A STUDY OF MVC – A SOFTWARE DESIGN PATTERN FOR WEB APPLICATION DEVELOPMENT ON J2EE ARCHITECTURE

Vedavyas J 1, Venkatesulu.S 2

1 Asst. Prof. Department of MCA, BITM, Bellary, VTU
2 Asst. Prof. Department of MCA, BITM, Bellary, VTU

ABSTRACT

The Model-View-Controller design pattern is cited as the architectural basis for many J2EE web development frameworks. This paper presents an analysis of those changes, and proposes a separate Web-MVC pattern that more accurately describes how MVC is implemented in web frameworks. The MVC is very useful for constructing dynamic software systems. Partitioning decisions can be changed without modifying the application. This paper brings the conception of Web-Application Partitioning, and programming model that allows engineers to implement the Model View Controller design pattern in a partition-independent manner.

Key Words: Model, View, Controller, Framework, Partitioning

1. INTRODUCTION

MVC is the design pattern for the architecture of web applications. Many languages have implemented the frameworks and adopted them universally. Basics Components of MVC model.

1. Model: business logic & processing
2. View: user interface
3. Controller: navigation & input
MVC Design pattern is one of the most fundamental architecture for web applications like J2EE, .Net, Rails and Struts etc. J2EE and JSP technologies are the fundamentals for struts 2 framework. This Struct2 framework consists of MVC pattern as follows.

- **User Interface component** as views
- **Application logic** as Model
- **Control functions** as Controller

The view component of strut 2 framework is done by embedding JSP tags which provides diversified functionalities like flow control, accessing model component and effectual HTML forms structures. The controller component is corporeal with java classes for the actions to be implied. Each action has a responsibility to validate user Input and to engineer transaction processing by invoking appropriate model operations.

The XML configuration file or the java annotation mechanisms are used to outline and configure the actions. Such information is used to control the flow of web applications by finding the outcomes of each action. Value stack eliminates much of the tasks involved in handling HTTP requests and provides the information for JSP to display. It is the key factor which contains the information between view and controller and converts when needed.

This section describes how the MVC is being represented in the web application frameworks. It also reflects the evolutionary changes in the web frameworks.

The primary responsibilities of MVC-Web model are:

1. It has to maintain a database for the data persistence
2. It has to execute the application logic that operates on the application state called transaction processing.
3. It has to manage interactions with external agents such as web services known as External Interface.
4. It should handle query to provide the information to view and controller elements in response for queries.
The primary responsibilities of the view component are:

1. It is used for information retrieval and display because it displays information to the user based on the query in the model.
2. It provides input forms and controls for the user to interact with the application.
3. It provides interactive dynamic behavior at the client side for the users.

The primary responsibilities for the MVC-Web Controller are:

1. It receives the incoming request and routes them to appropriate handler.
2. It receives the request parameters and handles the action such as invoking appropriate model elements.
3. It provides the response for the request depending upon the action invoked.

2. J2EE WEB APPLICATION: AN EXAMPLE ON PARTITIONING

Let us consider the web-application where a client wants to fetch information about a company’s employees in a simple way by executing two operations.

1. By supplying a name, and clicking on a "search" button, search the employee directory "by name". The search returns the set of employees that match the search criteria in a format that displays an abbreviated employee record for each member of the returned set.
2. By clicking on a "details?" button, get detailed information about a specific employee. Implementation in a stand-alone, single address-space, environment, is straightforward. From the perspective of the MVC design pattern (see Figure 1):

   The Model consists of the records in the employee directory. There are four Views: a "search" panel; a display of abbreviated information about a set of employee records; a display of detailed information about a specific employee; and a report that no employees match the search criteria.

   There are two Controllers: one that, given a "search" directive, drives the process of querying the Model and returns a result set; and one that, given a "details" directive, queries the Model to get the full set of information about the specified employee. Implementation as a web-application in a server environment raises the issue of partitioning which is conceptually irrelevant to, but in practice complicates, the MVC design pattern. Naively, as there are two Controllers, the application can be implemented in one of four ways. Either both Controllers execute exclusively on the client or server, or one Controller executes on the client and the other executes on the server. Each partitioning decision greatly affects the way that the application is implemented. For example, if both Controllers run on the client (the "fat-client" approach), the entire Model must be downloaded to the client -- which is often impractical. If both Controllers run on the server (the "thin-client" approach), two round trips between client and server must be performed each time that the client searches for an employee and then asks for more detail about that employee.
In fact, for many environments, either the thin-client or the fat-client is ideal. Instead, using a dual-MVC approach, we partition the Controllers between the client and server. Specifically, the "search" Controller executes on the server in association with a Model consisting of the complete employee directory. However, when returning relatively small sets of employee records, the Controller also returns the full record for each of the employees, so that they can be maintained in the client-side Model. The dual-mvc approach allows requests for detailed employee information to be served by the client, thus eliminating a client/server interaction. (This implementation is beneficial only when application scenarios typically consist of a preliminary search for an employee using a "partial name", followed by request for more information after the specific employee is determined by inspection. Remember: this is only a motivating example!)

Of course, what we really want is to do avoid partitioning while implementing the application, since the correct partitioning decision depends on factors that are not necessarily determined until actual deployment. For example, if the employee directory is relatively small, the "fatclient" approach with both Controllers executing on the client makes sense and would provide better performance.

Conversely, if the application is deployed in a "internet" environment in which users want minimal customization of their environment, the "thin-client" approach may be the only solution possible. Delaying application partitioning for as long as possible is even more attractive because partitioning gets in the way of designing the Views and developing the business logic needed by the Controllers. Flexible web-application partitioning addresses these needs. In fact, flexible web-application partitioning goes further, allowing partitioning decisions to vary dynamically, during application execution.
The J2EE programming model explicitly supports the MVC design pattern, and enables programs executing in MVC mode to execute in a single address space. When deployed, these programs can be flexibly partitioned without changing the source code used during smvc development. We refer to such J2EE applications as J2EElications.

III. FUTURE WORK AND CONCLUSION

This paper describes how the partition-independent Model View Controller design pattern can be used in the intrinsically locution-dependent environment of partitioned Web-applications. By understanding the scenario flows, the application can be partitioned in a way that improves performance. In contrast, traditional implementation techniques require that such analysis be performed only in the design and requirements phase because it is much too costly to repartition the application once it is deployed. Unfortunately, the necessary insights can often be made only after the application has been deployed and in production for some time. In future repartitioning, under fwap, imposes no extra cost; an application can therefore be readily tuned after deployment based on feedback from actual client use. We are currently implementing the algorithms and infrastructure needed to enable fwaplications to scale over non-trivial application Models. We are also working with a customer to validate the fwap concepts and implementation.

REFERENCES