# A Brief Overview of MVC in Contemporary Software Design

Inzamam Rahaman - 810006495

Jherez Taylor - 812003287

Steffan Boodhoo - 812003126

November 29, 2014

## ABSTRACT

Software decays at a faster rate than most other engineering artifacts - the circumstances under which a piece of software must operate can evolve at a rapid pace and render said piece of software obsolete. Consequently, software engineering grew largely out of the need for software that could be easily maintained , debugged , and extended as the environment in which it operated evolved, grew, and adapted to new technological, social, and economic mores. An important notion to arise from this need is that software should be developed such that its components exhibit high cohesion and low coupling. If our components exhibit high coupling, then modifying some component A will directly affect all components that depend on component A.

However, writing software that is decoupled and cohesive is not a trivial task. In light of this fact, software engineers and software developers need methodologies and software architectures that they can leverage to make their code as easy to maintain and as decoupled as possible while facilitating responsible code-reuse, flexibility, and low code complexity. One of the post popular software architectures used in trying to achieve this is the Model-View-Controller (henceforth referred to as MVC) software architecture.

# CONTENTS

# ACRONYMS

**DRY** Don't Repeat Yourself

**API** Application Programming Interface

**MVC** Model View Controller

**MVVM** Model View View-Model

**ORM** Object-Relational Mapper

**OOD** Object-Oriented Design

**UI** User Interface

**HTML** Hyper-Text Markup Language

**CSS** Cascading Style Sheet

**JS** Javascript

# 1

## WHAT IS MVC ?

## 1.1 MODEL

A web application that displays agricultural data to users. An application is a warehouse that monitors the current stock. Both of these applications share an important feature - there is some underlying data model that we want to operate upon in some manner in accord with constraints and domain logic.

Abstracting over the underlying data model and its domain logic that is to be used by an application, and providing services to access and manipulate the data in accord with said domain logic is the responsibility of the **Model**.

In OOD, it is not uncommon for elements of data model or domain to represented as Objects that either encapsulate the data directly, or act as an intermediary between the software and the database system using an ORM.

Whenever the Model is updated, it may notify the entities in the system, which can include the entities in the View, about those changes.

## 1.2 VIEW

For our users to view their requested data, our application would need some mechanism of present our data to our users. The **View** refers to the component or components that facilitate this task.

Since, there can be more than one way to display the data to the user, a single application may have multiple Views - one for each way of presenting data to our end users.

For example, in our web application, our View(s) would comprise our client side markup - our HTML and CSS. Similarly, for our warehousing application, our View(s) would be the display components of our UI that facilitates the presentation of data to our users.

Moreover in many implementations of MVC, the Views can subscribe to the Models using some variant of the Observer design pattern in order to get the information to display to the user.

## 1.3 CONTROLLER

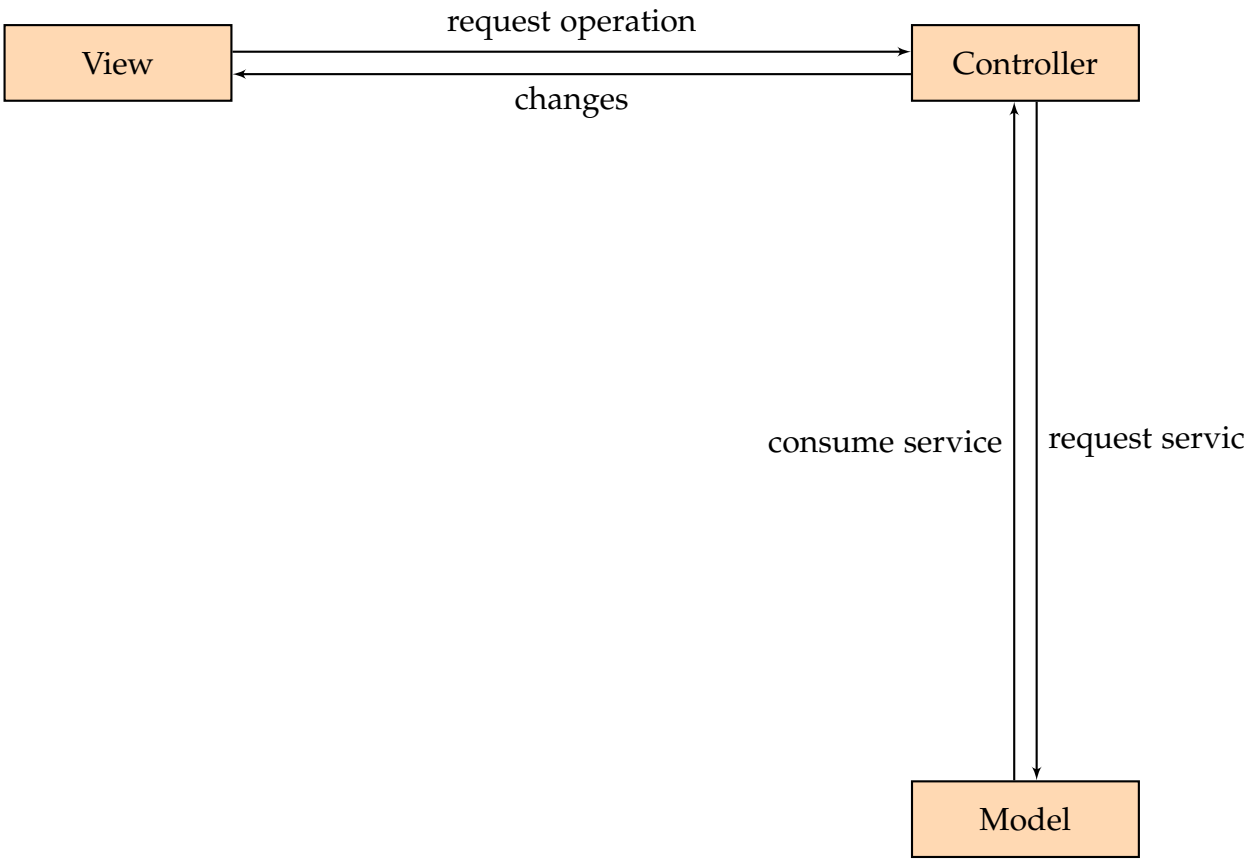
In order to request services from our Model in a manner decoupled from our View, we need some component that acts as a middleman, translating between the two and requesting appropriate services from the Model on the behalf of the end-user utilizing the View. This is the responsibility of the **Controller**.

Since, different Views may have different service requirments of the Model, there is typically a one-to-one mapping between Views and Controllers. As such, an application can have many Controllers.

In addition to supplying the View with a mechanism for interacting with the Model, the Controller can also manipulate the View. For example, the Controller can disable a button in re-

sponse to notifications sent or delivered by the Model. In some implementations of MVC, their is no clear distinction between a View and its Controller.

## 1.4 DIAGRAMTIC SUMMARY




## 1.5 AN EXAMPLE OF HOW IT SOLVES THE PROBLEM

Let us consider the case of an application that renders a table, a bar chart, and a line graph, each on a separate page, for the user.

1. The data that is to be presented to the user is initially stored in some database. Our **Model** is responsible for retrieving the data in our database

2. As aforementioned, our application has three means of presenting information:

   - A table
   - a bar chart
   - a line graph

   Each of these means of presenting information to the user, would require a different **View**, comprising its own UI components

3. Our application would also need for each view, a **Controller** that serves as an intermediary between their respective View and the Model - translating between them and eliciting the responsible updates and responses

Now let us examine our application and our above scheme for MVC to understand how MVC helps make our code modular, resuable, and extensible

1. Recall that we have multiple View, Controller pairs. Each of the View, Controller pairs would need to request and consume similar services from the underlying data model. By encapsulating and abstracting our data model in our Model, we can simply reuse our Model for use in all our View, Controller pairs, thereby reducing the complexity and size of our code base, making our application easier

to extend, and increasing the re-usability of facets of our code base.

2. Many applications are developed by a team of people as opposed to a single person. Consequently, it would a great boon if our software architectures can accommodate a parallel approach to development. MVC can also be helpful in this regard. Since the Model and the Views are decoupled and the Controller is responsible for translating between them the Model and the Views can be developed independently of one another, with the concentration being placed on the services that ought to be provided by the Model.

# 2

# MVC IMPLEMENTATIONS

Why do we implement the MVC pattern, when should we use it, well unless your application is very simple, all the time. The MVC pattern's reasoning is that of separation of responsibilities, so that when modifying whether it be functionality, data or what the user see's, everything in the application does not have to be changed. It makes our code modular, reusable and just easier to maintain. This just sounds like a way of coding, well it is, but there are many different languages which all behave differently. Is it all hard ? No we've MVC frameworks for each language that help us achieve this. The implementation of these frameworks for some of these languages are listed as such:

## 2.1 JAVA

- Spring
- Apache Struts
- jsf (JavaServer Faces)
- tapestry
- stripes and wicket
- GlassFish
- Tomcat
- Jboss
- Jetty

- Wildfly

Through analysis, the frameworks usually includes a library of mark up tags, the use of these tags vary from framework to framework. Examples:

- Apache Struts uses their own tags to present information to the user as well as suitable functionality e.g. a form and it's validation

- Spring uses tags to bind Java code to pages. The tags are used to map paths to different components of the webapp, the DispatcherServlet uses these tag defined paths to service incoming http-requests by calling the correct component according to the request as defined by the tags. The components used by Spring can be subdivided as follows Handler mapping, Controller, View resolver, view. When the request is received by the Dispatcher servlet, it is passed to the Handler mapping which returns the necessary controller. When the controller recieves the request it calls the appropriate service. The DispatcherServlet passes the request path to the view resolver which then returns the appropriate view after which is returned after the model is updated. Note that the views for spring are written in jsp which utilizes tags.

- JSF provides a standard HTML library

In general there are explicitly different classes for Models, usually POJO's, Views that use some markup language, classes to manage the Model(s) update and return view(s).

## 2.2 PHP

- Code Ignitor
- CakePHP
- ZEND
- SYMFONY
- Daemon

Through investigation it has been noted that the MVC frameworks for PHP require little to no configurations(e.g. XML/YAML files). They contain a library. Contains a strict folder structure representing MVC as well as a strict naming convention where the framework itself associates files by location and name. Code ignitor/CakePHP there is a controller class and to use it, we just extend that class. Contollers are initialler loaded from a file that maps pages/uri's to controllers, otherwise controllers can be loaded from controllers. There is a model class and we just extend that class to create our own model. Models are loaded from the controllers from methods defined in the model. Views are written in HTML.

## 2.3 PYTHON

- Django
- Flask
- Pyramid

Flask/Django uses a python file(routes.py/urls.py) which contains functions to service requests, these functions are assigned a uri, and manipulate/retrieve views. Views are implemented using a templating strategy where you can define general templates and inherit them and modify pieces.

```
{%extends <template_name.html> %}
{%Block content%} your modifications go here {%end block%}
```

These templates allow snippets of python like code which have access to the variables of the function in which it was retrieved, i.e. The function would contain variables passed from the HTTPRequest, and the variables would be used in the inherited template to manipulate the view appropriately.

## 2.4 JAVASCRIPT

- Sails
- AngularJS
- BackboneJS
- Ember
- NodeJS (Server Side)
- RhinoJS (Server Side)
- Helma (Server Side)

Depending on the framework, there may or may not be a prototype defined for a model, but basically a model is represented as a javascript object with properties representing the application's data. JavaScript views are about building and maintaining a DOM element. A view typically observes a model and is notified when the model changes, allowing the view to update itself accordingly. They are implemented through templating (such as Handlebars.js and Mustache) to restrict repetition.

Controllers are an intermediary between models and views which are classically responsible for updating the model when the user manipulates the view. Controllers are defined as objects which may have a prototype defined by the framework with method(s) that manage a view and update the Model, it also contains state from the model for the view.

# 3

## WHAT ARE THE BENEFITS OF USING MVC?

MVC is an excellent means of separating presentation and interaction from system data it.

**Modular.** The code becomes modular as there is separation of concerns, views can be developed independently of the database.

**Maintainable.** The code is modular and better structured, therefore easier to understand because the code is separated into modules, bugs may be constrained to a particular module rather than entire system. This then speaks to testing as it becomes easier to test as well since the code is self contained.

**Parallel development by different entities.** The code is not coupled, therefore modules can be developed in parallel, UI can be designed, the business logic developed and the database created all in parallel.

**Easy Interface updates.** Views can be created without interference of the current system and just replace the old ones since they aren't closely coupled.

**Pluggable components.** This ties into updates, because modules are loosely coupled and foster separation of responsibilities, components are easy to plug in to the system. More specifically controllers who control the logic are easier to plugin as the layers for the view and database stay the same but only the logic changes.

**Reusable.**The code is self-contained and responsible for itself, therefore elements such as UI components which may contain some functionality within it such as validation may be reused e.g. buttons, text fields

# 4

## WHAT ARE THE DRAWBACKS OF USING MVC?

Although MVC is an excellent means of separating presentation and interaction from system data it does have its disadvantages.

**Increased Complexity.** When the blueprint for an application is being constructed it is important to consider its purpose and scale. The reason for this is that adhering to the Model-View-Controller structure is that it is not always the most effective method to build a UI based application. If the application being considered is to be simple in nature then the use of separate model, view and controller components increases the complexity of development without much gain.

**Close connection between view and controller.** While the use of the MVC structure mandates that the view and the controller be separate components, they still need to closely related which places a limit on the individual reuse of said components. A view would be need to be used with its controller and vice version. The exception to this would be views that do not trigger updates (read only) and thus they share a controller that ignores all input.

**Close coupling of controllers and views to a given model.** In an MVC structure, the views and the controller components make direct calls to the model. The implication here is that any changes to the model would likely break the code utilized by the view and the controller. If the system in question uses multiple views and controllers then the is magnified.

**Inefficient data access from the view.** Depending on how the model is implemented, it may be necessary for a view to make multiple calls to obtain all of the data required for dis-

play. From this, one can see that the view may unnecessarily request unchanged data from the model thus weakening the performance of the application if the updates occur frequently. The use of data caches within the view can be used to counter this and improve responsiveness.

**Changes must be made to the view and controller when porting the application.** Dependencies that are related to the user interface module of a given application are encapsulated within the view and the controller. The components also contain code that is independent of a specific platform. Thus, porting an MVC application requires the developers to separate the platform dependent code from the platform agnostic code before rewriting the application for a different platform. In light of this, developers can go a step further and encapsulate platform dependencies as required.

# 5

# THE FUTURE OF MVC IN SOFTWARE DEVELOPMENT

The lion's share of software development takes place across the web and it is likely to increase in the future. This is understandable as the internet allows for applications to reach a large audience. As time progressed, web developers looked for ways to structure their applications by turning to software patterns. MVC gained popularity but there are some concerns with how it is treated. A misunderstanding of MVC has resulted in situations where developers think of data as being handled one way, the view handled another way and everything else being labelled under 'controller'.

Angular JS in particular was developed in order to address the root of the issue, that HTML was not defined for dynamic views.

"AngularJS is a structural framework for dynamic web apps. It lets you use HTML as your template language and lets you extend HTML's syntax to express your application's components clearly and succinctly. Angular's data binding and dependency injection eliminate much of the code you would otherwise have to write. And it all happens within the browser, making it an ideal partner with any server technology.

Angular is what HTML would have been had it been designed for applications. HTML is a great declarative language for static documents. It does not contain much in the way of creating applications, and as a result building web applications is an exercise in what do I have to do to trick the browser into doing what I want? "

Facebook has chosen to discontinue the use of MVC. Their concern is that MVC does not not scale up for their needs and

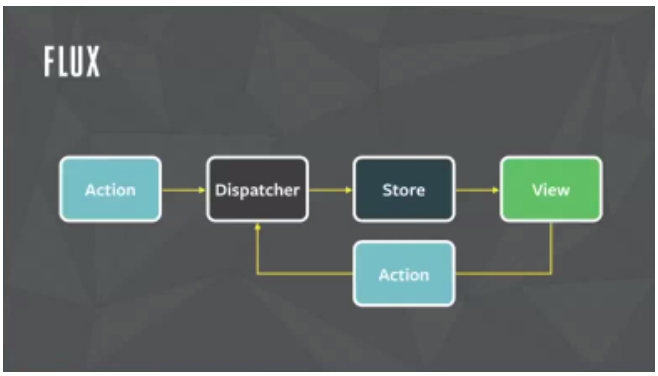



Figure 1: Flux Overview

they have chosen to develop a different software pattern: Flux. http://facebook.github.io/flux/docs/overview.html. Their reasoning is that for a large codebase, implementing and managing MVC becomes complicated. System complexity grew exponentially with each feature addition which lead to the code being "fragile and upredictable." Flux is designed to structure the code in a way that increases the predictability of what action will occur. It promotes single directional data flow through an application.

The above image was sourced from http://www.infoq.com/news/2014/05/facebook-mvc-flux. "The Store contains all the applications data and the Dispatcher replaces the initial Controller, deciding how the Store is to be updated when an Action is triggered. The View is also updated when the Store changes, optionally generating an Action to be processed by the Dispatcher. This ensures a unidirectional flow of data between a systems components. A system with multiple Stores or Views can be seen as having only one Store and one View since the data is flowing only one way and the different Stores and Views do not directly affect each other."

By using this approach, the Data Layer is allowed to complete the update of the view before any other action is triggered and the Dispatcher can reject actions if it is currently processing a previous action. This results in cleaner code that is easier for future developers to debug.

# 6

# CLIENT-SIDE JAVASCRIPT AND MV*

In addition to being used in traditional desktop applications, MVC also finds use in client-side of web applications through client-side Javascript. To this end serveral libraries have been developed to implement MVC using Javacript. In addition to MVC, some of these libraries also facilitate the usage of software architectures that emerged in response to the perceived shortcomings of the MVC software architecture.

| Architecture | Frameworks | | |
|---|---|---|---|
| | AngularJS | EmberJS | BackboneJS |
| MVC | yes | yes | yes |
| MVVM | yes | no | no |
| MVP | yes | yes | yes |