A NON PROPRIETARY FRAMEWORK FOR POLICY CONTROLLED MANAGEMENT OF THE MODEL IN THE MVC DESIGN PARADIGM

Aaron Jackson
NUI Maynooth, Maynooth, Co. Kildare, Ireland

John G Keating
IBM Dublin Centre for Advanced Studies, Cloughran, Dublin 17, IRELAND
and
NUI Maynooth, Maynooth, Co. Kildare, Ireland

Keywords: WWW, Content Management, Policy, MVC, XML, JAVA, component-based, Model, View, Controller.

Abstract: There are a variety of systems available to help automate and control the Web Content Management (WCM) process. Most of these systems are modelled using the Model-View-Controller (MVC) design paradigm. This is a design technique frequently adopted by software developers to assist in modularity, flexibility, and re-use of object oriented web developments. This design paradigm involves separating the objects in a particular interaction into 3 categories for the purpose of providing a natural set of encapsulating boundaries, encouraging many-to-many relationships along the separate component boundaries, and segregating presentation and content. These MVC based systems control what is known as static content. In this paper we propose a new framework for controlling the software tools used in MVC based systems. More precisely, the automatic deployment of model software tools based on XML defined policies. This framework incorporates a non-proprietary component based architecture and well structured representations of Policies. The Policies are not embedded in the system, they are generated, and therefore each component is self contained and can be independently maintained. Our framework will work on a centralized or distributed environment and we believe that the use of this framework makes it easier to deploy MVC based systems.

1 INTRODUCTION

The pervasive nature of the Web means that it has become the preferred vehicle for content distribution. The explosive growth of the Internet, in particular the World Wide Web, has resulted in heavy demands being placed on Internet servers and has raised great concerns in terms of performance, scalability and availability (Yang and Luo, ICDCS 2000). In order to address these issues, with a main focus on performance, several approaches have been examined. The three most prominent approaches to managing the performance of applications include the IntServ/RSVP (Braden, 1997) signaling approach, the DiffServ approach (Blake, 1998) and the content distribution approach (Akamai, 2004) (Digital, 2004). Whilst each of these approaches have their own advantages, they all require a consistent configuration of a large number of widely distributed devices. The paradigm of using policy based techniques addresses this issue and is being developed for Integrated services and Differentiated services technologies by various groups working within the IETF (Moore, 2001). Policy based management has also been applied to content distribution networks (Dinesh et al, 2002).

In this paper, we propose a technique for using policy based technologies to manage code as if it were content in a content management system. In particular, we "generate" the software tools used in the development of a content management system based on the Model-View-Controller design paradigm (MVC) (O'Reilly, 2003), specifically, the model abstract classes. Most management systems today "hard code" the model. These systems allow for manipulation of the view to create desired web systems. They tend not to deal with the model and only allow for the development of the model tools by a prescribed developer. There are frameworks, like *Jakarta struts* (Cavaness, 2003), that allow a developer to extend the framework for their own needs. This however, still requires extensive development knowledge and programming skill. We propose a framework whereby

Jackson A. and G Keating J. (2005).
A NON PROPRIETARY FRAMEWORK FOR POLICY CONTROLLED MANAGEMENT OF THE MODEL IN THE MVC DESIGN PARADIGM.
In *Proceedings of the Seventh International Conference on Enterprise Information Systems*, pages 451-454
DOI: 10.5220/0002521904510454

the designer does not need extensive programming skills as the configuration and design of the deployed systems can be done simply and effectively by editing the configuration files or policies. This framework is designed for integration with deployment systems for controller and view components of an MVC design.

This paper is structured into 4 more sections. Section 2 outlines the Framework design concepts and gives an architectural overview. Section 3 describes the various components and technologies within the framework. Section 4 illustrates how this framework is applied to a real world situation and is followed by the conclusion.

2 FRAMEWORK DESIGN AND ARCHITECTURAL OVERVIEW

An object-oriented abstract design, also called a framework, consists of an abstract class for each major component (Johnson, 1991). This framework consists of various independently maintained components and uses non-proprietary software technologies. Application-specific frameworks cover precise domains, are highly re-usable and significantly reduce the amount of development required to deploy further customized applications (Parsons, 1999). With this in mind, we propose a 3-tiered architectural approach as illustrated in Figure 1. Our framework is a policy based system. Given a set of policies, determination of model software tools to be used may be achieved. They are a set of rules that the developer issues to the system for it to know how to deploy. These policies are in XML format and adhere to a specific standardized schema. This framework operates as an extension to the WCM process framework illustrated in Figure 2.

Our framework is composed of several interconnected, yet self contained components that are independently maintained. The policies for this framework are not embedded in the evaluation software, but are introduced by the developer. This gives a level of software manageability that cannot be achieved with embedded systems. The first tier of our framework defines the XML polices. This is where the decisions on which tools to use occurs. The policies created must adhere to a specified XML schema for this framework. XML was chosen as the format for these policies and other system components in this framework because of its flexibility in structuring content (W3C, 2004). The second tier in our framework is the "Controller Model" (CM), which interprets the policies and formats the selection for the "Software Tool Repository" (STR) to enable it to select the tools defined by the developer. This information is used by the third tier, STR.

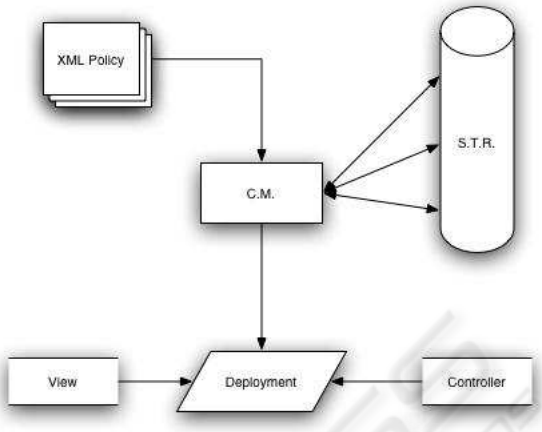

Figure 1: This figure illustrates our frameworks components

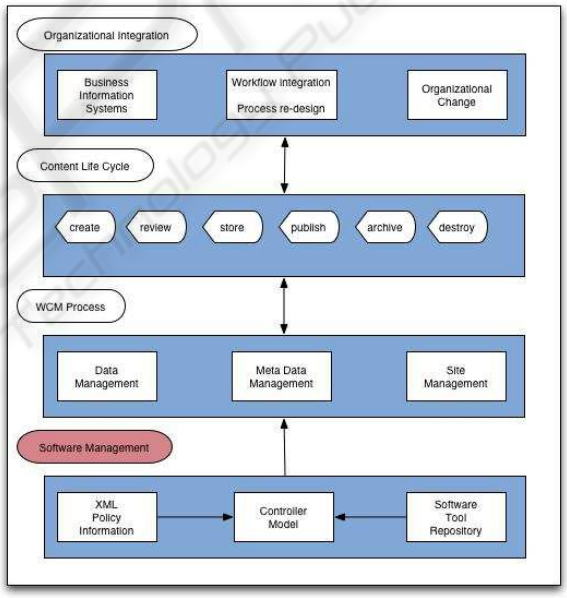

Figure 2: This diagram illustrates the integration of our framework into the WCM process

3 COMPONENTS AND TECHNOLOGIES

The separate components in this framework utilize various non-proprietary technologies. Given that each component is self contained within the framework, the selection of technologies to use was based around the premise that the communication protocols would have to be a generic standard. The first tier of the architecture used XML formatting for the policies. The

configurations are set by the developer at the first tier and enforced into XML based on a schema. When the policies are formed they are sent to the CM via the SOAP messaging protocol (W3C, 2004). SOAP is a messaging protocol that provides a way for applications to communicate with each other over the Internet, independent of platform, which serves our purposes exactly. Additionally, this protocol is lightweight, requiring a minimal amount of overhead and occurs over HTTP.

The CM receives the policies, via HTTP and SOAP from the PM and interprets the information contained within the policies. This information remotely calls handlers that define the files to be retrieved from the STR, based on the indexing system within the STR. It then sends this information to the STR for file retrieval. The CM receives the desired XML file, unmarshalls the XML in order to extract the necessary Java abstract class information. Castor, is used to perform this task (W3C, 2004). These unmarshalled Java programs are the desired abstract Java class information for use in the creation of an MVC based web application.

The STR is the storage repository and interface for the abstract Java classes. A native XML database is used as storage medium (W3C, 2004) (Berkely DB, 2004). Xpath is used to query for information and it supports DOM and SAX. The lack of alternatives and the undesirable results obtained with a relational database resulted in our decision to use this technology. Indexing, matching that of our policy information, leads to fast retrieval of desired files from the database. These XML files are securely sent back to the CM for processing.

4 COMPARATIVE IMPLEMENTATION

An example of how the proposed management of the model would fit into a real life situation will better help to illustrate the concept of storing code as content. A comparison of the "normal" way of writing this model and a look at how we propose to handle the model will yield a clearer understanding of our framework. If we look at a simple web development based on the MVC design paradigm which manages an online employee database for a company. The model functionality provides us with the ability to store, retrieve and edit information in a database about our employees. The controller was written using *Jakarta Struts* with the view written using JSP's and cascading stylesheets.

Since we're dealing with an employee that we want to insert, we need a way to store information about this employee that we could hand off to a business object (the model layer of MVC). Our model layer will be responsible for doing the actual insert. So the first thing we need is a class representing our employee. We'll make a bean that has just a couple of fields and appropriate get and set methods. The object shown in 'EmployeeDTO' class will transfer stored information from one part of our application to another, and is therefore called a Data Transfer Object (DTO) or Value Object. We will create an 'EmployeeService' class to handle the small amount of business logic we have.

The main classes used in the Struts framework are those of org.apache.struts.action.Action. The Action classes are the true concrete bridges between the client's request and the business layer. Each Action class will handle one particular type of business operation the client wants to perform. When we submit our JSP page the Action Servlet and RequestProcessor will find this Action class which we associated with the /insertEmployee action in the struts-config.xml file and the execute method will be called. Since the /insertEmployee action uses the EmployeeForm the InsertEmployeeAction will have this form bean of type ActionForm, so we first cast this into type EmployeeForm. Now we want to get the information in our EmployeeForm into the EmployeeDTO so we can pass that off to our Model layer. We created an instance of EmployeeDTO and then we can use BeanUtils to transfer the data from EmployeeForm into the EmployeeDTO. *BeanUtils.copyProperties(employeeDTO, employeeForm)*. After copyProperties() the now populated EmployeeDTO is passed off to the service object and the model layer takes over. The insertion of data is done by the EmployeeService class and a collection of Department is returned.

From this illustration we can see that the model is the logical operator of the system. It performs the actually desired operations on the data. In this case the interaction with a database. In comparison to coding this, our framework would "plug" this model component in from a repository of model classes based on the configurations needed which are specified in an XML policy configuration file. This would give a level of abstraction in the model that is not present using the above methods.

In the above example important information pertaining to the functionality of the model is essential information for model selection. Pre-defined classes are stored in XML format in an XML database. These files are indexed so as to relate their functionality. For example, a file would have JDBC drivers and knows that the transport medium to the controller will be using JavaBeans and is indexed accordingly. These files, when selected by the "Controlling Model" in our Framework, are unmarshalled from XML. Our framework would plug the model into place. Utilization of our framework allows for better re-use of the

components and technologies. Using the repository of pre-defined classes within our framework, allows for component re-use for similar scenarios and circumstances. This is not achievable using existing methodologies as the Model would have to be re-written to a large degree.

Another comparison to make is that of change within the model and its affects on the system. If a change in the model has to be made using existing methods, the model has to be re-created, changed, tested, re-compiled and implemented back into the system. Using our framework the classes used in the model are remotely stored, with respect to the system, within the XML repository. This allows for remote changes to be made. Testing and re-compilation will have to occur in any realtime Object Oriented system, however the implementation of the newer model would be performed by our CM. This comparison, while simplistic, illustrates how the model can be predefined and then plugged into a development using our framework.

5 CONCLUSION

This framework is the basis for storing code as content. We believe that this is the next logical step towards yielding a level of automation and efficiency that cannot occur with the management of static content alone. Our use of standardized W3C technologies provides for excellent transferal of knowledge and skills across heterogeneous platforms and languages, without the risk of vendor lock-in.

Further development of this concept of storing code as content is essential for providing a viable extension to the WCM process. This includes a standardized integration technique for introducing this paradigm into existing management systems. Ideally, we believe this design framework should be the new standard when designing management systems in the future. Design of management systems without adopting this technique would only serve to further stagnate the evolution of Content Management System design.

This paper also touches on ontological philosophies within computer science. Ontology is the way we carve up reality to understand and process it, with information being a product of that carving (Castel, 2002). In this case, the perception of managing code as content relative to the processes that manage it. Future work will address this ontological issue.

REFERENCES

Akamai Technologies Inc. *FreeFlow content distribution service*. http://www.akamai.com.

Apache Xindice (2004). *Apache Xindice*. http://xml.apache.org/xindice/. 2004 The Apache Group.

Blake, S. (1998). *An Architecture for Differentiated Servers*. Proceeding from IETF RFC 2475, 1998.

Braden, R. (1997). *Resource ReSerVation Protocol (RSVP) Version 1 - Functional Specification*. Proceedings from IETF RFC 2205, 1997.

Castor.org(2004). *The Castor Project*. http://www.castor.org/. 2004 The ExoLab Group.

Cavaness, C (2003). *Programming Jakarta Struts*. Book number 0-596-00328-5.

Digital Island Inc. *Footprint content distribution service*. http://www.digitalisalnd.net/services/cd/footprint.shtml.

Dinesh, C., Seraphin C., and Khali A. *Policy based Management of Content Distribution Networks*. IBM Thomas J Watson Research Centre, New York. TechReport - March 2002.

Felipe Castel. *Ontological Computing*. Communications of the ACM. Feb. 2002/Vol 45, No. 2

Johnson, Ralph(1991). *Designing Reusable Classes*. Journal of Objectg oriented programming. University of Illinois.

Moore, B (2001). *Policy Core Information Model - Version 1*. Proceedings from IETF RFC 3060, 2001.

O'Reilly - MVC arch. (2003). *Model-View-Controller (MVC) Architecture*. http://www.indiawebdevelopers.com/

MVC (2003). *Model View Controller*. http://www.exciton.cs.rice.edu/javaresources/DesignPatterns.

Parsons D, Rashid A, Speck A, Telea A(1999). *A framework for object oriented frameworks design*. Proceedings Technology of Object-Oriented Languages and Systems. IEEE Computer society: Los Alamoitos CA, 1999, 141-151.

Sleepycat Software systems, 2004 *Sleepycat Software Systems* Berkeley XML DB http://www.sleepycat.com

W3C.org(2004). *Extensible Markup Language - XML*. The W3C worldwide Consortium.

W3C.org(2004). *HTTP - communication protocol*. http://www.w3.org/Protocols/. The W3C worldwide Consortium.

W3C.org(2004). *Simple Object Access Protocol (SOAP) 1.1* http://www.w3.org/TR/soap/ The W3C worldwide Consortium.

Yang, Chu-Sing and Luo, Mon-Yen (2000). A Content Placement and Management System for Distributed Web-Server Systems.