Model-View-Controller Architecture in a Systems Analysis and Design Course

Robert F. Zant
Illinois State University
Normal, IL 61790, USA
rfzant@ilstu.edu

Abstract

Information systems programs typically include a system analysis and design course that requires students to develop a system for either a real or simulated firm. This is inherently a less structured task than students have confronted in other courses. The Model-View-Controller (MVC) paradigm has proven to be very useful in industry and also can be effectively used in student projects. It is applicable to projects using either traditional or object-oriented methodologies. The MVC paradigm is presented along with guidelines for its use in a systems analysis and design course. A simple example of the code structure using a procedural scripting language is given.

Keywords: System Analysis and Design, MVC, Framework

1. INTRODUCTION

The IS 2002 Model Curriculum includes an example implementation consisting of ten courses that could be used to deliver the curriculum (Gorgone, 2002). These ten courses include a three-course sequence in systems development and database design (courses 7, 8, and 9). A recent survey of 50 schools indicates that this is a common pattern used in curricula--48 schools reported offering a database design course, 44 offered one course in systems analysis and design, and 25 reported offering a second course (Waldman, 2005).

The content of each of the two courses in systems analysis and design varies; but, commonly, the first course emphasizes analysis and design using cases and the second emphasizes design and implementation (Chen, 2005; Morien, 2005; Roggio, 2005). The second course, typically, is a capstone course requiring either a simulated project or a real project. Both approaches offer benefits. A simulated project allows more control over the technology used to implement the project, does not have the issues associated with working with a real client, and, consequently, allows more focus on the process. An actual project provides more fidelity in accessing feasibility and in determining requirements (Chen, 2005; Helwig, 2005).

The systems development course sequence at Illinois State University is a three-course sequence that includes both a simulated project and an actual project. The first course covers systems analysis and design using cases. Both the traditional structured systems methodology and the object-oriented methodology are covered using a text such as *Systems Analysis and Design in a Changing World* by Satzinger, Jackson and Burd (Satzinger, 2004). For the second course, students take one of two courses depending on the track in the curriculum the student has selected. One track is oriented towards traditional methodology so the second course taken by students in that track builds on the introduction to traditional methodology from the first course. The second track is oriented toward web application development so the second course in that track goes more in depth into object-oriented method-

ology. Both of these courses require students to implement a simulated project. The third course is a capstone course requiring student teams to develop a system for a real client. Completing an actual project and interacting with an actual client is an important experiential learning activity to prepare students for careers as an IS/IT professional (McGann, 2005; Morien, 2005; Rebhun, 2005; Scott, 2004].

Since students from both tracks enroll together in the capstone course and may be on the same team, the use of an implementation methodology in all three courses that can be used in both the traditional and OO approaches is advantageous. It facilitates both the students' interaction on the project and the instructor's mentoring process.

This paper describes a system design strategy that may be used in either a simulated or real project. It may be used with both procedural-oriented and object-oriented languages. Consequently, the strategy, known as the model-view-controller (MVC) paradigm, is well suited for use in a multi-course systems analysis and design sequence that includes both traditional and object-oriented methodologies and the implementation of a project. An example of its use with a procedural-oriented script language is given below.

2. MODEL-VIEW-CONTROLLER PARADIGM

The MVC paradigm dates to the late 1970's when it was developed in conjunction with the Smalltalk-80 programming language as a means of solving problems arising from developing systems with graphical user interfaces (GUI) (Krasner, 1988). The paradigm was based on the "input-processing-output" view of a system with the goal of separating a system into three parts. The *Controller* handles the input portion controlling the interface with keyboard and mouse, as well as controlling the interface between the *Model* and associated *Views*. The *Model* contains the application logic and accesses persistent data. A *View* is responsible for displaying output created by the *Model*. The structure of a simple MVC system is depicted in Figure 1 in the Appendix.

Since its development, MVC has proven to be an important design pattern for facilitating the development, debugging, and maintenance of systems. And, while it was originally intended to more easily develop GUI systems, it has been successfully applied to the development of client-server and web-based systems as well. For example, it is the basis for the JSP "model 2" architecture (Seshadri, 1999). Its usefulness has been extended by the advent of software that enhances the de-coupling of the View from the Model and the Model from the data store. Tag libraries used with scripting languages like JSP and PHP or the use of XML and XSLT provide a degree of independence between a View and Model. Models and databases can be de-coupled through the use of ODBC or JDBC drivers, the PHP Data Objects Interface (PDO) (PDO, 2006), and object-relational mapping packages (ORM) such as Hibernate (Hibernate, 2006).

There are many advantages to be gained by the use of the MVC paradigm (Parr, 2004). In fact, a study by IBM of WebSphere "Best Practices" places the use of the MVC paradigm at the top of the list (Brown, 2004). Among the most noted advantages, development and maintenance are facilitated by the separation of components that are programming oriented (Controllers and Models) from components that are design oriented (Views). This allows for the better matching of skills to tasks on project teams (Brown, 2004; Kojarski, 2003; Parr, 2004). The factoring of the system also aids in debugging allowing the causes of undesirable behavior to be more quickly isolated, and then resolved without producing unintended system-wide effects. The separation of components in the MVC approach also allows multiple Views to be easily associated with the same Model. Thus, a Model's results can be displayed in multiple formats (screen, print, PDF) or in multiple languages.

Of course, the components of a system cannot be completely de-coupled or they would cease to be a system. Since "models and views are intrinsically coupled," we seek only to "de-couple them as much as possible" (Hanson, 2005). The problem is that it is not always easy to decide where to draw the line between related components (Parr, 2004). For example, user prompts might rightly be considered a part of user input

action and, hence, part of the controller function. But, often prompt screens contain persistent data generated by a Model, such as when displaying a record for update. This would place the screen in the scope of a View. Clearly guidelines are needed, particularly for novices learning to develop systems.

3. GUIDELINES

Controllers are responsible for system-level control and navigation to the appropriate Model and View component. These responsibilities are often divided between one (or a few) Front Controller and many Page Controllers. The Front Controller accepts all input from the user (except perhaps the log on response that may require special handling), sets system variables, checks security, and invokes the appropriate Page Controller. More than one Front Controller is used in distributed sites or in otherwise segmented sites such as a site with a secure and an unsecured area.

Each Page Controller is related to one Model and to one View component. Page Controllers are typically "lightweight," simply invoking the appropriate Model and then View component. A Page controller should contain no application logic. It may only contain logic required to Invoke an appropriate Model and View.

A Model contains the application logic for a page including data access for persistent data. The Model's results are made available to a View through some intermediate device, e.g., an XML file or an array object. Since a Model may be invoked by many different Page Controllers, the results may be presented by different Views (screen, print, PDF, etc.). A View is related to only one Page Controller. It accesses data created by the Model and creates the source for presentation (XHTML, PDF, text, etc.).

These guidelines may be summarized for students as follows:

- All responses from users are processed first by a Front Controller (except the log on response)
- A Front Controller invokes a Page Controller, not a Model or View
- A Page Controller invokes one Model and one View
- A Model executes application logic and accesses data stores (contains no HTML)
- A Model creates an XML file or object containing its results
- A View creates a presentation stream
- A View contains no application logic
- A View obtains all non-constant text data from the XML file or result object produced by the Model
- A View does not directly reference any data in a Model or URL for the site
- Communication of user responses to the Front Controller is by name (e.g., field names on HTML forms)
- Communication of Model results to a View are by name (e.g., in XML DTD)

4. FRAMEWORKS

The MVC architecture is often implemented with a software framework such as WebSphere, Struts, or Spring. Frameworks provide structure by enforcing naming conventions (directories, files) and rules for constructing a system (Shiflett, 2006). They also provide components that aid in the construction of a system. For example, Spring contains over 1500 classes that may be used for such functions as transaction management or database access.

MVC is also often implemented without a framework, but you must still create naming conventions and follow certain rules in order to maintain the de-coupling that is the goal of the approach. This framework-less or lean approach is recommended by Rasmus Lerdorf, the Infrastructure Architect at Yahoo! and creator of the PHP language (Lerdorf, 2006). He notes that frameworks, in order to be widely applicable, tend to provide many functions that are not needed for each system. This can actually introduce overhead and complicate the code for small systems. For student use, frameworks introduce a steep learning curve into a course. It has been noted by others that the use of complex software in a systems analysis and design course can detract from the core concepts (Luce, 2005; Hanson, 2005). For this reason, frameworks are not used in the courses described in this paper.

5. MVC EXAMPLE

The MVC paradigm can be used with procedural as well as object-oriented languages. Many examples that apply MVC--using languages such as PHP, Python, Perl, and Java, can be found in the literature (Davies, 2004; Hanson, 2005; Lerdorf, 2006; Seshadri, 1999; Shiflett, 2006). In the courses sited herein, one uses a procedural script language, ODB Script (ODB Script, 2006), and the other uses Java. Figure 2 in the Appendix depicts the data flow in the example system. Figures 3 through 7 in the Appendix are examples of MVC components for the system written in ODB Script.

Figure 3 shows the Front Controller. A DEFAULT command is used to provide default values for system variables. Then the SESSION statement checks to see if there is an active session (i.e., if there is a cookie) and, if not, a log on script is invoked. Finally, the Page Controller specified in the URL is invoked. Only a Page Controller can be invoked since the "c/" directory and the extension ".c" are added to the controller name passed in the URL.

The example Page Controller in Figure 4 is very simple, as controllers should be. It just invokes a specific Model and then a specific View. The invoked Model, shown in Figure 5, contains the most programming logic. It retrieves rows from a database and then creates an XML file for the View to use in producing an XHTML stream.

The View component in Figure 6 combines some "boilerplate" XHTML with other XHTML statements produced by using an XSLT file to transform the XML file created by the Model. A portion of the XSLT file is shown in Figure 7.

6. SUMMARY

Information systems programs typically include a system analysis and design course that requires students to develop a system for either a real or simulated firm. This is inherently a less structured task than students have confronted in other courses. The MVC paradigm has proven to be very useful in industry and also can be effectively used in student projects with both traditional and object-oriented methodologies. The MVC architecture provides a structure that provides a guide for students to develop factored systems that facilitate coding, debugging, and maintenance.

7. REFERENCES

Brown , Ken, Keys Botzum, and Ruth Willenborg, (2004) "The Top 10 (more or less) J2EE Best Practices." IBM WebSphere Developer Technical Journal, May 12, 2004

Chen, Brady, (2005) "Teaching Systems Analysis and Design: Bring the Real World into the Classroom." ISECON 2005, Columbus, OH, October 8, 2005

Cooper, Peter, "Model, View, Controller HowTo." Retrieved June 2, 2006 from http://www.bigbold.com/snipets/posts/show/1050

Davies, Trento (2004), "Mojavi – An MVC Framework for PHP." June 2, 2004, http://ad.hominem.org/log/2004/06/ tutorial_on_mojavi.php

Gorgone, John T., et. al., (2002) "IS 2002 Model Curriculum and Guidelines for Undergraduate Degree Programs in Information Systems." Association for Computing Machinery, Association for Information Systems, and Association of Information Technology Professionals, 2002

Hanson, Stuart and Timothy V. Fossum, (2005) "Refactoring Model-View Controller." Journal of Small Systems Computing, 2005

Helwig, Janet, (2005) "Using a 'Real' Systems Development Project to Enrich a Systems Analysis and Design Course." ISECON 2005, Columbus, OH, October 8, 2005

Hibernate, Retrieved June 2, 2006 from http://www.hibernate.org/

Kojarski, Sergei, and David H. Lorenz, (2003) "Domain Driven Web Development with WebJinn." OOPSLA, Anaheim, CA, October 26-30, 2003

Krasner, Glenn E. and Stephen T. Pope, (1988) "A Description of the Model-View-Controller User Interface Paradigm in the Smalltalk-80 System." ParcPlace Systems, 1988

Lerdorf, Ramus (2006), "The No-framework PHP MVC Framework." February 27, 2006, http://toys.lerdorf.com/categories/9-PHP

Luce, Thom, (2005) "Moving the Senior Development Class from Web Development to Life Cycle Development." Issues in Information Systems, Vol. VI, No. 1, 2005

McGann, Sean T. and Matthew A. Cahill, (2005) "Pulling It All Together: An IS Capstone course for the 21st Century." Issues in Information Systems, Vol. VI, No. 1, 2005

Morien, Roy, (2005) "Student Experence of Using Agile Development Methods in Industrial Experience Projects." ISECON 2005, Columbus, OH, October 8, 2005

ODB Script, Retrieved June 2, 2006 from http://www.odbscript.com/

Parr, Terence John, (2004) "Enforcing Strict Model-View Separation in Template Engines." Proceedings of the 13th International World Wide Web Conference, New York, NY, May 17-20, 2004

PDO-PHP Data Objects Interface, Retrieved June 2, 2006 from http://pecl.php.net/package/PDO

Rebhun, Herb and Shohreh Hashemi, (2005) "Systems Development Projects – Gaining Practical Experience While Meeting Community Needs: A Win-Win State of Affairs." ISECON 2005, Columbus, OH, October 8, 2005

Roggio, Robert F., (2005) "Robust Software Development: A Technical Approach Using the Rational Unified Process." ISECON 2005, Columbus, OH, October 8, 2005

Satzinger, J. W., Jackson, R.B. and Burd, S.D., (2004) Systems Analysis and Design in a Changing World, 3rd Edition. Thomson Course Technology

Scott, Elsje, "Systems Development Group Project: A Real World Experience." ISECON 2004, Newport, RI, 2004

Seshadri, Govind, (1999) "Understanding JavaServer Pages Model 2 Architecture." JavaWorld, December 1999, http://www.javaworld.com/ javaworld/jw-12-1999/jw-12-ssj-jspmvc.html

Sheidlower, Jesse, (2005) "Catalyst." http://www/perl.com/lpt/a/2005/06/02/catalyst.html, June 2, 2005

Shiflett, Chris, "Zend Framework Tutorial." (2006) Retrieved June 1, 2006 from http://hades.phparch.com/ceres/ public/article/index.php/art::zend_framework::tutorial

Waldman, Marc, Mehmet Ulema, and Kyungsub Steve Choi, (2005) "An Analysis of IS 2002 Compliance in Selected US Business Schools." ISECON 2005, Columbus, OH, October 8, 2005

Appendix

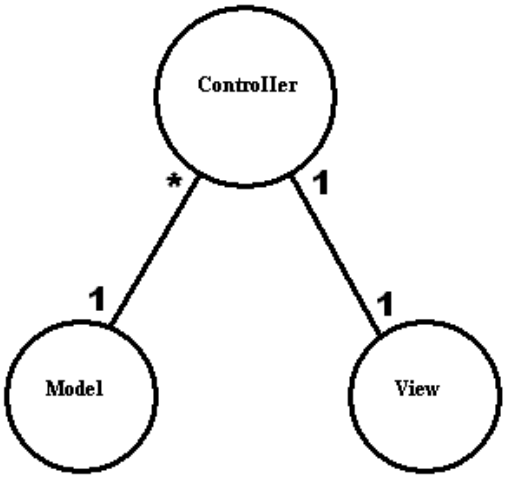

Figure 1. MVC Structure

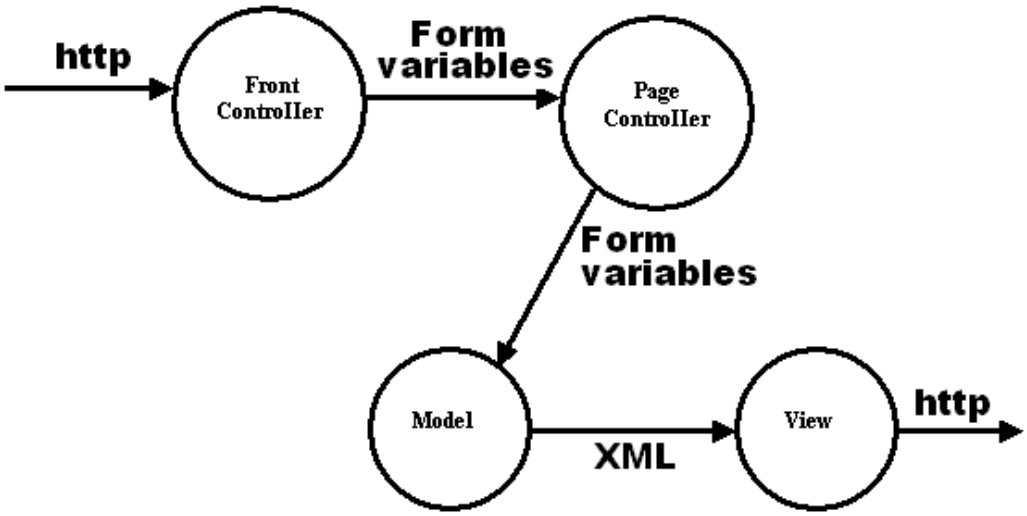

Figure 2. Data Flow in Example MVC System

```
<% NOTE: This is the Main Front Controller.
         It MUST reside in the application's root directory.
         It sets system variables, checks for log-in, and
         then loads the requested Page Controller. The default
         Page Controller is named 'menu'.

         URL Form: http://site_url/index.odb?pc=name

         Primary Directories:
             controller  = "c/"
             model       = "m/"
             view        = "v/"

         The variable 'home' must link to this controller page. ;

   DEFAULT pc       = "menu",
           sys_base = $path_Translated_dir$"/",
           url_base = http:"//"$server_name$$path_info_dir$"/",
           home     = $url_base$index.odb ;
   NOTE: If there is no active session, invoke login.odb ;
     SESSION LOGIN = "login.odb?from=$pc$", TIMEOUT = 10 ;
   INCLUDE $sys_base$"c/"$pc$".c"; NOTE: invoke page controller ;
%>
```

Figure 3. Example Front Controller

```
<%  NOTE: Page Controller for List of Products;
    INCLUDE $sys_base$m/Products/table1.m ;  NOTE: invoke Model ;
    INCLUDE $sys_base$v/Products/table1.v ;  NOTE: invoke View ;
%>
```

Figure 4. Example Page Controller

```
<% NOTE: Model for List of Products,

   Set up ODBC linkage and retrieve rows ;
     DATABASE "DSN=myProducts" ;
     SELECT Category, ProductID, Heading,
            Description, UnitPrice, UnitsOnHand
            FROM Products ORDER BY ProductID ;

   NOTE: Now create the XML file ;
     TRANSLATE UnitsOnHand 0 = "Sold Out" ;
     OUTPUT $xmlfile$ ;
     INCLUDE $sys_base$common/XML_begin.incl ;
%>
   <links>
     <home>$home$</home>
   </links>
   <products>
<%   EACHROW  %>
       <row>
         <ProductID>$ProductID$</ProductID>
         <Category>$Category$</Category>
         <Description>$Description$</Description>
         <UnitPrice>$UnitPrice$</UnitPrice>
         <UnitsOnHand>$UnitsOnHand$</UnitsOnHand>
       </row>
<%   ENDROW  %>
   </products>
<% INCLUDE $sys_base$common/XML_end.incl ;
   OUTPUT ;  NOTE: XML file completed, close it ;
%>
```

Figure 5. Example Model module

```
<% NOTE: View for List of Products,
               This module creates an XHTML page ;

   INCLUDE $sys_base$common/XHTML_begin.incl ;

   NOTE: Transform XML with PHP script ;
      SET xsl = $sys_base$"v/Products/xslt/table1.xslt" ;
      SET xml = $xmlfile$ ;
      HTTPGET $url_base$v/xslt.php, xsl, xml ;

   INCLUDE $sys_base$common/XHTML_end.incl ;
%>
```

Figure 6. Example View module

```
<!-- XML Transform for List of Products -->
<xsl:template match="/">  <!-- root template -->
  <div id="pageHeader">
    <a><xsl:attribute name="href">
          <xsl:value-of select="root/links/home"/>
       </xsl:attribute>
       <h1>World Wide Widgets</h1>
    </a>
  </div>   <!-- End of Page Header  -->
  <div id="content">
    <xsl:apply-templates select="root/products" />
  </div>   <!-- End of Content     -->
  <div id="pageFooter">
    [<a><xsl:attribute name="href">
          <xsl:value-of select="root/links/home"/>
        </xsl:attribute>Continue...
    </a>]
  </div>   <!-- End of Page Footer -->
</xsl:template>  <!-- End of root template -->
```

Figure 7. A Portion of XSLT file