CS 635 Advanced Object-Oriented Design & Programming
Spring Semester, 2004
Doc 16 Model-View-Controller

Contents

Model-View-Controller (MVC) ................................................................. 2
Template View .......................................................................................... 6
Page Controller ...................................................................................... 11
Front Controller ................................................................................... 14
Intercepting Filter .................................................................................. 16
Composite View ..................................................................................... 17

References

Pattern-Oriented Software Architecture: A System of Patterns v1, Buschman, Meunier, Rohnert, Sommerlad, Stal, 1996, pp 125-143

Patterns of Enterprise Application Architecture, Folwer, 2003, pp 330-386

Core J2EE Patterns: Best Practices and Design Strategies, 2nd, Alur, Crupi, Malks, 2003

Copyright ©, All rights reserved. 2004 SDSU & Roger Whitney, 5500 Campanile Drive, San Diego, CA 92182-7700 USA. OpenContent (http://www.opencontent.org/opl.shtml) license defines the copyright on this document.
Model-View-Controller (MVC)

Context

Interactive application with human-computer interface

Forces

• Same data may be displayed differently
• Display & application must reflect data changes immediately
• UI changes should be easy and even possible at runtime
• Changing look & feel or port to other platforms should not affect core application code

Solution

Divide application into three parts:

• Model (core application)
• View (display, output)
• Controller (user input)
Model

Core application code

Contains a list of observers (view or controller)

Has a broadcast mechanism to inform views of a change

Same mechanism as subject in Observer pattern

View

Displays information to user

Obtains data from model

Each view has a controller

Controller

Handles input from user as events

• Keystrokes
• Mouse clicks
• Mouse movements

Maps each event to proper action on model and/or view

Many people misinterpret what a controller does
Model

- CoreData
- observers

- attach(Observer)
- detach(Observer)
- notify

- getData
- service

Controller

- myModel
- myView

- initialize(Model,View)
- handleEvent
- update

- call service

View

- myModel
- myController

- initialize(Model)
- makeController
- activate
- display
- update

Observer

- update

Observer updates are propagated to the controller, which manipulates the model and displays the view. The controller handles events and updates the model. The model provides data and services for the view and observer.
View & Controller

Make up the user interface

Some GUI frameworks combine these

VW Smalltalk contains both, but hides controller from programmer

<table>
<thead>
<tr>
<th>Some Existing Smalltalk Controllers &amp; Views</th>
</tr>
</thead>
<tbody>
<tr>
<td>Controllers</td>
</tr>
<tr>
<td>--------------</td>
</tr>
<tr>
<td>ApplicationDialogController</td>
</tr>
<tr>
<td>BasicButtonController</td>
</tr>
<tr>
<td>ClickWidgetController</td>
</tr>
<tr>
<td>ColoredAreaController</td>
</tr>
<tr>
<td>ComboBoxButtonController</td>
</tr>
<tr>
<td>ComboBoxInputBoxController</td>
</tr>
<tr>
<td>ComboBoxListController</td>
</tr>
<tr>
<td>ControllerWithMenu</td>
</tr>
<tr>
<td>ControllerWithSelectMenu</td>
</tr>
<tr>
<td>DataSetController</td>
</tr>
<tr>
<td>DataSetControllerProxy</td>
</tr>
<tr>
<td>DelayingWidgetController</td>
</tr>
<tr>
<td>DrawingController</td>
</tr>
<tr>
<td>DropDownListController</td>
</tr>
<tr>
<td>EmulatedDataSetController</td>
</tr>
<tr>
<td>EmulatedSequenceController</td>
</tr>
<tr>
<td>EmulationScrollBarController</td>
</tr>
<tr>
<td>HierarchicalSequenceController</td>
</tr>
<tr>
<td>InputBoxController</td>
</tr>
</tbody>
</table>
Template View

Renders information into HTML by embedding markers in an HTML page

Server Pages

Java

<html>
<body>
<%! int x = 1; %>
<%! int y = 2; %>
If we add <%= x %> to <%= y %> we will get <%= x + y %>
</body>
</html>

Smalltalk

<HTML>
<BODY>
<%
    x := 1.
    y := 2.
%>
<p>
When we add <%= x %> to <%= y %> we get <%= x + y %>
</p>
</BODY>
</HTML>
SDSU Template Class

Markers
- Treated as variables
- Separated by `@@@`

### Sum.html

```html
<HTML>
<BODY>
<p>When we add `@@@x@@@` to `@@@y@@@` we get `@@@sum@@@`</p>
</BODY>
</HTML>
```

### Sample Use

template := Template fromFile: 'Sum.html'.
template replace: 'x' with: '1' printString.
template replace: 'y' with: '2' printString.
template replace: 'sum' with: '3' printString.
template asString

### Result

```html
<HTML>
<BODY>
<p>When we add 1 to 2 we get 3</p>
</BODY>
</HTML>
```
**Advantage**

Graphic designers can generate view

**Disadvantages**

Poor module structure

Leads to mixing model, controller and view logic

Leads to repeated code in files

Many programming tools do not work on template files

**Some common Issues**

Conditional display

```html
<p>Please pay your bill
<If user.isDeadBeat()> <B> </IF>
now.
<IF use.isDeadBeat()> </B> </IF>
```

Iteration over collection

- Given a list create a drop down menu

Use View Helper to separate out processing logic
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class HelloWorld extends HttpServlet {

    public void doGet(HttpServletRequest request,
            HttpServletResponse response)
            throws IOException, ServletException
    {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println("<html>");
        out.println("<body>");
        out.println("<head>");
        out.println("<title>Hello World!</title>");
        out.println("</head>");
        out.println("<body>");
        out.println("<h1>Hello World!</h1>");
        out.println("</body>");
        out.println("</html>");
    }
}
Smalltalk
Smalltalk.VisualWave defineClass: #VeryBasicServlet
   superclass: #{VisualWave.HttpServlet}
   indexedType: #none
   private: false
   instanceVariableNames: ""
   classInstanceVariableNames: ""
   imports: ""
   category: 'Web Toolkit-Servlet'

VisualWave.VeryBasicServlet methodsFor: 'serving'

doGet: aRequest response: aResponse

   aResponse write: '<HTML><BODY>GET<BR>Hello world</BODY></HTML>'.

doPost: aRequest response: aResponse

   aResponse write: '<HTML><BODY>POST<BR>Hello world</BODY></HTML>'. 
Page Controller

An object that handles a request for a specific page or action on a Web page

- Decodes URL
- Extracts all form data and gets all data for the action
- Create and invoke model objects, pass all relevant data to model
- Determine which view should display the result page and forward model information to it

Each page or url on the site has a different page controller
public class AlbumController extends ActionServlet {

    public void doGet(HttpServletRequest request, HttpServletResponse response) throws IOException, ServletException {

        Album requestedAlbum = Album.find(request.getParameter("id"));
        if (requestedAlbum == null) {
            forward("missingAlbumError.jsp", request, response);
            return;
        }

        request.setAttribute("helper", requestedAlbum);

        if (requestedAlbum.isClassical())
            forward( "/classicalAlbum.jsp", request, response);
        else if (requestedAlbum.isRock())
            forward( "/rockAlbum.jsp", request, response);
        else
            forward("/album.jsp", request, response);
    }
}
Parent Class of All Page Controllers

public class ActionServlet extends HttpServlet {

    protected void forward(String target, HttpServletRequest request,
                            HttpServletResponse response) throws IOException, ServletException {
        RequestDispatcher dispatcher =
            getServletContext().getRequestDispatcher(target);
        dispatcher.forward(request, response);
    }
}
Front Controller

A controller that handles all requests from a Web site

**Forces**

- Avoid duplicate control logic
- Apply common logic to multiple requests
- Separate system processing logic from view
- Have a centralized controlled access point into system
How it works

All requests to a Web site are directed to the FrontController

The FrontController

• Examines the URL & form data
• Determines the correct command to handle the request
• Create the correct command
• Forwards the request to the command

Command is part of controller so it uses a separate view

Disadvantage

• More complex than Page Controller

Advantages

• Only one controller has to be configured into the web server
• A command object handles only one request so command does not have to be thread-safe
• Commands can be added dynamically (if controller uses reflection to create a command object)
• Factor out common code from multiple Page Controllers
**Intercepting Filter**

You want to manipulate a request and a response before and after the request is processed.

**Forces**

You want:

- Common processing across requests like:
  - Logging
  - Compressing
  - Data encoding

- Pre & post processing components loosely coupled with core request-handling services

- Pre & post processing components independent of each other

**Solution**

Add a chain of decorators (filters) that end on the Front Controller
Composite View

Build a view from atomic components while managing content and layout independently.

Forces

• You want subview, such as headers, footers and tables reused in different pages.

• You want to avoid directly embedding and duplicating subviews in multiple pages.

• You have content in subviews that frequently change or are subject to access control.

Solution

Use the composite pattern on views.

A page then is created as a composite object of views.