(1) name of the workshop：Workshop on Databases and Software Engineering

(2) title of the paper：A Study of Document-based MVC Design Pattern for Improve Web-based Application System Development

(3) author1 name：Shuan-Ju Chiu
current affiliation：Institute of Information Management, Chung Yuan Christian University, Taiwan
postal address：22 Pu-Jen,Pu-chung Li, Chung-Li (32023), Taiwan, R.O.C
e-mail address：gpie.tw@yahoo.com.tw
telephone number：886-927889233
fax number：886-3-4660094

(4) author2 name：Kuo-Ming Lo
current affiliation：Institute of Information Management, Chung Yuan Christian University, Taiwan
postal address：22 Pu-Jen,Pu-chung Li, Chung-Li (32023), Taiwan, R.O.C
e-mail address：gpie.tw@yahoo.com.tw
telephone number：886-955060358
fax number：886-3-4660094

(5) author3 name：Danny Lai
current affiliation：Institute of Information Management, Chung Yuan Christian University, Taiwan
postal address：22 Pu-Jen,Pu-chung Li, Chung-Li (32023), Taiwan, R.O.C
e-mail address：dannylai@seed.net.tw
telephone number：886-938108234
fax number：886-3-4660094

(6) author4 name：Yu-Liang Chi
current affiliation：Institute of Information Management, Chung Yuan Christian University, Taiwan
postal address：22 Pu-Jen,Pu-chung Li, Chung-Li (32023), Taiwan, R.O.C
e-mail address：maxchi@mis.cycu.edu.tw
telephone number：886-3-4563171 ext 5408
fax number：886-3-4660094

(7) name of the contact author：Shuan-Ju Chiu

(8) keywords： Design Patterns、MVC、Document-based MVC、XML、Web

A Study of Document-based MVC Design Pattern for Improve Web-based Application System Development

Shuan-Ju Chiu, Kuo-Ming Lo, Danny Lai, Yu-Liang Chi

Institute of Information Management, Chung Yuan Christian University, Taiwan

Abstract

This paper proposes an improvement approach for software development, named DMVC (Document-based Model-View-Controller) design pattern. DMVC, based on XML and MVC, improve flexible on modification especially when system requirement change and software maintenance.

For solving decouple and interaction problem, we add two low-level design patterns, “Abstract Factory” and “Façade”, between Model and Controller to reduce dependency. Therefore, Controller and Model can collaborate just by XML document. We also describe an application flow that is made by XML document, so the display logic of view can be adapted when workflow or requirement change. Consequently, DMVC generate an improved design pattern to assist in large application system development.

To base on the XML characteristics of interoperability and human-machine readable, the benefit of DMVC is not only improving maintainability, but more decoupling Model、View、Controller components.

Keyword: Design Patterns、MVC、Document-based MVC、XML、Web-based Application

1. Introduction

Design Patterns are well-organized development approaches that collect experiences or proven "best practices" during system development. For years, developers get advantages from using design patterns in reducing developing time and the effort of try errors. The MVC (Model-View Controller) is one of design patterns,and it is a kind of High-level Architectural Patterns[6]. The idea of MVC is usually divided into three distinct components by separating application functions, such as “Model”, “View”, and “Controller”[7]. The View is responsible for presenting data to users and forming the interface of the whole application system with the controller which process the user’s interaction. The Model represents the data of application and the business logic. The Controller is responsible to coordinate the relationship between View and Model by criteria. MVC does a good work on structured system development and modularized components; however; it is still inflexible on modification especially when system requirement change and maintenance. For solving above issue, XML is introduced into MVC model to become an Document-based MVC (DMVC). The benefit of DMVC is not only improving maintainability, but more decoupling Model、View、Controller components.

2. Document-based MVC design pattern

The name of the design pattern we propose in this paper is Document-based MVC (DMVC)design pattern.The definition of the MVC in this paper is based on the concept of the "web-based application"[1], which is different from the traditional MVC design pattern. In the traditional type of MVC design, each View needs a Controller in order to form the_user interface. However, in our research, the MVC will be applied to the "software architecture". Under each of the relative workflow or subsystem, all Views will be managed and assigned by one Controller, and the Controller will then dispatch the request to the relevant Model object.

2.1 Purpose

To provides more flexable maintainability and looser coupling relation between Model、View and Controller. We define the interrelation in DMVC components,and Controller and Model can collaborate just by XML document.

2.2 Motivation

- To develop larger-sized application systems which need long time maintenance.
- To single out the static information which easily adapt to requirement onto the XML document in order to reduce the possibility of a massive revision of the program when requirement changes.
- To easily adapt to new requirement by configuring the XML document.
- To provides a looser coupling between Model and Controller, thus to replace the

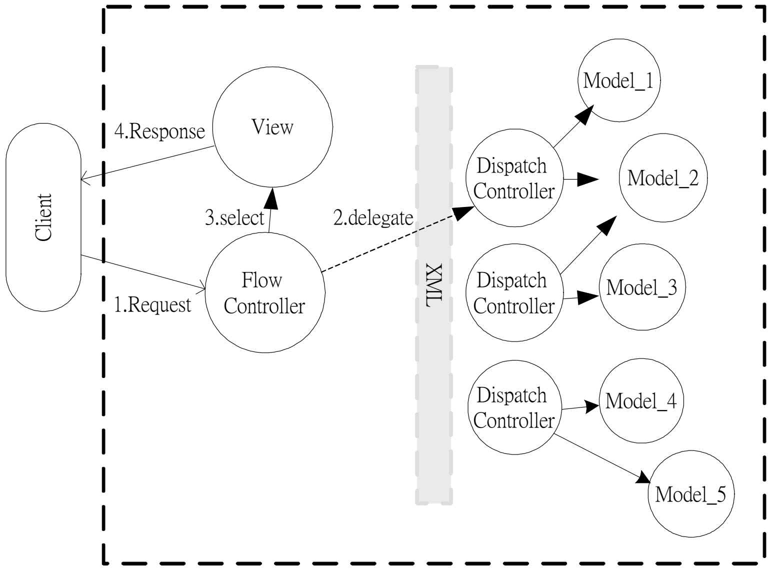


Figure 1、Document-based MVC design pattern

Model without influencing the Controller by setting the XML document.

- To inquire the status of the View/Model while the Controller is operating. This information can also be obtained from the XML document which is more readable.
- When the display flow of the View is managed by the Controller, and when these information can be configured by the XML document.

2.3 DMVC Architecture

The detail composition of the DMVC is the following,and the conceptural diagram show as Figure 1:

1.Model: In charge of the encapsulation of the application status and the relevant business rule. But the Model by itself does not know how to show the data or who should be responsible of the business rule.

2.View: Views act as an interaction interface with the users, taking charge of showing the data and receiving user requests.

3.Controller is divided into two types depending on their functionalities:

- *Flow Controller* is used to represent the application behavior and application logic. In the web-based application, Controller is focus on managing the GET/POST functions under the HTTP. Controller described what the application behavior will be and which Model should request be dispatch to. Finally, Controller decides the responsible Views to response to users. DMVC hopes to single out the static information which be easily adapt to requirement onto the XML document in order to reduce the possibility of a massive revision of the program when requirement changes. "*Flow Controller*" and "*Dispatch Controller*" form the role of the Controller in DMVC.
- *Dispatch Controller*:Façade is also one of the Design patterns [5], and provides a unified model manipulating interface to "*Flow Controller*". When the detail of Model invoking is wrapped in the "*Dispatch Controller*", as Figure 2 we can effectively reduce the level of dependency and communication relation between Controller and Model. "*Flow Controller*" and "*Dispatch Controller*" form the role of the Controller in DMVC.

4.Document for Process Flow: The dynamic information that originally belongs to Model, View or Controller. When the information is abstracted from program, the complexity of the maintenance can be reduced. Due to the human-machine readable characteristics of XML, the maintainer could understand the whole system better and easier.

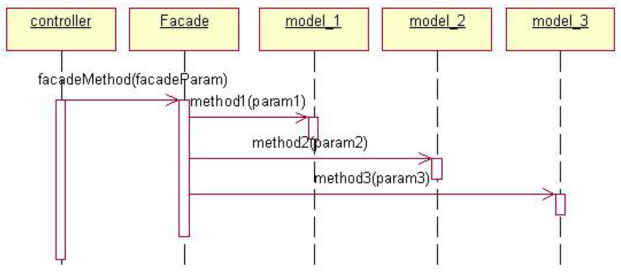


Figure 2、interaction after facade

2.4 DMVC Designing Steps

As shown in Figure 1, we are able to design a *Flow Controller* for each workflow in the application. We can also design a *Dispatch Controller* based on different procedures of the workflow. And the *Dispatch Controller* will handle the complex duty of invoking the back-end Model. Finally, we can describe the sequencing of the presentation of the Views and the information of the *Dispatch Controller* in the "ProcessFlow Document".

When users send their requests to the application, the *Flow Controller* will be responsible for managing and dispatching requests, and will make its judgment and act upon the requests according to the "ProcessFlow Document". During the handling process, if there is a need for the Model in performing the business rule, the *Flow Controller* will then assign a proper Model to handle the case based on the configuration of "ProcessFlow Document". The way of the assignment is to pass the requests to the *Dispatch Controller* so that the *Flow Controller*'s only duty is to find the proper *Dispatch Controller* information in the "ProcessFlow Document" without understanding the information of the Model. After the requests are successfully handled, the *Flow Controller* will then choose the next phase of the View component that should be response to the users according to the "ProcessFlow Document".

3. Implement Issues

We will now discuss some relevant issues and solutions in DMVC design pattern.

3.l The dependency between Controller and Model

When users make a request to a Web-based application, then it compute in several back-end Model objects. In the process of the on-line shopping cart, for example, user sends an order request and may need a Model object for storing the order data. In the same time, another checking request will ask different kind effort of processing database. If the Controller handles all Model invocation, it has to know all the implement details of all Models. Therefore, this way will results in difficulties of maintenance.

For solving the dependency between Controller and Model, we can add Façade design pattern into DMVC. Façade can provide a simple interface for the complex back-end subsystems. This interface then forms the bridge of communication between Controller and Model. The information about the sequence of invoking Model objects and the signature of Model objects will be hide in Façade without the acknowledgement of the Controller. Therefore, we can reduce the complexity of couple between Controller and Model. Consequently, DMVC divides the Controller's loading into *Flow Controller* and *Dispatch Controller* based on different situations.

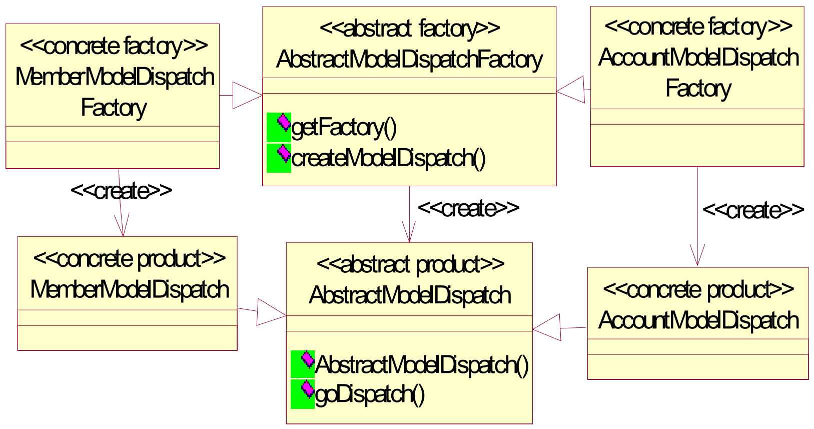


Figure 3 、 the structure of Abstract Factory

3.2 Sending Parameters between Components

As mentioned above, using Façade design pattern reduces the dependency between Controller and Model. However, we must make sure that all the parameters will be integrated at the same time. When Controller transmits a job to Façade, Controller will already gathered all the parameters before it can pass the job to Façade. Such as Fiqure 2, The FaçadeMethod() must transfer parameters to satisfy corresponding methods. In order to simplify the usage of Façade design pattern to reduce the dependency relation between Controller and Model, we can achieve this goal by taking advantage of the characteristics of the Web-based application. That is, to make use of the Request and Response object that be transmitted on the HTTP protocol between client and server. In DMVC, *Flow Controller* can pass the Request object to *Dispatch Controller*. When *Dispatch Controller* needed to call Model objects, there is enough information in Request object mostly. Besides, some of the Model objects may need several static parameters to initiate its function, such as the account/password or ODBC name for connecting the database. It is not proper to get these parameters from the Request. Therefore, we can initial a Model object by configuring the static parameters on XML document, as a result of simplifying the complexity of the maintenance of Model.

3.3 Hiding details of the assigned Model

Application will then have many specific categories of *Dispatch Controller* to implement various strategy of invoking Model objects. In order to separate the Model details from the *Flow Controller*, we need to apply the Abstract Factory design pattern [5] to create instances of matched *Dispatch Controller* object.

The Abstract Factory provides a way to create instances of those abstract classes from a corresponding set of concrete subclasses. In this situation, an abstract factory class defines methods to create an instance of each abstract class that represents a *Dispatch Controller*. Concrete factories are concrete subclasses of an abstract factory that implement its method to create instances of concrete *Dispatch Controller* classes, as shown in Figure 3. Thus, by using the Abstract Factory interface, *Flow Controller* does not need to know which or how concrete factory object creates which specific *Dispatch Controller*.

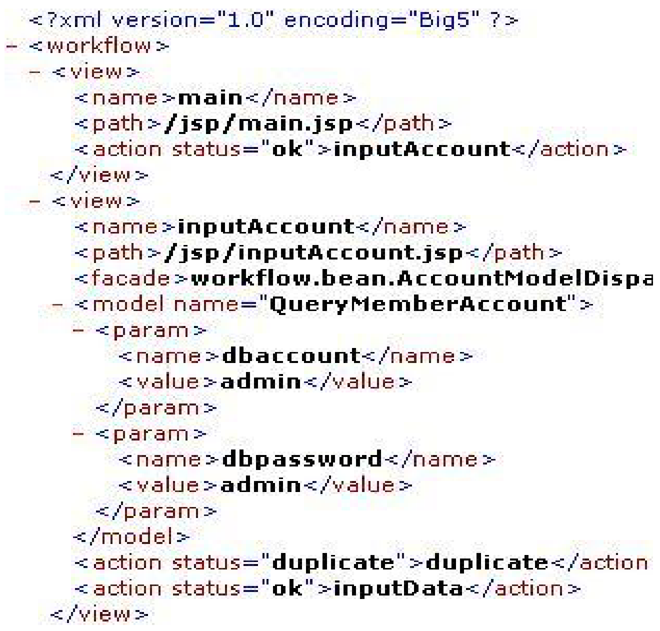
3.4 Design of the ProcessFlow Document

The Process Flow is the logic of the application process. The Controller can derive the information from this kind of XML document in order to better know how to control the system and handle demands. There is some information in ProcessFlow Document as the following:

- System Procedure Information: That is, the order of the application process. Controller will be able to tell which view is in the next procedure from this information. These procedures may be the workflow of the users from the perspective of an enterprise, and they need to be constantly adjusted according to the change of the business environment. Therefore, when we describe these procedures in XML document, Controller will be able to choose a proper View accordingly in response to users requirement.
- Model Information: Under the design of DMVC, *Flow Controller* just has to pass the request to specific *Dispatch Controller*, and delegate *Dispatch Controller* to call Model objects. Therefore, when we describe the information of a specific *Dispatch Controller* needed for each procedure in the "Process Flow Document", *Flow Controller* will analyze the information in the XML document and then use abstract factory method to create proper *Dispatch Controller* object, but *Flow Controller* doesn't know which *Dispatch Controller* had been created.
- Initiative parameter of Model: We may need some static parameters to initiate the Model objects, such as the information of the database account name and password. By putting these parameter in the "ProcessFlow Document" like Deployment Descriptor in Java Servlet [9]. We can re-deployment the Model objects without influencing Controller and View.
- Status Information: This type of information records the different reaction of a Controller under different Model status or user status. These statuses will actively change according to the application process. Controller must respond accordingly or choose a different view based upon different system status. By describing the summarized information in the XML document, Controller can then choose a responsive and proper way to handle the situation. For example, when verifying the data which users input, there will be two statuses in application: the success or failure of data verification, under this situation Controller may have to make either one of the responses: one is to continue with the next application process; the other is to go back to the previous

application process that requires the users to put in accuracy information.

We can describe this workflow process by the XML method. Please see the figure 4 as followed:



```
  <?xml version="1.0" encoding="Big5" ?>
- <workflow>
  - <view>
      <name>main</name>
      <path>/jsp/main.jsp</path>
      <action status="ok">inputAccount</action>
    </view>
  - <view>
      <name>inputAccount</name>
      <path>/jsp/inputAccount.jsp</path>
      <facade>workflow.bean.AccountModelDispa
    - <model name="QueryMemberAccount">
      - <param>
          <name>dbaccount</name>
          <value>admin</value>
        </param>
      - <param>
          <name>dbpassword</name>
          <value>admin</value>
        </param>
      </model>
      <action status="duplicate">duplicate</action
      <action status="ok">inputData</action>
    </view>
```

Figure 4、example of ProcessFlow Document

3.5 Implement Consequences

1. Clearly distinguish the business rule from the requests processing.
2. Clearly define the job of a Web Page designer and that of a programmer.
3. Decoupling the Controller and Model. Developer can simply replacing or revising the Model through the configuration of XML document.
4. Increase the maintainability by dividing the Controller's role into two parts: Flow Controller and *Dispatch Controller*.
5. When the Model changes, we only need to revise or update the *Dispatch Controller* without influencing the View and the Flow Controller.

As a result of XML document, the application process can be flexibly adapted to the change of the workflow.

4.Conclusion

In order to facilitate testability and improve maintainability, developer must establish a extended software architecture by modular development method. In this way, maintainer also can handle all situation quickly and maintain effectively.

This paper proposes a document-based MVC design pattern to improve Web-based application development. MVC separates application functions into three distinct components, clearly decoupling data presentation, data representation and application operations. It's also a good way to develop a Web-based application. As a result of the XML ability of interoperability and human-machine readable, we can separate the information of application process into the XML document. By this way, it not only improves maintainability, but more decoupling Model、View、Controller components.

In DMVC, we using "Abstract Factory" and "Façade" design pattern in the interaction between Model and Controller to reduce the dependency. Therefore, Controller and Model can collaborate just by XML document. Besides we describe the information of process flow by XML document, so the logic of view display can be flexibility adapt when workflow or requirement change. Moreover, the view design is independent of application logic.

We should consider the system requirement when apply design pattern to develop application. DMVC design pattern is suitable for application with a long lifetime. And it also increases the degree of complex and reduces the efficiency when developing small size application. Future work should aim at establishing the validity of these idea by applying them to new application. It would also be valuable to apply DMVC idea to develop framework.

Additionally, this paper describes how to establish a scalable, manageable software architecture using DMVC. By these characteristics, it's maybe suitable to develop user-driven Web Services using DMVC architecture. And the details of implementation are good issue to discuss in Future.

5.Reference

1. Avraham Leff, James T Rayfield , "*Web-Application Development Using the Model/View/Controller Design Pattern*" , Enterprise Distributed Object Computing Conference, 2001. Proceedings. Fifth IEEE International , 2001.
2. Barracuda:Open source Presentation Framework, http://barracuda.enhydra.org/,2001.
3. Carla Sadtler et al., "Patterns for e-business: User-to-Business Patterns for Topology 1 and 2 using WebSphere Advanced Edition," IBM Redbooks publication, 2000.
4. Enhydra XMLC 2.1 , http://xmlc.enhydra.org/.
5. E. Gamma, R. Helm, R. Johnson, and J. Vlissides, "*Design Patterns:*Elements of Reusable Object-Oriented Software", Addison-Wesley,1995.
6. F. Buschmann, H. Rohnert, M. Stal, P. Sommerlad and R. Meunier, "Pattern-Oriented Software Architecture: A System of Patterns," John Wiley & Sons, 1996.,123-168.
7. G.E. Krasner and S.T. Pope, "A Cookbook for Using the Model-View-Controller User Interface Paradigm in Smalltalk-80," 26-49, Journal of Object-Oriented Programming, Auguest/September, 1988.
8. Inderjeet Singh, Beth Stearns,Mark Johnson, and the Enterprise Team, "Designing Enterprise Applications with the J2EE Platform, Second Edition," Addison-Wesley, March 2002 ,Ch 4.4.
9. Java Servlet Technology Implementations & Specification, http://java.sun.com/products/servlet/download.html#specs , 2001.
10. M.J Mahemoff, L.J Johnston, "Handling multiple domain objects with Model-View-Controller**,**" Technology of Object-Oriented Languages and Systems, 1999. TOOLS 32. Proceedings , 1999 ,Page(s): 28 -39
11. Michael Ball, "Dispatcher eases workflow implementation", http://JavaWorld.com, October 19 2001
12. S.Bodoff, D.Green, Khaase, E.Jendrock,

M,Pawlan, B.Strearns, “The J2EE Tutorial,” Copyright 2002, Addison-Wesley, “http://java.sun.com/j2ee/tutorial”.

13. Simon Bennett, Steve McRobb, Ray Farmer, “Object-Oriented System Analysis and Design Using UML,” McGRAW-HILL International Editions, 1999.
14. Struts,http://jakarta.apache.org/struts/index .html, 2001
15. WebWork, http://sourceforge.net/projects/webwork, 2001
16. World-Wide Web Consortium, *An introduction to XSL*, 1998,http://www.w3c.org/Style/XSL.
17. World-Wide Web Consortium, *XML 1.0*, February 1998,http://www.w3.org/XML.