Presentation Patterns : MVC, MVC2, MVP

In-class Presentation

Comp. Sci. 9831B : Software Engineering for Cyber-Physical Systems

Instructor: Prof. Kostas Kontogiannis

Aasminpreet Singh Kainth

Department of Computer Science

Middlesex College, Western University

Agenda

Why do we need these patterns ?

One can certainly build software applications without using any of these patterns, but by using these patterns we can achieve **separation of concerns** design principle.

These help in improving maintainability of the application.

Implementing these patterns improve the testability of the application using automated unit tests.

Why do we need these patterns ?

Separation of Concerns.

If your components start doing more than one thing, code gets hard to read, things become more complicated than they need to be -- basically an app turns into a big plate of Spaghetti Code.

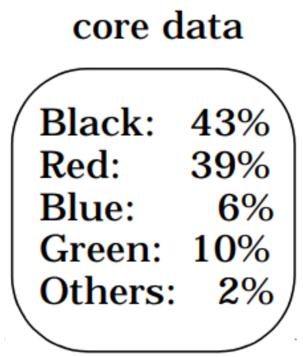
The *models* contain the business logic.

The *views* present the data to the user.

The *controllers* decide what to do with the various user actions.

Code is easy to read because things are as simple as possible

Example



core data

Black:	43%
Red:	39%
Blue:	6%
Green:	10%
Others:	2%

Problem User interfaces are especially prone to change requests.

- Extending the functionality of an application
- Specific user interface adaptation
- Different ‘look and feel’ standard.

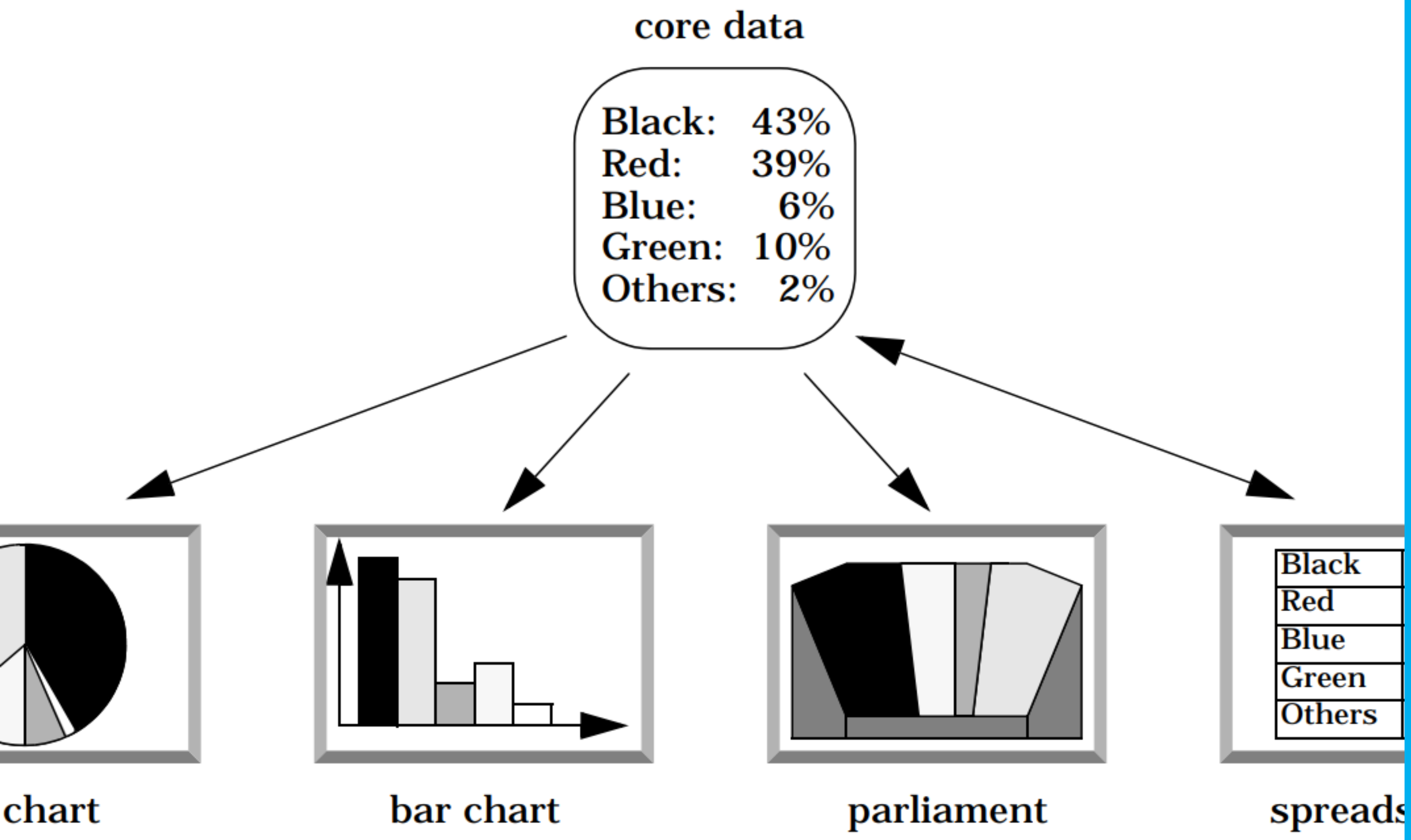
Upgrading to a new release of our windowing system can imply code changes

Different users place conflicting requirements on the user interface

Easily incorporation of several UI paradigms

Building a system with the required flexibility is expensive and error prone

Example



The following forces influence the solution:

Same information is presented differently

The display and behavior of the application must reflect data manipulations immediately

Changes to the UI should be easy

Supporting different ‘look and feel’ standards should not affect code

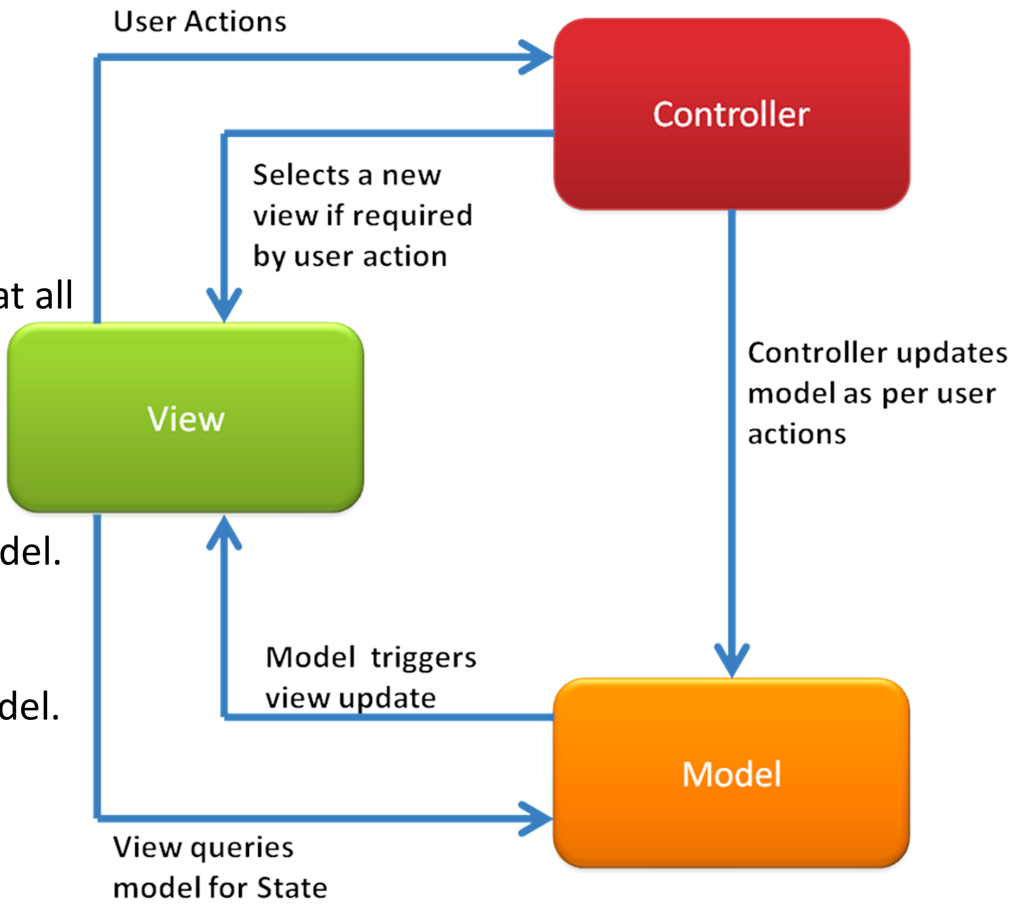
The Classic Model-View-Controller Pattern (MVC)

In MVC pattern, model, view, controller triad exists for each object that can be manipulated by the user.

Model : Storing the domain data and their basic logic
Has no reference to the other modules
Domain data does not depend on the presentation of domain data at all

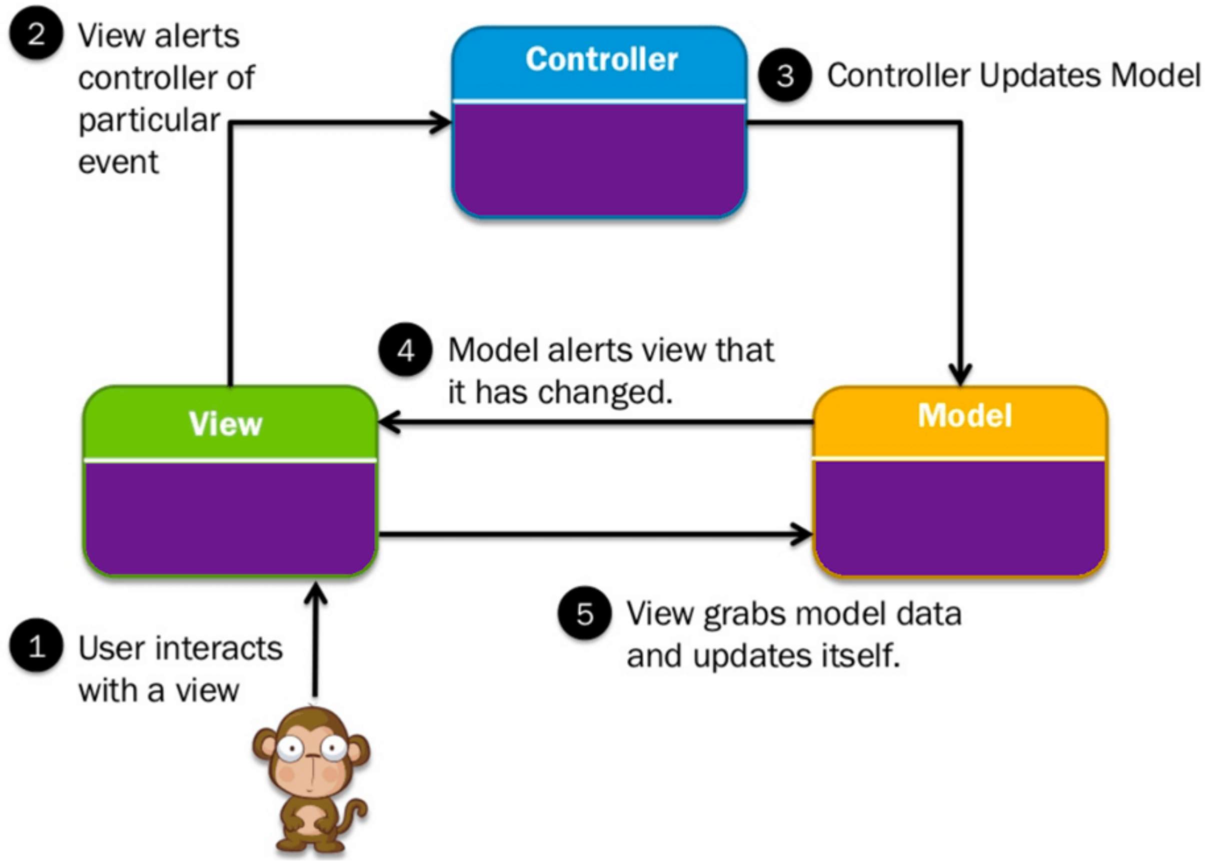
View : View is something that displays data to user.
Simple and free of business logic implementation.
Invokes methods on Controller depending on user actions.
View monitors the model for state change and displays updated model.

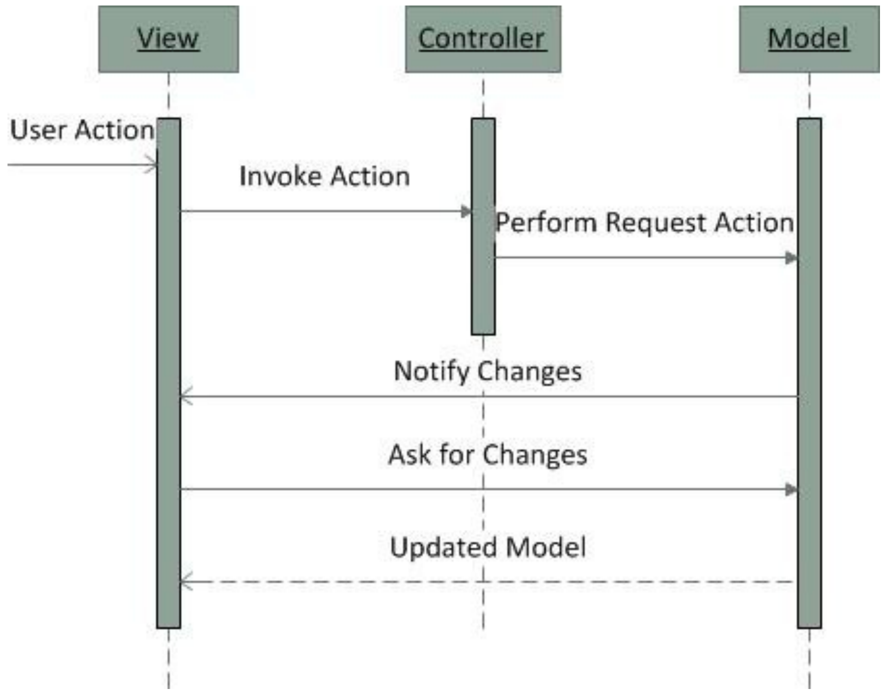
Controller:
Interacts with the model and performs actions that updates the model.
doesn't mediate between the view and model.
simply process the user action and updates model


User Actions
Controller
Selects a new view if required by user action
View
Controller updates model as per user actions
Model triggers view update
Model
View queries model for State

The Classic Model-View-Controller Pattern (MVC)

Understanding the interaction between Model, View and Controller using sequence diagram:


2 View alerts controller of particular event
Controller
3 Controller Updates Model
4 Model alerts view that it has changed.
View
Model
5 View grabs model data and updates itself.
1 User interacts with a view



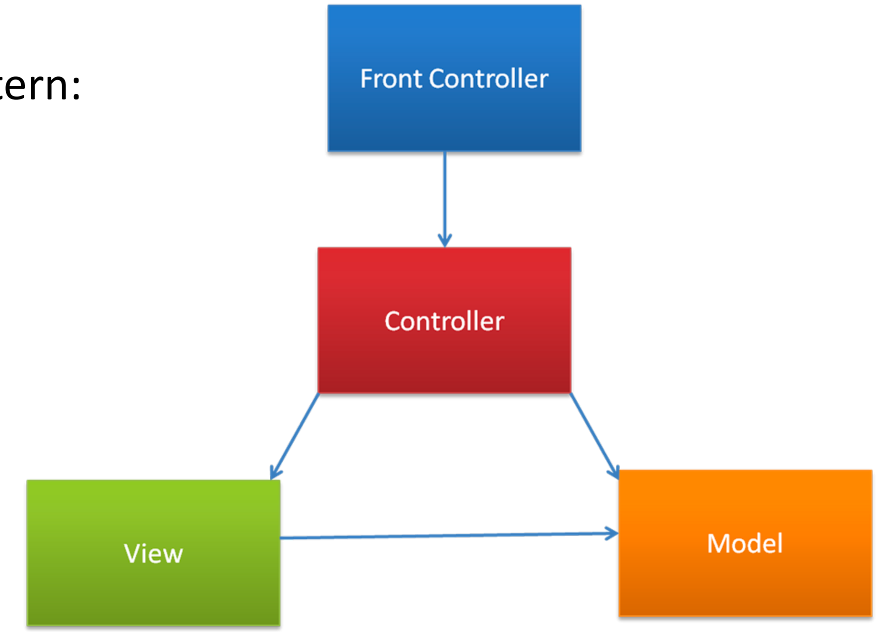
MVC Pattern for Web Applications (Model 2)

Model2 is MVC adapted to web

One of the most popular variation is Model 2

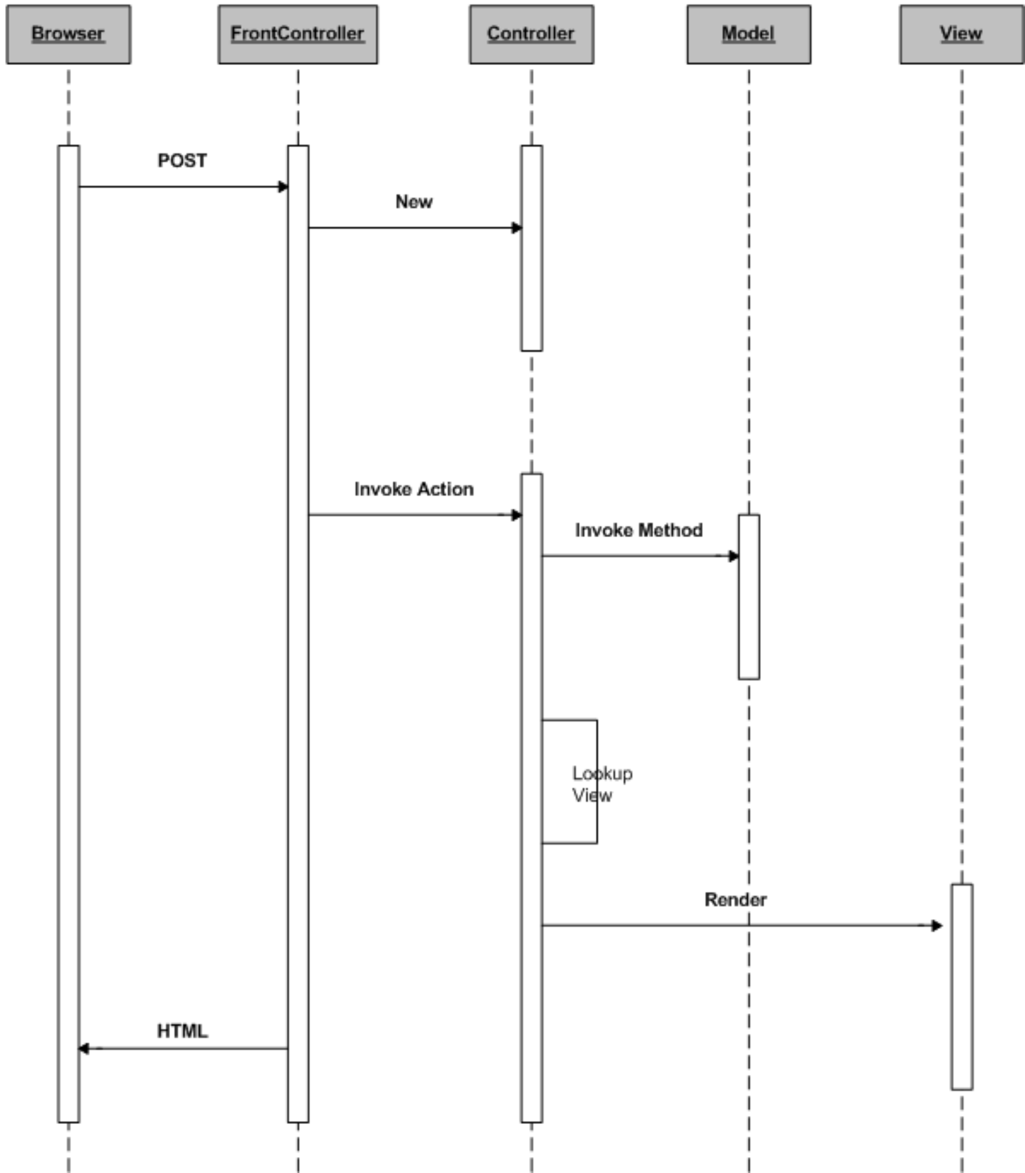
It’s a pattern originally created for Java Server Pages (JSP) and owes a lot of its popularity to Struts framework. It’s the same pattern implemented by the recent ASP.NET MVC framework in .NET technology stack.

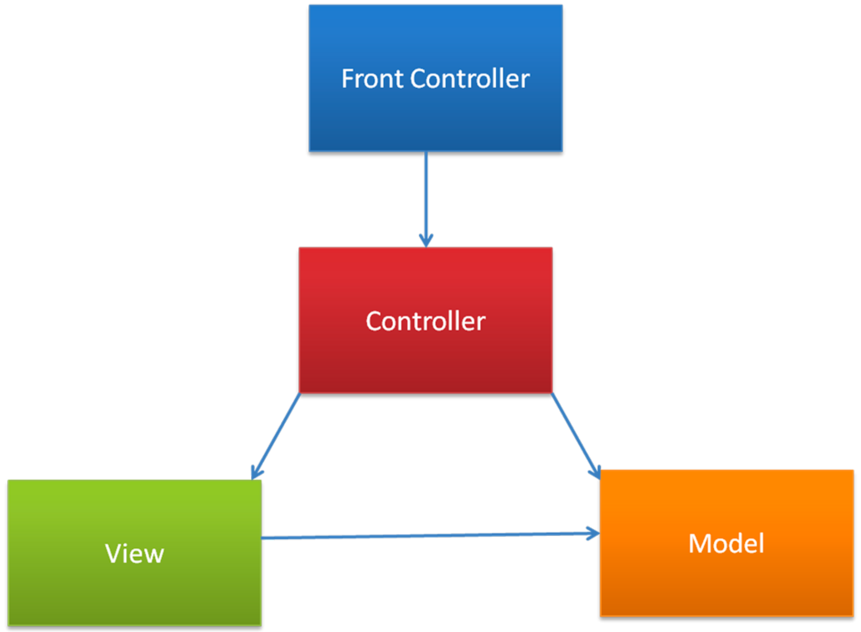
The following diagram depicts the structure of Model2 pattern:



MVC Pattern for Web Applications (Model 2)

Following sequence diagram depicts Model2 interactions using sequence diagram :



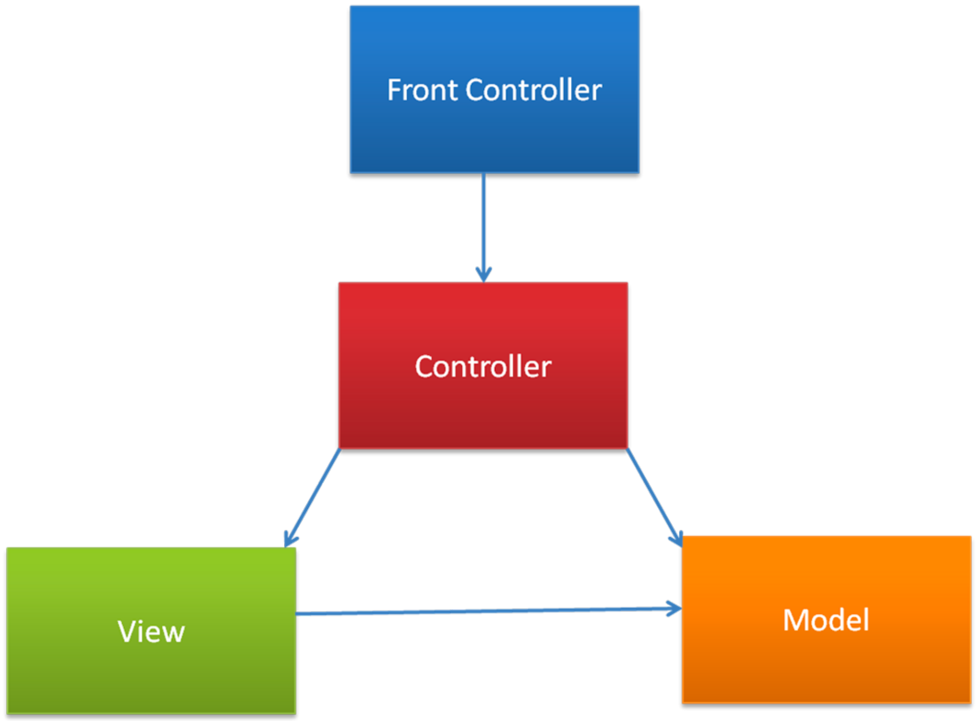


Difference between classic MVC and Model2

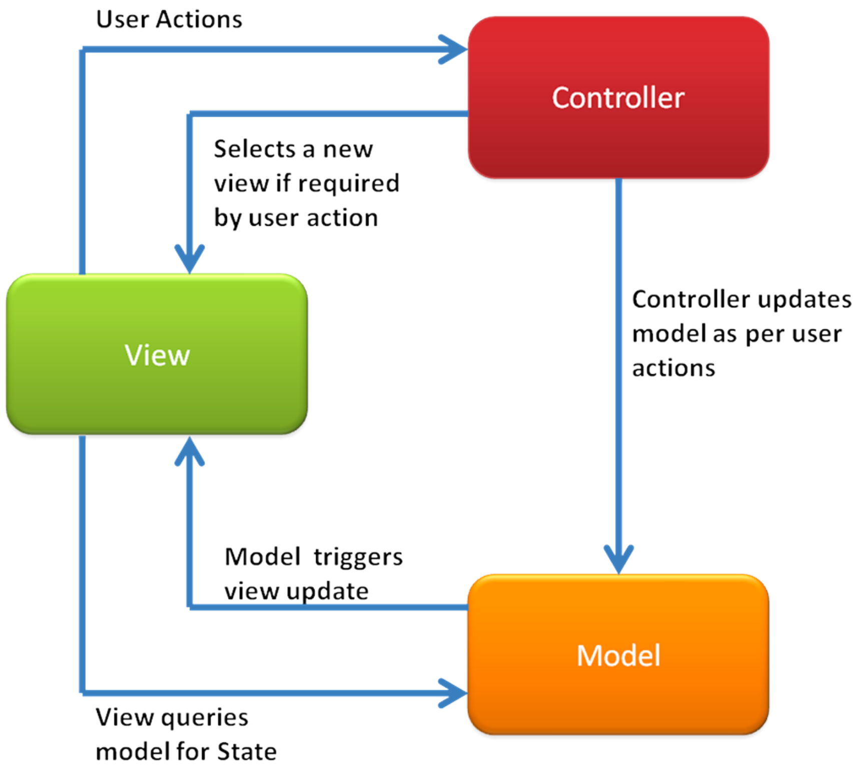
There is no direct contact between view and model in model 2.

The controller will be the one who will talk to business logic layer and update the model.

The interaction between the view and model is an indirect relationship.



MVC Pattern for Web Applications (Model 2)



User Actions
Controller
Selects a new view if required by user action
View
Controller updates model as per user actions
Model triggers view update
Model
View queries model for State

Classic MVC

Classic MVC pattern drawbacks

Classic MVC pattern has two drawbacks:

Model needs to communicate change of state to the view.

The view has complete knowledge of the model, there is no explicit contract between view and model.

Model View Presenter (MVP)

Model View Presenter (MVP) pattern is evolved from MVC, it tries to address the previous concerns. MVP was originally developed at Taligent, in 90’s.

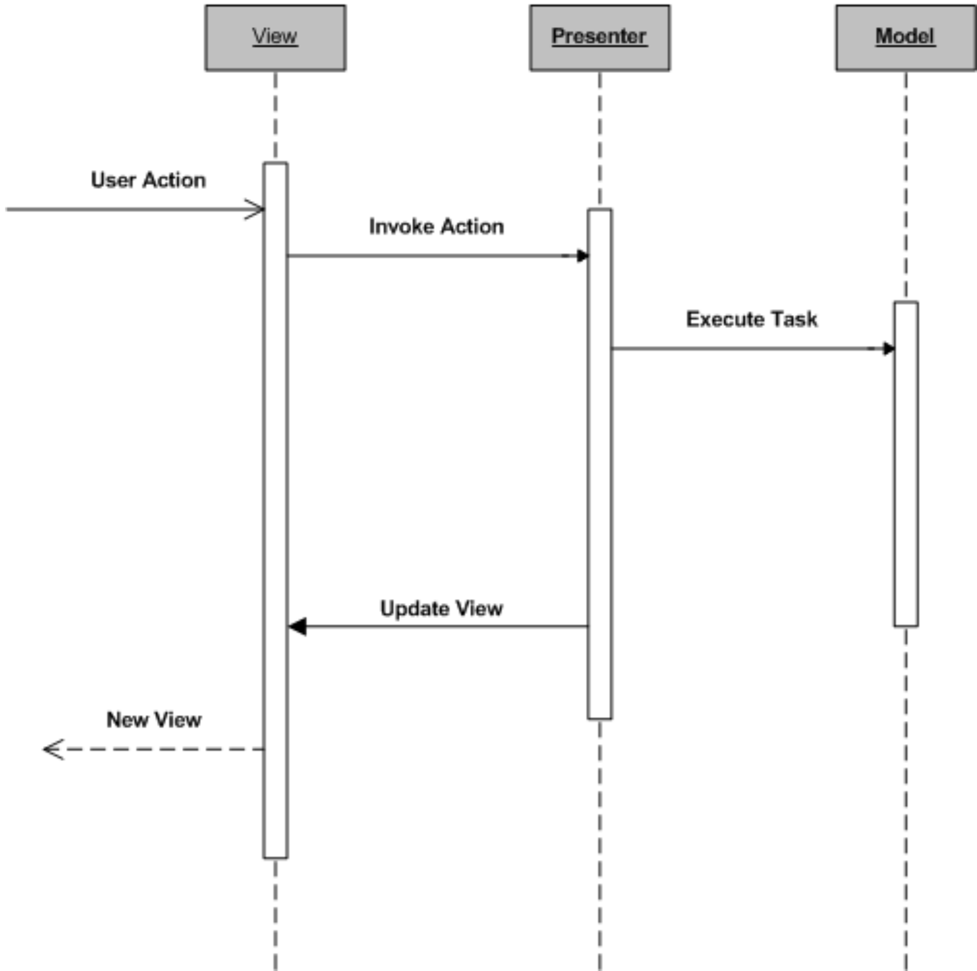
The MVP pattern neatly separates the Model from the View and breaks the direct relationship between them

The View exposes a **contract** through which the Presenter interacts with the View.

The core of MVP is the interaction between View and Presenter.

According to Martin Fowler, you never use MVP, rather you use Passive View or Supervising Controller or both variants.

MVP Pattern Diagram



Model View Presenter (MVP)

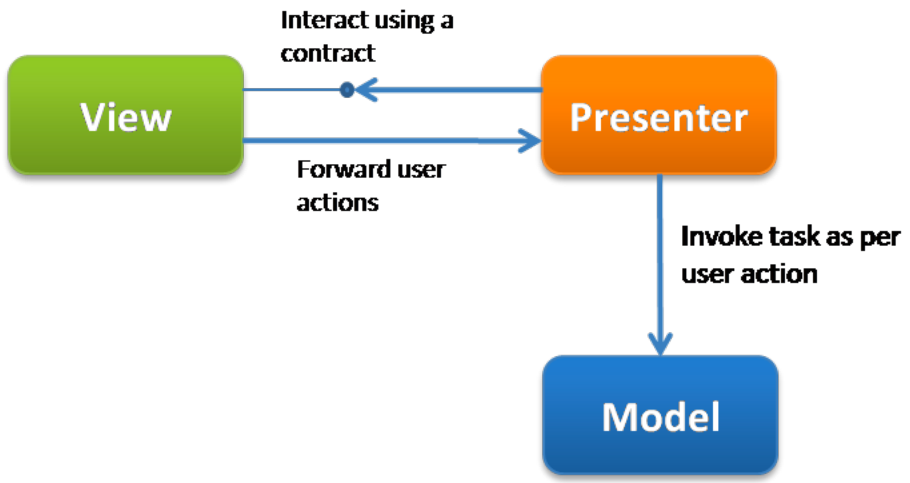
Model : Model in MVP represents business entities or domain model or object model of the business tier.

View : In MVP pattern, View is light weight. It should only have the UI elements and shouldn't be aware of the model. But that is in a ideal scenario, building a real passive view is quite complex in practice, hence the implementation of MVP falls into two categories :

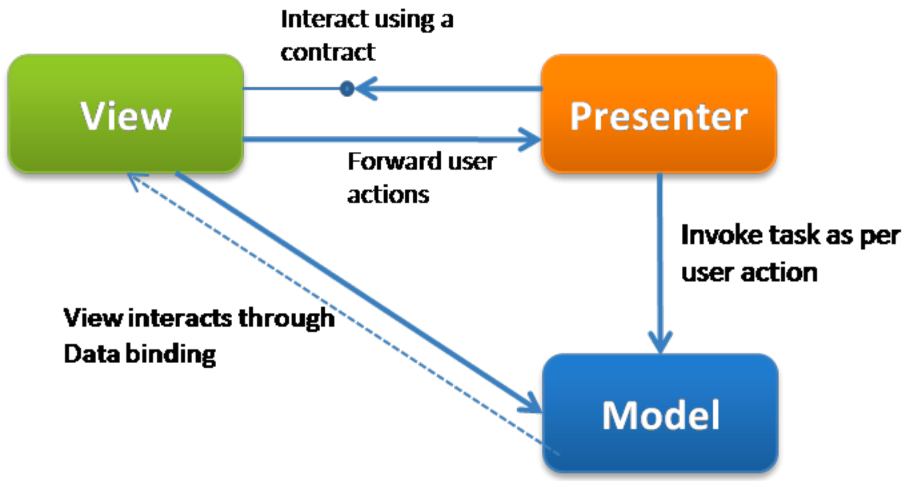
1. **Passive View** :
   The view is really passive, light weight.
   View doesn't know the model

2. **Supervising Controller**
   The view is active, binds the view using data binding or simple code in view

MVP - Passive View



MVP – Supervising Controller


Interact using a contract
View
Presenter
Forward user actions
Invoke task as per user action
View interacts through Data binding
Model

Why Presenter? Why the name change?

Presenter :

The classic MVP triad diagram looks similar to MVC diagram, the noticeable difference is Controller replaced with Presenter.

The Presenter in MVP, presents user actions to the backend system; after getting the response it presents the response to the users, whereas the Controller in the MVC pattern doesn’t mediate between Model and the View, it doesn’t update the view, it just mediates between user actions and model.

Summary

MVC:

Traditional MVC is where there is a

Model: Acts as the model for data

View : Deals with the view to the user which can be the UI

Controller: Controls the interaction between Model and View, where view calls the controller to update model. View can call multiple controllers if needed.

MVP:

Similar to traditional MVC but Controller is replaced by Presenter. But the Presenter, unlike Controller is responsible for changing the view as well.

1. **Passive View**
2. **Supervising Controller**

Questions?

Thank You.