Achieving Loose Coupling Between Different Components of Model-View-Controller

For Web Based Application

Dr. S. A. M. Rizvi
Department of Computer Science, Jamia Millia Islamia University, New Delhi.
samsam_rizvi@yahoo.com

Syed Imtiyaz Hassan
Department of Computer Science, Hamdard University, New Delhi.
s.imtiyaz@yahoo.com

ABSTRACT
The task of providing different access paths to the same set of data simultaneously was successfully addressed by Model-View-Controller (MVC) design pattern. While MVC provides a highly organized and uniform decomposition of the tasks performed by an application in the form of Model, View, and Controller, the close relationship between them leads to poor scalability and maintainability. Every framework based on MVC has implemented it differently; trying to minimize the coupling between its components. As MVC was not explicitly designed for web application and no enough study has been done on implementation guideline, this often results in misapplication of the pattern in web development environment. To address these issues, we propose an evolution of MVC pattern called MVC-web. MVC-web can be a way to reach loose coupling among different components and can also be used as a general guideline while implementing MVC for web based application.

KEYWORDS
Model-View-Controller (MVC), Web Application, User Interface Design, Human Computer Interface (HCI), Design Pattern.

INTRODUCTION
The object-oriented paradigm of software development has led to the emergence of the concept of open systems, which is flexible to upgrade and maintain with minimum efforts. It encourages reuse by inheritance and polymorphism, but still the reusability can only be reached by a careful design [1]. Nowadays, as the sizes of software products grow larger and larger, the architecture, design and implementation have to be well organized, clear and maintainable. Traditional structured methods are severely limited to design such systems in that they offer limited means of reasoning about the validity of specifications and hence there is a need to integrate structured method with design pattern (or formal methods) [2]. A design pattern [3] is a form of solution intended to address problem and the purpose of the pattern is to reduce the complexity of user interfaces for a large and complex information system. The ability to compose user interface representations by composing component specification is an essential requirement for design pattern. A design pattern is not all about program proving; rather, it works largely by making one think very hard about the system one proposes to build [4]. The emergence of World Wide Web and Internet has brought further impetus and applicability to object oriented design pattern all over the globe. To mitigate the risk associated with creating a robust client tier, developers have produced several frameworks and design patterns with varying degrees of success. One of the most famous and most used is surely MVC [5, 6], which divides each application (or part of it) into three different fundamental elements and states the rules for linking them together [7]. While MVC provides a highly organized and uniform decomposition of the tasks performed by an interactive desktop application, the components prescribed by the MVC are not agnostic of the web development environment. However, most of the web based descriptions give their own interpretation while implementing MVC for web based application. This often results in the misapplication of the pattern. Thus we propose MVC-web, a generalized implementation guideline of MVC for web. This guideline also produce loose coupling among different components of MVC.

THE MVC AND ITS WEAK POINTS
Providing user interface for meaningful observation and manipulation of data was thought in the year 1971 while conceptualizing the language Smalltalk [8]. Thinking on same line, MVC was conceived in 1978 as the design solution to a particular problem with the top level goal to support the user's mental model of the relevant information space and to enable the user to inspect and edit this information. One of the motivations behind the MVC pattern was to allow alternative visual representations of the same object such as a table or a graph. [5] defines MVC as follows:

MVC (Figure 1) programming is the application of the three-way factoring whereby objects of different classes take over the operations related to the application domain (the Model), the display of the application's state (the View), and the user interaction with the Model and the View (the Controller).
In MVC, the Model represents the state, structure, and behavior of the data being viewed and manipulated by the user. The Model contains no direct link to the View or Controller, and may be modified by the View, Controller, or other objects with the system. When notification to the View and Controller are necessary, the Model uses the Observer Pattern [3] to send a message informing its dependants that the data has changed. A View is responsible for displaying the data held by the Model. A Controller connects the Model and the View and coordinates activities between them. Based on user inputs, Controller determines which methods on the Model should be invoked and which View should display the updated data in the Model. The Controller is responsible for processing input from the user. The Model does not have any dependency on Views or Controllers. A View depends on its associated Model. A Controller depends on its associated Model and View.

One of the major drawbacks of MVC is the close relationship that exists between different components, which leads to poor scalability and maintainability of applications [6, 7, 9]. Another drawback, the lack of availability of general implementation guidelines for MVC for the web development environment often results in misapplication of the pattern. The major APIs based on MVC: .Net [11], Struts [12], and Java Swing [13] offer substantially different interpretations of how the core concept should be turned into a practical implementation, strongly suggesting that MVC need an implementation guideline for web applications.

RELATED WORK

Below is the short review of the notable works done on design patterns for developing interactive software systems.

Presentation-abstraction-control (PAC) [14] was proposed in 1987, somewhat similar to MVC. PAC used a hierarchical structure of agents, each consisting of a triad of Presentation, Abstraction and Control parts. One major difference between MVC and PAC is that inputs and outputs both are processed by Presentation (View) in PAC. The PAC model structures the interface system as a composition of PAC agents, but does not define a particular communication and control mechanism. Moreover, PAC is only the conceptual pattern. A very similar pattern, the Abstraction-Link-View (ALV) [15] was proposed in 1993 that was not simply a conceptual architecture, but had been implemented. Its Abstraction components have the same designated role as in MVC. The View component of ALV is similar to the Presentation of PAC, and the Link of ALV is similar to the Controller of PAC. ALV supports the separation of the Abstraction and the Views, by storing all the information in each View it needs from the Abstraction. The Abstraction holds data that is shared between the multiple users of the system. This means that there is a significant redundancy amongst Views and between each View and the Abstraction. Due to the significant redundancy in ALV, it is not a good choice for implementation. Thinking on the same line, Brown recorded the 4-layer pattern [16] in 1995. The four layers are labeled GUI (View), Mediator, Domain model, and Infrastructure layer. In this four-layer pattern, a single business concept (such as a User) will usually be represented in all four layers, in different forms. Moreover, the relationships between the elements in those four layers often require a complex, many-to-many mapping. Another variation of PAC under the name Hierarchical-Model-View-Controller (HMVC) was published in [17] in the year 2000. As described by [17] the main difference between HMVC and PAC is that HMVC is less strict in that it allows the View and Model of each agent to communicate directly, thus bypassing the Controller. This freedom led the tight couple relationship between Model and View. While MVC provides an efficient framework for developing GUI interaction, HMVC scales it to the entire client tier. The HMVC pattern decomposes the client tier into a hierarchy of parent-child MVC layers. The repetitive application of this pattern allows for structured client-tier architecture. The layered MVC approach assembles a fairly complex client tier.

All the above variations of MVC do not address the issues related to Internet software development. A very popular variation of MVC specifically addressed Java implementation of MVC in Internet is Model 2 architecture [18]. It was proposed to gain optimal separation of presentation from content by using Java Server Page (JSP) and Java Servlet in web application. It takes advantage of the predominant strengths of both technologies, using JSP to generate the Presentation layer and servlets to perform process-intensive tasks. Although Model 2 of JSP has implemented the basic idea of the MVC design pattern, the relationship of these three layers (Servlet, Java-Bean/EJB and JSP) is still close [9]. That is why the latest frameworks Swing, Struts and Spring have changed Model 2 architecture for implementation.

MVC-web AND ITS ARCHITECTURE

MVC-web describes general implementation guidelines of MVC in the context of web application development. Though an implementation note is provided to implement it in Java, the guideline is general enough to implement MVC-web with any programming language.

The definition of Model, View and Controller in MVC-web is same as described in original MVC. This is to note that, in web applications, the server side code which is responsible for the
user interface is usually referred to as the View. Perhaps this is to distinguish it from the client browser which is closer to the user and is often referred to as the user interface. In other words, the View is the layer which constructs the User Interface but resides at the Server.

To eliminate the indirect coupling between Model and View (For change notification in Model to View) a new component ‘Dispatcher’ is introduced between View and Model as shown in Figure 2. The Dispatcher registers both Model and View for event notification purpose. Whenever Model causes an event due to the change of state, a notification is passed to the Dispatcher. The Dispatcher scans all its registered View and the appropriate View method is invoked and the View updates itself. There is another benefit of using Dispatcher. The burden of registering different listeners and its management is transferred from Dispatcher to the Controller, which implements a very complex logic for transfer of control.

Since the real life solution is complex enough. So, during implementation, the responsibilities of Controller may often be divided between one Front Controller and many Page Controllers. The Front Controller accepts all input from the user, sets system variables, checks security, and invokes the appropriate Page Controller. A Page Controller then invokes one Model and one View. More than one Front Controller can be used in distributed sites or in otherwise segmented sites such as a site with a secure and an unsecured area. A configuration file can also be used for these Controllers for initialization information.

The main design patterns used by MVC-web are the Observer, Composite, and Strategy one [3]. The observer pattern is used to notify dependent objects about changes. The composite pattern enables to treat individual objects and compositions uniformly, such as views and composite views. The strategy pattern makes algorithms interchangeable, e.g. the controller class can be configured with one of many behaviors. In general the MVC-web pattern and its design patterns address design for change.

The primary difference between this design and that of traditional MVC is that the notification of the state changes in Model objects is communicated to the View through the Dispatcher. View objects, as always, use the Controller to translate user actions into property update on the Model. The other difference is; we provide a configuration file for Controller which increases the extensibility of Controller as change in configuration file does not need recompilation of application.

IMPLEMENTATION NOTE TO JAVA
In Java, the Controller is generally implemented with Servlets and Configuration file can use XML. Model can be implemented with Java Beans or Plain Old Java Objects (POJO). One should use Java Reflection API to decouple Model with Controller. Dispatcher is a Servlet, which can be thought as adapter design pattern [3]. It converts the interface of a Model class into another interface the clients expect. The View usually implemented with JSP, HTML, CSS, XSLT and other designing tools.

CONCLUSION
The improved design pattern MVC-web based on MVC, is explicit than the traditional one for web application development environment. Advantage of adopting this pattern is twofold. The coupling between Model and View is looser than the original one, due to the placement of Dispatcher between them and it serves as an implementation guideline for web development. Adherence to guidelines MVC-web will improve the flexibility, maintainability and scalability of web application greatly.

FUTURE SCOPE
Although the MVC-web provides an adequate separation by adding Dispatcher in MVC, it can be enhanced further. While implementing MVC-web, we use XML for binding information. XML binding is static in nature. Possibilities of incorporating dynamic binding information in MVC-web can be explored further. Future work will also include an investigation of how best MVC can be implemented in next generation Web 2.0 using Rich Internet Applications (RIA) technologies and in mobile application using portlet or other related technologies within resource limitations in order to provide attractive user interfaces.

REFERENCES


