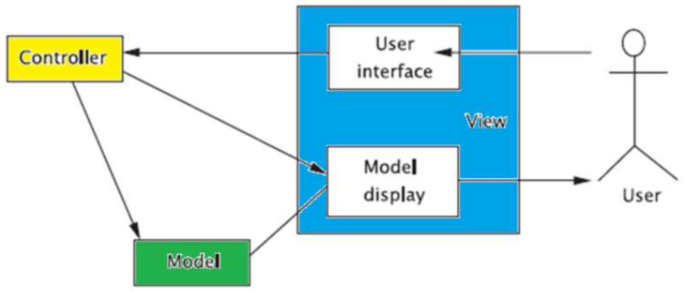
MVC Examples

- **Example 1**: Suppose that in the library system we have a GUI screen using which users can place holds on books. Another GUI screen allows a library staff member to add copies of books. Suppose that a user views the number of copies, number of holds on a book and is about to place a hold on the book. At the same time, a library staff member views the book record and adds a copy. Information from the same model (book) is now displayed in different formats in the two screens.

- **Example 2**: Suppose that we have a graph-plot of pairs of (x, y) values. The collection of data points constitutes the model. The graph-viewing software provides the user with several output formats — bar graphs, line graphs, pie charts, etc. When the user changes formats, the view changes without any change to the model.

Implementation

- The view is responsible for all the presentation issues.
- The model holds the application object
- The controller takes care of the response strategy



An alternate view of the the MVC architecture

The definition for the model will be as follows:

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

The definition for the observer will be as follows:

```
public class View implements Observer {
   // code
   public void update(Observable model, Object data)
   {
    // refresh view using data
   }
}
```

If a view is no longer interested in the model, it can be deleted from the list of observers. Since the controllers react to user input, they may send messages directly to the views asking them to refresh their displays.

Benefits of the MVC pattern

Cohesive modules: Instead of putting unrelated code (display and data) in the same module, we separate the functionality so that each module is cohesive.

Flexibility: The model is unaware of the exact nature of the view or controller it is working with. It is simply an observable. This adds flexibility.

Low coupling: Modularity of the design improves the chances that components can be swapped in and out as the user or programmer desires. This also promotes parallel development, easier debugging, and maintenance.

Adaptable modules: Components can be changed with less interference to the rest of the system.

Distributed systems: Since the modules are separated, it is possible that the three subsystems are geographically separated.