openMVC: A Non-proprietary Component-based Framework for Web Applications

Ronan Barrett
Dublin City University
School of Computing
Dublin 9, Ireland
+353 1 700 5616
rbarrett@computing.dcu.ie

Sarah Jane Delany
Dublin Institute of Technology
School of Computing
Dublin 8, Ireland
+353 1 402 4936
sarahjane.delany@dit.ie

ABSTRACT
The lack of standardised approaches in the development of web-based systems is an ongoing issue for the developers of commercial software. To address this issue we proposes a hybrid development framework for web-based solutions that combines much of the best attributes of existing frameworks but utilises open, standardised W3C technologies where possible. This framework called openMVC is an evolution of the Model-View-Controller (MVC) pattern. An implementation of openMVC has been built over a 5-tier architecture using Java and .NET.

Categories and Subject Descriptors
D.2.11 [**Software Engineering**]: Software Architectures – *Patterns;* D.2.12 [**Software Engineering**]: Interoperability – *Distributed Objects;* D.2.13 [**Software Engineering**]: Reusable Software – *Reuse Models*; D.2.7 [**Software Engineering**]: Distribution, Maintenance, and Enhancement – *Portability*;

General Terms
Design, Reliability, Standardization.

Keywords
MVC, Patterns, Frameworks, Web Services, XML, XSLT, XML Schema, W3C.

1. INTRODUCTION
The lack of standardised development approaches to building web-based solutions in many organisations typically leads to poor maintainability. An associated problem is the numerous web technologies currently available which are proprietary leading to vendor lock-in to proprietary software.

Software patterns and frameworks provide ready made solutions to common problems that can be reused again and again. The Model-View-Controller (MVC) [1,2] pattern is one of the oldest design patterns and is also perhaps the best known. MVC logically splits applications into a triad of independent components or object roles. A number of web technologies and frameworks have implemented MVC such as Struts [3] and Maverick [4].

This paper presents openMVC, an MVC based framework that will promote a standardised component-based development approach for web-based systems and will utilise non-proprietary interoperable technologies to avoid vendor lock-in.

This paper is organized as follows: In Section 2, the openMVC framework is introduced. Section 2.1, the framework architecture is explored at the logical level. Section 2.2, the components of the framework are introduced. Section 3, an implementation of the framework is discussed. Section 4 summarises our conclusions.

WWW 2004, May 17-22, 2004, New York, New York, USA.
ACM 1-58113-912-8/04/0005.

2. openMVC FRAMEWORK
The MVC pattern provides an ideal starting point for discussion of the openMVC framework. The openMVC framework combines a number of the approaches taken by existing framework implementations like Struts and Maverick to produce a hybrid MVC implementation.

We extend MVC beyond the separation of presentation information, controlling logic and business logic concerns. We present an architecture where the style information (fonts, colours etc.), layout (tables, headings etc.), controller logic (request handling), business logic, data validation constraints and the persistent data are separate components. We do not rely on any specific technology instead we utilise many of the open, standardised technologies defined and developed by the World Wide Web Consortium (W3C) [5] including XML, XML Schema and XSLT. Figure 1 below shows the architecture of openMVC.

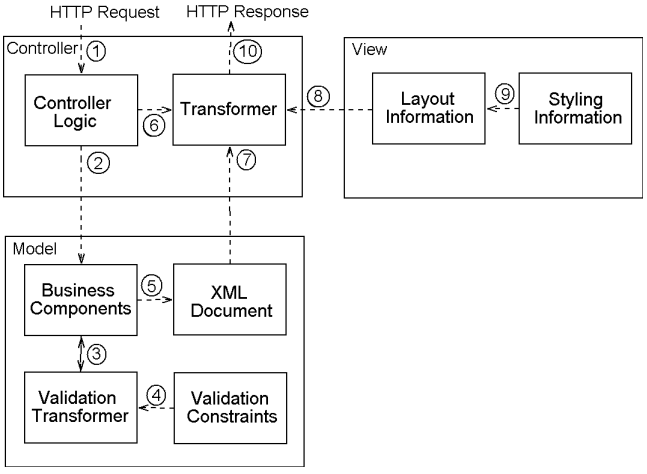

Figure 1. openMVC's architecture.

2.1 openMVC Architecture
The framework is implemented using a 5-tier architecture. The first tier is the client.

The second tier, the Presentation Logic Layer (PLL) is the web server. At this layer the data is given a presentation structure that the browser will be able to display. Details of how the layout elements should look are defined in the style information. Finally data validation constraints which have been sent as XML with the data (from the Model), are transformed to JavaScript and sent to the client. The PLL communicates with the third tier via a Remote Procedure Call (RPC) mechanism.

The third tier, the Business Logic Layer (BLL), typically located on an application server implements the domain specific business processes and rules as well as defining all the data validation constraints.

The fourth tier or Data Abstraction Layer (DAL) abstracts the Database Management System (DBMS) from the code that accesses it. The BLL uses this tier to manipulate the database.

The final tier is the database where the persistent data for the solution exists.

2.2 openMVC Components

The framework contains a number of discrete components at each layer. The separation provided by these components allows for different concerns within the application to be isolated from other unrelated concerns. A number of these components are looked at in detail below.

The MVC's View object role is comprised of two components. The Styling Information component defines presentation styles using Cascading Style Sheets (CSS), which are associated with the structural layout elements. The use of CSS allows style information to be separated from the data structure. A Layout component encapsulates the presentation layout information. The layout will be written using XSLT stylesheets.

The MVC's Controller object role also consists of two components. The Controller Logic component is independent of the View object role and makes the controller workflow logic more readable and easier to maintain as no layout or style information is interspersed within it. A Transformer component transforms the XML returned by the BLL to XHTML using the Layout component. It also transforms the validation constraints returned by the BLL and creates a client side validation script at run time based on these constraints. This script is then returned with the page requested by the client.

The MVC's Model object role is comprised of three components. The Business component represents the objects that exist within a business and the processes that can occur between these objects. The Validation Constraints component ensures the data integrity and security of the system by defining the data that is valid in the context of the business. A Validation Transformer component serializes the Business components into a format that can be validated against the Validation Constraints component to ensure system integrity is upheld.

3. IMPLEMENTATION

A simple e-commerce shopping cart application was built using the openMVC framework.

The DAL was written entirely in Microsoft's .NET framework using the C# language. Stored procedures were used exclusively for improved encapsulation and maintainability.

The BLL was also written in C# using .NET and deployed on a Microsoft IIS web server. Security was provided by the .NET role based security classes. Like any other comparable application server software such as J2EE, .NET provides enterprise services such as transaction control, persistence and security.

The validation constraints for the classes on the BLL are specified in XML Schema files. The business class instances must be serialised to a format to which the XML Schema can be applied and validation can then occur using an XML validating parser.

The BLL classes are exposed via web service functionality provided by the ASP.NET web service technology. The PLL was developed in Java using Sun's Java Servlet technology. Apache Axis an open source implementation of a SOAP toolkit was used to gain interoperability with the web service exposed on the BLL. The PLL was deployed on an Apache Tomcat Servlet container. XSLT templates were cached and reused to improve performance.

The Apache HTTP Web Server provided load balancing functionality to the framework implementation.

4. CONCLUSION

We have presented openMVC a framework based on the MVC pattern. openMVC allows the Style Information, Layout and Validation Constraints components to be updated with no code change or recompile requirement. Redundancy is also reduced as validation constraints, style and layout information are never repeatedly defined. Apart from facilitating faster maintenance, this promotes easier configuration management and reduces version control problems in application rebuilds. Our use of standardised W3C technologies provides for excellent transferal of knowledge and skills across heterogeneous platforms and languages, without the risk of vendor lock-in.

5. ACKNOWLEDGMENTS

The author thanks all at the School of Computing, Dublin Institute of Technology for their support in the creation of this paper.

6. REFERENCES

[1] Krasner, G. E. and Pope, S. T. A Cookbook for Using the Model-View-Controller User Interface Paradigm in Smalltalk-80. Journal of Object-Orientated Programming, 1(3), Aug/Sep 1988, pp26-49.

[2] Gamma, E., Helm, R., Johnson, R., & Vlissides, J. Design Patterns: Elements of Reusable Design. Reading, Massachusetts: AddisonWesley, 1995

[3] Apache Struts Web Application Framework, The Apache Software Foundation (2004), http://jakarta.apache.org/struts/.

[4] Maverick Project, Source Forge (2004), http://mav.sourceforge.net

[5] World Wide Web Consortium (2004). http://www.w3c.org