STUDY OF MVC ARCHITECTURE AND IMPLEMENTATION

Ms. Lavina .S. Jadhav
MCA University of Mumbai.

Mr. Burhanuddin A. Sodawala
MCA University of Mumbai

***ABSTRACT*:** *Model View Controller (MVC) is a standard design pattern used in website design or in web application development. It is valuable due to its various capabilities like extensibility, maintainability, and reusability. Commonly we separate a web-based application into three tier architecture. Other than web domain MVC Architecture also uses in embedded domain. MVC achieves this by separating the application into three logical components: Model: The model layer is responsible for the business logic of an application. It will encapsulate access to data stores and will provide a reusable class library. Typically, within the model, you will find facilities for database abstraction, e-mail delivery, validation, and authentication. View: The view layer is typically considered as web design, or templating. It controls the look and feel of data and provides facilities to collect data from the user. Technologies exclusively found in the view are HTML, CSS, and JavaScript. Controller: The controller layer combines everything together and merges the styling of the view with the functionality of the model. It is responsible for collecting input data from the view and deciding program execution. The controller will call model facilities and interpret the returning data so that it can be rendered by the view. It is also responsible for all application exception and flow control.*

Keywords—MVC Architecture, Model View Controller Architecture, MVC

I: INTRODUCTION

The concept of Model-View-Controller (or MVC) is a common one and has become increasingly popular as a design framework. However we found that most introductions to the subject are very abstract. In this paper we try to discuss some of the more practical reasons to go with MVC. Also while MVC can be used for many different kinds of application design, we will focus mostly on its use in web applications. In its most basic sense, MVC is just a clean way of organizing your code. In most web applications, you have a database containing multiple "business objects", for example different products in an online store. Users can go to different pages in the application to view, update, or delete these "objects", which are really just rows in a table most of the time. For any given query, the code in your application needs to be able to access the database, retrieve/update the relevant objects, and generate HTML code to be sent back to the browser. As an example, imagine a simple user management system. People can login and view a list of users. Each user has a name, email address, and password. In addition, there are can be user groups, which are sets of users under a given name. A simple web interface for this system would be to have two listing pages, one of users and one for user groups. The user listing page would display a list of all users, and for each user there would be an "edit" link, that would bring you to a new page containing a form for the different user settings. There would also be a "delete" link to remove a user, and a "create" button as well. The group page would work the same way. One way to implement an application like this would be for each page to have a corresponding file written in say, ASP.NET C#.

The page for listing users would connect to the database, get back all the users, and place the user information inside HTML that it would send back to the browser. Form submissions would be handled by getting the new settings from the form, and updating the database.

While the above system works, it has a number of disadvantages. For one, each file would need to directly execute SQL statements in the database. In the case of a change in schema, each file might potentially have to be updated. Also, because the HTML code is intermixed with the rest of the business logic, changes to the interface could become complicated. In addition there would likely be a lot of code that would be duplicated when it comes to retrieving and manipulating objects. Let's look at how we could code this application with MVC. We can separate and consolidate all of the database code (the model), and all of the interface code (the view). All of the actual database interaction lies in the model, which produces a number of simple methods for manipulating objects. For any given page, there is controller, which gathers the appropriate objects and passes them off to views. The view is responsible for displaying the data in a user interface, is often 90% HTML/JavaScript, with the rest being embedded code (ASP.NET C#). There are many benefits to such a system. First, development can be simplified by allowing the designers to work on HTML and CSS code, and programmers to work on the backend model. These two pieces can be combined with small, simple controllers. Another benefit is in testing. The core of the application lies in the model, which can be tested programmatically dealing with any interface. Changing and updating the user interface is also much easier. In case we described above, MVC is used as way of organizing code. However, there are a number of frameworks that use MVC to make developing an application simpler.

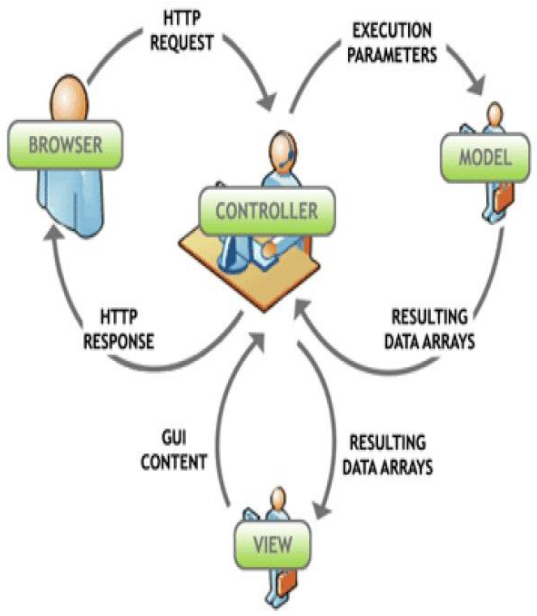

Figure 1 MVC Architecture

II: MVC FOR THE WEB

To apply MVC to the Web, we use a combination of scripts, server pages, and ordinary objects to implement the various components in a framework we call Web Actions. In this context, both versions of the MVC are relevant, particularly the dual uses of the term controller. For HTTP applications, input and presentation are entirely separate, so an input controller distinct from the view is useful. For applications of any complexity, we also need an application controller to separate the details of application flow from the business logic. Figure 3 shows the framework's basic object structure.

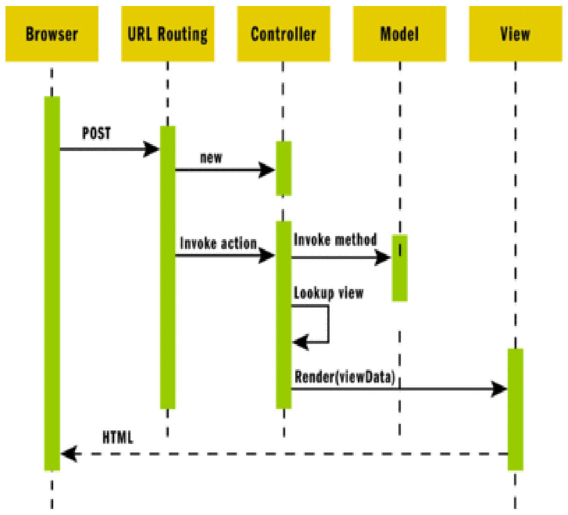

Fig 3. Mvc for web

III: ASP.NET MVC FRAMEWORK

ASP.Net started off using a pattern called Model-View-Presenter, but this pattern got "replaced" when Microsoft introduced ASP.Net MVC, which implements a similar MVC pattern used in Rails or Django. CodeGuru has a good blog post on MVP vs. MVC and in it there's a good graphic explaining the difference between these two patterns:

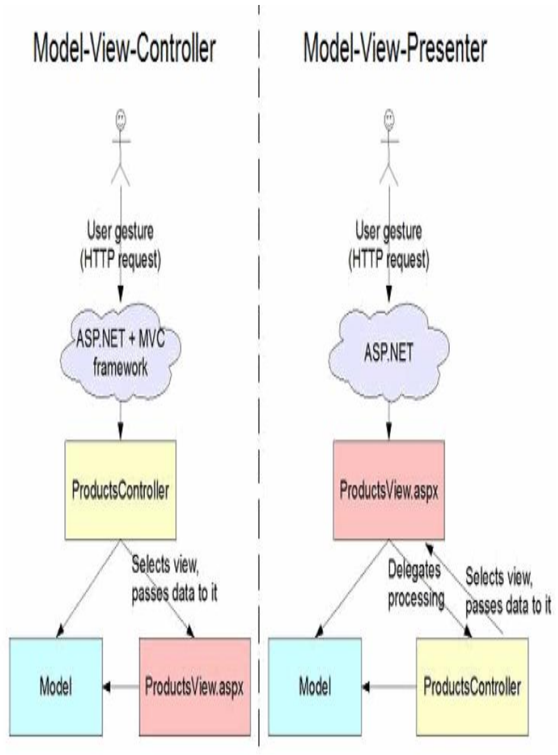

Figure 4 ASP.Net MVC framework

IV: Implementing MVC with Asp.net

From the time that web was used for something than showing static information, developers had to solve a lot of problems derived from HTTP and HTML because they were not thought to support, so many things necessaries to give dynamism to web. So, new concepts and tools were developed to make programming web applications easier and possible such sessions, cookies, etc. We will try to take advantage of these and many other technologies when considered appropriate to help the MVC implementation. In this point it is explained how to implement our MVC model and what technologies we are using for it.

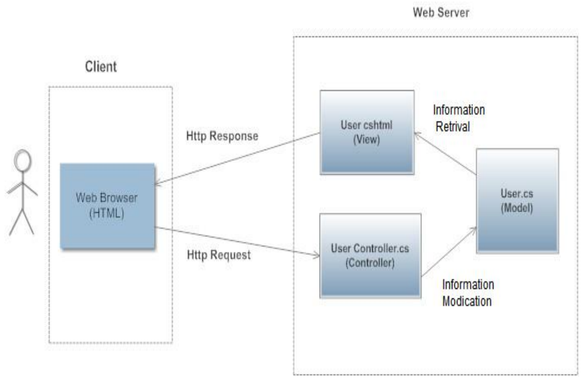

Figure 5: Scheme of the design. It shows the collaboration between different layers

V: Design & Implementation

In this section, we briefly explain the design architecture and operation of the framework. This is a three-tier application framework that is organized into three major parts. The three parts are-

- Users
- MVC components and
- Database.

The first tier consists of only users. User can send request and get response by the "View" components of middle tier. Secondly, middle tier consists of three components named Model, View and Controller. These components can communicate and process data each other. Finally, third tier contains only database where data is stored permanently.

The operations are described in the following steps.

- User input is accepted by the component "View" through graphical user interface .In a stand-alone GUI client, user interactions could be button clicks or menu selections.

- The "Controller" processes the user requests. Based on the user request, the Controller calls methods in the View and Model to accomplish the requested action. A pure GUI controller accepts input from the user and instructs the model and viewport to perform action based on that input. The controller adapts the request to the model. The model represents, or encapsulates, an application's business logic or state. It captures not only the state of a process or system, but also how the system works.

- The view is responsible for the output of the model. A pure GUI view attaches to a model and renders its contents to the display surface.

VI: Advantages

- ▶ Clear separation between Business logic and presentation logic.
- ▶ Each object in MVC have distinct responsibilities.
- ▶ Parallel Development
- ▶ Easy to maintain and enhancements.
- ▶ All objects and classes are independent of each other.

VII: Disadvantages

- ▶ For parallel development there is need of multiple programmers.
- ▶ Inefficiency of data access in view.
- ▶ Knowledge on multiple technology is required.

VIII: Conclusion

In this paper, we have presented a dotnet framework to develop web application software rapidly based on MVC. Software developer can use this framework to build software rapidly. Using this framework, not only achieve the completely separation of view, controller and the model of the MVC Mode, but also achieve a separation of business logic layer and presentation layer. From our testing application, we believe software can be effectively developed by using this MVC framework in a right way and this MVC framework can be one of active participants to software communities. The actual operation has proved that this framework is stable, efficient and able to develop high quality applications.

REFRENCES

1. Alan Knight, N. D. (2002). Objects and the Web. IEEE SOFTWARE , 51-59.

2. Amir Salihefendic. (2011, March). Model View Controller: History, Theory and Usage. Retrieved November 2, 2012, from http://www.amix.dk: http://amix.dk/blog/post/19615

3. Kingsley Sage. (n.d.). Model View Controller (MVC) architecture. 18-27.

4. Marty Hall, L. B. (2003). Integrating Servlets and JSP: The Model View Controller (MVC) Architecture. In L. B. Marty Hall, Core Servlets and JavaServer Pages: Volume 1: Core Technologies (pp. 434-463). Prentice Hall PTR.

5. Micael Gallego-Carrillo, I. G.-A.-H. (2005). Applying Hierarchical MVC Architecture To High Interactive Web Application . Third International Conference I.TECH - 2005, (pp. 1-6).

6. Shivprasad Koirala. (2011, March 17). Comparison of MVC implementation between J2EE and ASP.NET, Who is the best? Part 1. Retrieved November 3, 2012, from http://www.dotnetfunda.com/: http://www.dotnetfunda.com/articles/article1254-comparison-of-mvc-implementation-between-j2ee-and-aspnet-who-is-the-best.aspx

7. Venkat, N. T. (n.d.). Advantages & Disadvantages of MVC Architecture. Retrieved November 3, 2012, from http://www.allinterview.com/: http://www.allinterview.com/showanswers/56984.html

About Author

Lavina .S. Jadhav currently pursuing MCA Final Year from ASM's Institute of Management & Computer Studies, IMCOST, Thane which belongs to University of Mumbai, has an interest in learning new technologies.

Burhanuddin .A. Sodawala currently pursuing MCA Final year from ASM's Institute of Management & Computer Studies (IMCOST), Thane, which belongs to University of Mumbai, has an interest in Android Development.