MVC/MVP Design Pattern

Hsing-Yu Chen (h492chen)
Neesarg Patel (n94patel)
Vigneshwaran Sendhil (vsendhil)

MVC

Purpose and Motivation

The purpose of MVC is to separate the user interface (UI) and the underlying data (business logic). This means each component does one thing, and can be changed without affecting the other components. It tackles the Separation of Concerns in computer science.

In real life, how the data is presented (UI) changes more frequently than the data itself (business logic). Thus, the motivation behind MVC is to let developers and designers work parallelly, so UI changes can be made without affecting the underlying data or logic.

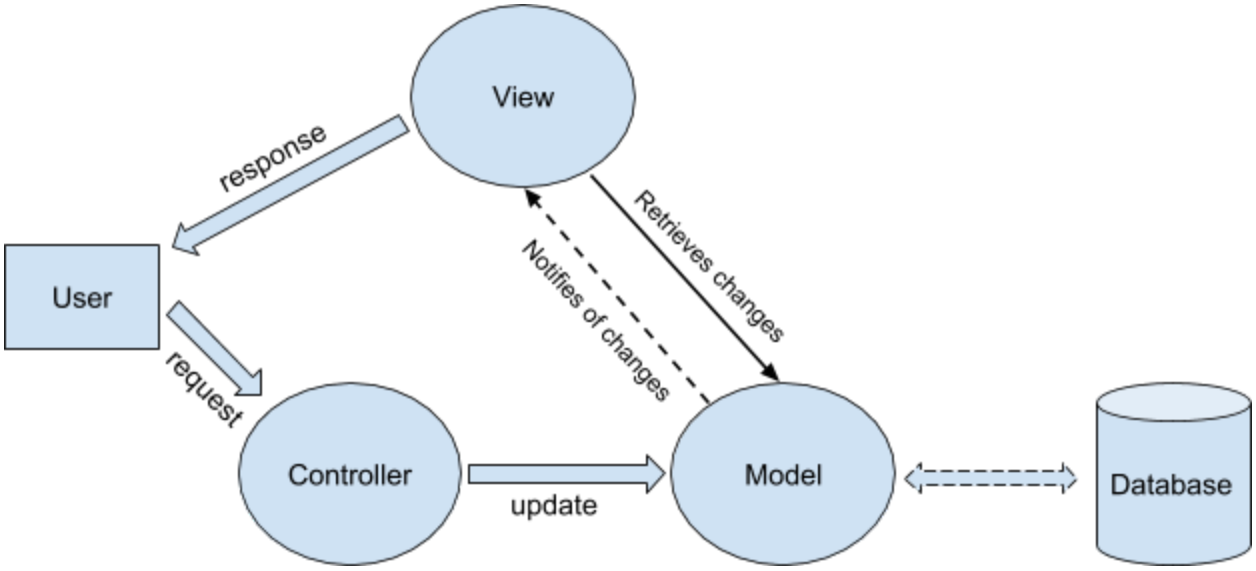
Intended Use Cases

- MVC can be used where parallel development is needed. That means different people can work simultaneously on different components without affecting or blocking others. A popular example is the division of front-end and back-end developers.
- MVC is useful when the same piece of data has to be presented to the user in multiple ways (e.g. pie chart and histogram) at the same time.
- MVC can be adapted, like in some cases the UI and logic can be combined.

Vocabulary

- Model - manages application data and its modification
- View - manages interface to present data
- Controller - manages interaction to modify data

Structure



- The user perceives View and send requests to Controller.
- Controller decide how to modify Model according to the requests.
- Model may interact with a database, and if it is modified, it will notify View(s) that there are some changes needed to be updated.
- View retrieves the changes and refreshes itself with the updated data
- The user perceives the updated view

Advantages

- Simultaneous or parallel development - Several people can work at the same time on the code.
- Model is domain independent.
- Code can be reused.
- Multiple views can be created based on one model.
- Code can be maintained and modified easily.
- MVC has loose coupling between models, controllers and views.
- High cohesion is also enabled in MVC between views and controllers.
- Testing can be done separately for the different components since testing UI is usually expensive.

Disadvantages

- Navigation of code can be complex since there are many different components and abstractions which require time to adapt to them.
- Consistency should be maintained throughout the code for navigation and readability. Otherwise, it causes scattering and difficult to read.
- Using MVC requires the knowledge of multiple technologies, like a web application can require the knowledge of HTML, PHP, SQL. So, the learning curve is high for this pattern compared to others.

NFPs supported

- Evolvability - MVC can easily be updated with new views, controllers or models, and that doesn't impact the old components much.
- Readability - When the code is properly implemented and consistent, navigating and understanding the code becomes easy.
- Adaptability - MVC can be adapted or modified to other patterns such as MVP and MVVM.

NFPs inhibited

- Complexity - MVC can become complex quickly with many models, controllers and views.

MVP

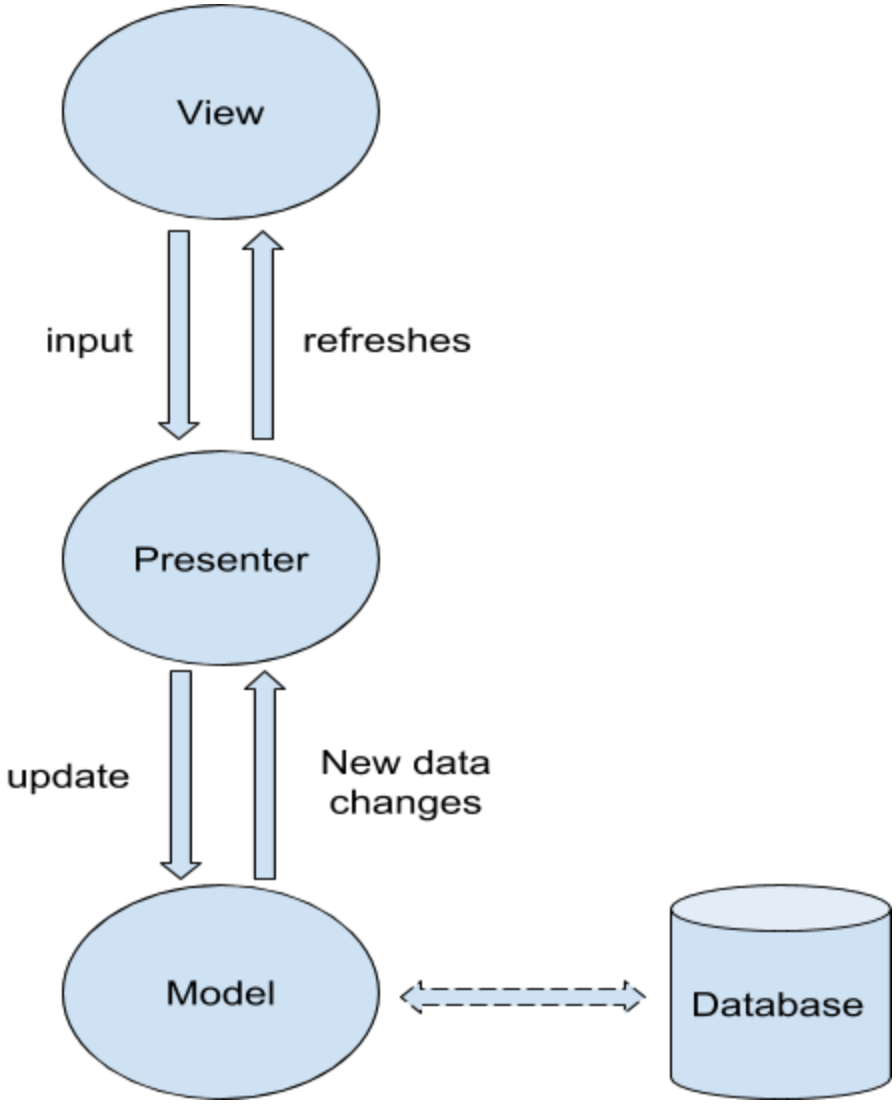
Purpose and Motivation

MVP is very similar to MVC but its purpose is improve the separation of concerns furthermore. It also improves testing.

Vocabulary

MVP has model and view similar to MVC. But instead of a controller, MVP has presenter. The presenter is like the middle-man, where it connects model and view. Model and view never communicate with each other, and all communication goes through the presenter.

Structure



- View sends the changes to presenter.
- The presenter has all logic, decides what actions to do with the inputs.
- Presenter updates the model.
- Model interacts with database if needed, and sends the new changes back to Presenter.
- Presenter formats and refreshes the view with the new changes.

Advantages

- Better decoupling than MVC
- More testing can be done
- Less complex than MVC
- Rest similar to MVC

Disadvantages

- All logic is in the presenter, can get messy.
- Rest similar to MVC

NFPs supported and inhibited

These are similar to MVC.

Demonstration

Scenario 1 - communication between a View, Model and Controller

Two friends are talking where one represents the view and another represents the controller. The View, being new in town, asks the Controller if he knows anyone who can show him around town. The controller then asks someone he knows (the model) if he wants a new friend to hang out with. Here, the “Model” is a play on the word model where models (aka supermodels) are highly desirable friends. The model then sends a text message to the view (representing the notification of state change). Then, the view

calls the model (representing the retrieval of the state change) and finds out that the model does not want to be friends with the view. The view's reaction to the state is to angrily say "life sucks".

Scenario 2 - can change View without affecting Model

Scenario 2 showcases MVC's ability to change the View without the need to change the Model. This time, the view calls the model and once again finds out that the model does not want to be friends with the view. Now, the view's reaction to the state is more positive so he says "OK lets stay positive, I should start working out". Note that the view has changed but the model has not.

Scenario 3 - can use a different Controller for a View

Scenario 3 showcases MVC's ability to have multiple controllers with a single view. The view has the option of contacting any of the two controllers. This is shown by the view first contacting one of the controllers requesting that he ask his friend the model to spend the weekend with him. This controller gives a poor recommendation to the model and thus the model rejects the view this time around. Then, the same view contacts another controller. This time, the controller is more enthusiastic in his recommendation of the view to the model. Thus, the model accepts the invitation this time around.

Scenario 4 - multiple Views can communicate with a Controller

Scenario 4 showcases MVC's ability to have multiple views with a single controller. This is shown by having two different characters represent two different views. Both views in this scenario contact the same controller. The controller then tells the model about his two friends and the model then decides which of the two views he wants to become friends with. Note that the Controller treats both views differently, in that he recommends one View more highly than the other when speaking with the Model.

Scenario 5 - shows advantage of MVP (Model, View, Presenter)

Scenario 5 showcases MVP's advantage of decoupling the view from the model using a presenter as the mediator. In this case, similar to the other scenarios, the view requests the presenter to ask the model if he would like to hang out with the view. This time, the model states he does not feel comfortable talking directly with the view and asks the presenter to relay the message back to the view for him. Thus, the model is decoupled from the view by having the presenter in between.