THE MVC-WEB DESIGN PATTERN

Ralph F. Grove and Eray Ozkan
Department of Computer Science, James Madison University, Harrisonburg, VA, U.S.A.

Keywords: Web, Web framework, Design patterns, Model view controller pattern.

Abstract: The Model-View-Controller design pattern is cited as the architectural basis for many web development frameworks. However, the version of MVC used for web development has changed as it has evolved from the original Smalltalk MVC. 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.

1 INTRODUCTION

The Model-View-Controller (MVC) design pattern is cited as the basis for the architecture of several web application frameworks, such as ASP .Net, Rails, and Struts. The MVC pattern was originally implemented in the Smalltalk-80 programming environment developed at Xerox PARC (Goldberg and Robson, 1985). As it has been adapted for web frameworks the MVC pattern has evolved in different ways, resulting in implementations that differ significantly from each other and from the original Smalltalk implementation.

The first goal of this paper is to present the MVC design pattern, both in its original form (section 2) and the variations currently used in well-known web application frameworks (section 3). Second, we present an evaluation of the changes in the pattern as it has evolved and the effectiveness of the evolved version (section 3). Finally, we propose a standard MVC-Web design pattern description that reflects the current use of the pattern in web frameworks while maintaining the original desirable qualities of MVC (section 4).

Revisions of the MVC-based web application framework design have been proposed (Chun, Yanhua, and Hanhong, 2003) (Barrett and Delaney, 2004). This paper, however, does not propose a new MVC architecture, rather it analyzes and documents the evolution of the MVC pattern as it was adapted from Smalltalk to web frameworks.

2 SMALLTALK MVC

The MVC design pattern was introduced with the Smalltalk programming environment as a way to structure interactive applications in a modular fashion (Krasner and Pope, 1988). As the name implies, the MVC design pattern decomposes functionality into three major components.

The *model* component encapsulates the domain-specific structure and functionality of the application. This essentially includes the state of the application and operations that can change state. The model also maintains dependencies of view and controller components, which it notifies in the event of changes in state. This behavior is an instance of the Observer pattern (Gamma, Helm, Johnson and Vlissides, 1995). The *view* component presents information to the user through a graphical user interface. There may be multiple views of different types operating within the application, presenting different views to multiple users. Views may also be hierarchical, constructed from smaller (subview) elements. When information contained in a view is updated (by a model component that is responsible for that information) the view is notified by the model and then the view may query the model to obtain information that it needs to present. The *controller* component responds to user actions via the user interface. It is responsible for passing transactions to the model for execution. Controllers exist in a one-to-one correspondence with views. The hierarchy of views is therefore also replicated among the corresponding controllers. When a controller receives input, it yields to its active

F. Grove R. and Ozkan E..
THE MVC-WEB DESIGN PATTERN.
DOI: 10.5220/0003296901270130
In *Proceedings of the 7th International Conference on Web Information Systems and Technologies* (WEBIST-2011), pages 127-130
ISBN: 978-989-8425-51-5

subcontrollers first, so that input is processed at the lowest levels of the controller hierarchy first.

User input and output devices form an implicit fourth component of the MVC pattern. The Smalltalk system was based on a graphical display and standard user input devices, primarily a keyboard and mouse. User menus were also considered to be a type of virtual device that transmitted input to the controller hierarchy just as the keyboard or mouse did. Though menus were implemented in the GUI, they were not considered as view components.

The primary benefit of the MVC design pattern is separation of concerns and the resulting modularity. The design isolates user interface presentation from user input handling, and isolates both of these from application state and transaction processing. This makes it possible to modify or replace one component without needing to modify or even understand the others. It also facilitates extensibility by making it possible to add a view/controller pair for a new interface medium, or to add new functionality to the model independently of the other components.

3 MVC IN WEB FRAMEWORKS

ASP .Net MVC 2 is the latest version of the Microsoft web development framework (Esposito, 2010). It adds the MVC design architecture to earlier versions of ASP .Net based on Web Forms. The ASP .Net MVC 2 framework uses a single handler for HTTP requests that determines and instantiates an appropriate controller for each request. Controllers are responsible for handling incoming requests, orchestrating transaction processing by the model, preparing data for the subsequent view element, and activating that view element to generate the response. A controller class can include multiple actions that respond to different types of requests, each action being a unique public method of the class. Views are defined in ASP files, which are HTML templates with server-side scripts that can generate dynamic content. Each view receives information from the controller that activated it, either as native objects or as view-model objects, which are unique compilations of data from the model, each designed for a specific view. The model component is intended to contain application logic and database access. A differentiation is made between view models, which are data structures intended to convey information to a specific view element, and application models, which are domain entities and related transactions that operate on them. The model component also provides object-relational mapping, which hides the details of how application domain objects are mapped to database tables.

Rails is a web application framework developed for use with the Ruby programming languages (Thomas and Hansson, 2007). HTTP requests are handled in Rails through a central router that directs requests to the appropriate ActionController class and method within the controller component. ActionController objects are responsible for filtering request parameters, for orchestrating transactions by invoking methods of appropriate model elements, and for arranging the appropriate view response. ActionControllers also arrange for view elements to have access to data from the model. The controller elements are also responsible for web session management, including cookie management. The user interface (view component) is presented through dynamic document templates that are standard HTML documents with embedded Ruby scripts, similar to ASP, PHP, JSP, etc. View elements can access the model component as necessary to obtain data for presentation. The Rails model component includes elements that encapsulate application logic and transaction processing (ActiveModel), and an object relational mapping scheme (ActiveRecord) that associates each database table with a model element. The model also includes ActiveResource elements that provide access to external resources.

The Apache Struts 2 framework is based on Java 2 Enterprise Edition (J2EE) and Java Server Pages (JSP) technology. Struts2 view components are JSP documents with embedded tags that provide a variety of functionality, including flow of control (iteration, conditionals), access to Java Beans (model components), and streamlined HTML Forms construction. Controller components of a Struts2 web application are embodied in *action*s, which are Java classes. An action can respond to user input from one or more JSPs. The primary responsibilities of each action are to validate user input and to orchestrate transaction processing by invoking appropriate model operations. Actions are defined and configured through an XML configuration file or through the Java annotation mechanism. This configuration information also controls the flow of a web application by determining what view follows each action, depending upon the outcome of the action. A central part of Struts2 is the Value Stack, where information flowing between view and controller is stored and converted as needed. This

eliminates much of the detail involved in handling HTTP request parameters and in providing information for JSPs to display.

All three of these web frameworks are consistent with the MVC pattern in that user interface components (View) are separated from application logic (Model) and control functions (Controller). They differ from the original MVC pattern in several respects, however.

- No inherent view->model dependency (Observer pattern): The model component in web applications does not notify view elements of changes to the model. Rather, the controller determines view behavior, depending on the outcome of model transaction processing.
- No 1-1 view-controller correspondence: In the original MVC, each view element has a unique controller element that is defined for that view element alone.
- The Front Controller pattern: All three frameworks use this pattern, in which a single controller element is responsible for routing incoming HTTP requests, based upon the requested URL and configuration data.
- The controller specifies view dynamics: The controller decides which view follows each controller action, based upon the action outcome. This amounts to what is essentially a view=>controller dependency.
- Controller elements are responsible for data validation: Transaction parameter validation is essentially a model function. Validation logic can be pushed into the model component, but the responsibility for executing the validation function is still with the controller.
- The model is not clearly defined: All of the frameworks lack a clear definition of the Model component. It is assumed to involve application logic and data management, but there is no clear structure to define the model or to cleanly separate it from the controller.
- Model classes are instantiated on demand: Rather than a persistent model component as envisioned in the original MVC, the web frameworks instantiate model objects as needed to handle transactions and to encapsulate domain entities. The database or data persistence layer of the application can be a persistent model component, however.

These changes to the MVC pattern reflect the fundamental nature of the client-server architectural style underlying the Web. The view platform (client) is physically separated from the other components, and so the view component has become less closely tied to the controller and model in structure and operation. At the same time, the view has taken on some of the model responsibility in order to provide a more responsive user interface. The added responsibility of the controller (front controller functions and view sequencing) are necessary because of the need to explicitly manage the flow of control inherent in the user experience.

4 MVC-WEB DESIGN PATTERN

The MVC-Web design pattern described in this section is a model for how the MVC pattern is now being interpreted in web application frameworks. MVC-Web reflects the evolutionary changes that have occurred in the MVC design pattern as it has been implemented in web frameworks.

The MVC-Web model component is generally responsible for maintaining the application state. Its responsibilities include:

- Data persistence: maintain a database or an abstract database interface
- Transaction processing: execute application logic that operates on the application state
- External interface: manage interactions with external agents, such as web services or legacy systems
- Query handling: provide information to view and controller elements in response to queries.

The MVC-Web view component presents a user interface, including data presentation and input devices. Its responsibilities include:

- Information retrieval and display: present information to the user; query the model as necessary to obtain information to be displayed
- User input: present input forms and controls that allow the user to interact with the application
- Client-side dynamic behavior: provide an interactive client-side experience for the user (using JavaScript, Ajax, or other means); This may include input completion, input validation or other implementations of application-specific rules or functions that are otherwise a responsibility of the model component.

The MVC-Web controller component has three primary responsibilities.

- Front controller: receive incoming requests and route them to the appropriate handler
- Action handlers: receive request parameters; validate request parameter syntax (this may repeat validation done by view elements); orchestrate request handling by invoking appropriate model elements
- Control flow: invoke the appropriate view element as a response to the request being processed, depending upon the outcome of the action invoked.

Components of the MVC-Web pattern interact in the following ways.

- Model-view: View elements may query the model to obtain information to be displayed to the user
- Model-controller: Controller action elements call on model elements to carry out requested transactions. Model functions can include executing application logic, updating the database, or invoking services of external agents. Controller elements may request data from the model to be passed to view elements.
- Controller-view: Controller elements respond to requests originating with view elements. The controller also determines which view element will be presented to the user in response to a request. Controller elements may prepare information for use by view elements.

This version of MVC differs from the original (Smalltalk) version in several important ways:

- The Observer pattern is not used to inform the view and controller of model updates. The controller has a more prominent role instead in propagating updates.
- There is no 1-1 correspondence between view and controller elements. The relationship is many-many instead.
- The controller must be able to route requests from multiple view elements (Front Controller pattern) and to manage the flow of the application in response. The controller may pass information from model to view as part of this function.
- Some application logic (e.g., for validation or completion of input data) can be present in the view and controller components. This muddies the separation of concerns, but improves efficiency of the application by providing faster response to users.

This MVC-Web pattern is intended to reflect the current implementation of MVC in web application frameworks. The pattern is not necessarily stable, however. Evolution of the MVC-Web pattern continues, for example in Ajax-based user interfaces that are becoming richer and more responsive. Such changes may further cloud the boundaries between MVC-Web components and reduce the degree of modularity that the pattern provides.

REFERENCES

Barrett, R., Delany, S., 2004, openMVC: A Non-proprietary Component-based Framework for Web Applications, WWW2004.

Chun, L., Yanhua, W., Hanhong, L., 2003, A Novel Web Application Frame Developed by MVC, *Software Engineering Notes*, 28(2).

Esposito, D., 2010. *Programming Microsoft ASP.NET MVC*, Microsoft Press.

Fowler, M., 2003. *Patterns of Enterprise Application Architecture*, Addison-Wesley, Boston.

Gamma, E., Helm, R., Johnson, R., Vlissides, J., 1995. *Design Patterns*, Addison Wesley, Reading, MA.

Goldberg, A., Robson, D., 1985. *Smalltalk-80: the language and its implementation*, Addison-Wesley.

Krasner, G. E., Pope, S. T., 1988. A Cookbook for Using the Model-View Controller User Interface Paradigm in Smalltalk-80. *Journal of Object-Oriented Programming*, 1(3), 26-49.

Mahmoud, Q., 2003. *Servlets and JSP Pages Best Practices*, http://www.oracle.com/technetwork/articles/javase/servlets-jsp-140445.html.

Thomas, D., Hansson, D.H., 2007. *Agile Web Development with Rails*. The Pragmatic Bookshelf.