Model-View-Controller application architecture

Introduction

The MVC (Model-View-Controller) pattern is an architectural pattern for software engineering. The pattern enables the programmer to separate the UI(presentation layer), application data and business logic making it much easier to maintain code and accommodate changes.

- **Model -** The Model is a representation of data specifically used in an application.
- **View -** The UI of an application. It renders the model for display or interaction.
- **Controller -** Handles events from user interaction and typically invokes changes on the Model.

In the MVC design pattern , application flow is mediated by a central controller. The controller delegates requests to an appropriate handler. The controller is the means by which the user interacts with the web application. The controller is responsible for the input to the model. A pure GUI controller accepts input from the user and instructs the model and viewport to perform action based on that input. If an invalid input is sent to the controller from the view, the model informs the controller to direct the view that error occurred and to tell it to try again.

A web application controller can be thought of as specialised view since it has a visual aspect. It would be actually be one or more HTML forms in a web application and therefore the model can also dictate what the controller should display as input. The controller would produce HTML to allow the user input a query to the web application. The controller would add the necessary parameterisation of the individual form element so that the Servlet can observe the input. This is different from a GUI, actually back-to-front, where the controller is waiting and acting on event-driven input from mouse or graphics tablet.

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. It notifies any observer when any of the data has changed. The model would execute the database query for example.

Control is then usually forwarded back through the controller to the appropriate view. 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. In addition, when the model changes, the viewport automatically redraws the affected part of the image to reflect those changes. A web application view just transforms the state of the model into readable HTML. The forwarding can be implemented by a lookup in a mapping in either a database or a file. This provides a loose coupling between the model and the view, which can make an application much easier to write and maintain.

MVC in Java Server Pages
Java Server Pages (JSP) becomes more interesting because the HTML content can be separated from the Java business objects. JSP can also make use of Java Beans. The business logic could be placed inside Java Beans. If the design is architected correctly, a Web Designer could work with HTML on the JSP site without interfering with the Java developer.

The Model/View/Controller architecture also works with JSP. In fact it makes the initial implementation a little easier to write. The controller object is master Servlet. Every request goes through the controller who retrieves the necessary model object. The model may interact with other business entities such as databases or Enterprise Java Beans (EJB). The model object sends the output results back to the controller. The controller takes the results and places it inside the web browser session and forwards a redirect request to a particular Java Server Page. The JSP, in the case, is the view.

The controller has to bind a model and a view, but it could be any model and associated any view. Therein lies the flexibility and perhaps an insight to developing a very advanced dynamic controller that associates models to a view.

The prior sections have concentrated on their being one controller, one model, and one view. In practice, multiple controllers may exist - but only one controls a section of the application at a time. For example, the administrator's functions may be controlled by one controller and the main logic controlled by another. Since only one controller can be in control at a given time, they must communicate. There may also be multiple models - but the controller takes the simplified view representation and maps it to the models appropriately and also translates that response back to the view. The view never needs to know how the logic is implemented.

The case for separating presentation and logic
Decoupling data presentation and the program implementation becomes beneficial since a change to one does not affect the other. This implies that both can be developed separately from the other: a division of labor. The look and feel of the web application, the fonts, the colours and the layout can be revised without having to change any Java code. As it should be. Similarly if the business logic in the application changes, for instance to improve performance and reliability, then this should not cause change in the presentation.

A model-view-controller based web application written with only Java Servlets would give this decoupling. If the presentation changed then the Java code that generates the HTML, the presentation, in the view object only has to change.

Similarly if the business logic changed then only the model object has to change. A web application built with MVC and Java Server Pages would be slightly easier if the business logic is contained only in Java Beans. The presentation (JSP) should only access these beans through custom tag libraries. This means that the Java Beans did not have Java code that wrote HTML. Your beans would only concern themselves with the business logic and not the presentation. The JSP would get the data from the Beans and then display the presentation (the "view"). Decoupling is therefore easy. A change to the implementation only necessitates changes to the Java Beans. A change to the presentation only concern changes to the relevant Java Server Page. With Java Server Pages a web designer who knows nothing about Java can concentrate on the HTML layout, look and feel. While a Java developer can concentrate on the Java Beans and the core logic of the web application.

(C) Copyright Nokia 2013. All rights reserved.

(C) Copyright Nokia 2013. All rights reserved.