2006

Re-engineering the solicitation management system

Yao-Long Fan

Follow this and additional works at: https://scholarworks.lib.csusb.edu/etd-project

Part of the Software Engineering Commons

Recommended Citation

This Project is brought to you for free and open access by the John M. Pfau Library at CSUSB ScholarWorks. It has been accepted for inclusion in Theses Digitization Project by an authorized administrator of CSUSB ScholarWorks. For more information, please contact scholarworks@csusb.edu.
RE-ENGINEERING THE SOLICITATION MANAGEMENT SYSTEM

A Project
Presented to the
Faculty of
California State University,
San Bernardino

In Partial Fulfillment
of the Requirements for the Degree
Master of Science
in
Computer Science

by
Yao-Long Fan
December 2006
RE-ENGINEERING THE SOLICITATION MANAGEMENT SYSTEM

A Project
Presented to the
Faculty of
California State University,
San Bernardino

by
Yao-Lung Fan
December 2006

Approved by:

Dr. David Turner, Chair, Computer Science

Dr. George M. Georgiou

Dr. Kerstin Voigt

11/29/2006
ABSTRACT

Solicitation Management System is a Web-based system for the government agency to process periodically solicitation of grant proposals. It reduces the amount of time and money for the actions of submission, collection and evaluation. Therefore, the future usage of this system is promising.

The purpose of this project is to rewrite the internal architecture in order to reduce the size of this system and make it more maintainable. The previous version of the system was written to rely on the Spring and Hibernate frameworks. Although Spring framework offers a powerful MVC web solution, it means lots of configuration code is needed, which is difficult to understand and maintain. Thus, one of the goals of this project is to remove reliance on the spring framework, but maintain reliance on Hibernate. Also, the current architecture contains plenty of redundant code to carry out similar functions. To reduce the amount of redundancy in the code, another goal of the project is to create generic implementations of system components that can be used for processes with similar actions. By doing that, the system should be more consistent and easier to understand, maintain and extend. Thus, the project proposed here is going to result in an updated version of this system.
ACKNOWLEDGMENTS

I would like to acknowledge my advisor, Dr. David Turner for all the efforts that he had devoted to make this project possible. I would also like to thank him for all the time that he put into this project and for helping me whenever I ran into a problem. I would also like to thank Dr. George M. Georgiou and Dr. Kerstin Voigt for serving on my committee and for giving me guidance along the way.
TABLE OF CONTENTS

ABSTRACT ............................................. iii
ACKNOWLEDGMENTS ...................................... iv
LIST OF TABLES .......................................... viii
LIST OF FIGURES ......................................... ix

CHAPTER ONE: INTRODUCTION

1.1 Purpose ............................................. 1
1.2 Scope ................................................ 1
1.3 Definition and Abbreviations ...................... 2
1.4 Overview ............................................ 5

CHAPTER TWO: OVERALL DESCRIPTION

2.1 Product Perspective .................................. 6
  2.1.1 System Interfaces ............................... 6
  2.1.2 User Interfaces ................................. 7
  2.1.3 Hardware Interfaces ......................... 8
  2.1.4 Software Interfaces ......................... 8
  2.1.5 Communications Interfaces ................. 9
  2.1.6 Memory Constraints ......................... 9
  2.1.7 Operations .................................... 9
  2.1.8 Site Adaptation Requirements ............ 9
2.2 Product Functions .................................. 9
2.3 User Characteristics ............................... 10
2.4 Constraints ....................................... 15

CHAPTER THREE: SOFTWARE REQUIREMENTS SPECIFICATION

3.1 External Interface Requirements ................ 17
  3.1.1 User Interfaces ............................... 17
3.1.2 Hardware Interfaces ...................... 24
3.1.3 Software Interfaces ...................... 24
3.1.4 Communications Interfaces ............. 25
3.2 Functions .................................. 25
3.3 Performance Requirements ................. 26
3.4 Local Database Requirement ............... 26
3.5 Design Constraints ......................... 26
3.6 Software System Attributes ............... 27
  3.6.1 Reliability ............................. 27
  3.6.2 Availability ............................ 27
  3.6.3 Security ................................. 27
  3.6.4 Maintainability ......................... 27
  3.6.5 Portability .............................. 27

CHAPTER FOUR:  SYSTEM ARCHITECTURE .......... 28
  4.1 Object-Oriented Persistence .............. 29
    4.1.1 Persistent Objects ................... 29
    4.1.2 Persistent Object DAOS ............... 32
  4.2 Controller .................................. 33
  4.3 Generic Request Handlers ................. 36
    4.3.1 Handler .................................. 37
    4.3.2 ViewObjectHandler ..................... 39
    4.3.3 ActionHandler ........................... 41
    4.3.4 Validator ................................. 43
    4.3.5 createActionHandler .................... 44
    4.3.6 EditActionHandler ....................... 46
    4.3.7 DeleteActionHandler ..................... 48
4.4 Page Sequence Action Handlers .............. 50
   4.4.1 PageSequenceActionHandler .............. 50
   4.4.2 RegistrationSequenceHandler .............. 53
   4.4.3 CreateApplicationActionHandler .............. 53
4.5 Special Purpose Action Handlers .............. 54
   4.5.1 LoginActionHandler ....................... 54
   4.5.2 LogoutHandler .............................. 56
   4.5.3 UploadProposalActionHandler ................ 56
   4.5.4 ViewProposalHandler ....................... 58
4.6 Utility Classes ................................. 60
   4.6.1 Job, JobQueue and Word2PdfJob .............. 60
   4.6.2 ProposalLocker ............................... 61
   4.6.3 CheckboxItem, CheckboxesTag, AreaTreeTag and ErrorTag ............................... 62

CHAPTER FIVE: SECURITY RELATED ISSUES
5.1 Protecting User Password .......................... 63
5.2 Blocking Unauthorized Access ......................... 63

CHAPTER SIX: CONCLUSION AND FUTURE DIRECTIONS
6.1 Conclusion .................................... 65
6.2 Future Directions .................................. 66

REFERENCES .......................................... 67
LIST OF TABLES

Table 1. Persistent Objects Association Table ...............31
LIST OF FIGURES

Figure 1. Deployment Diagram ........................................ 7
Figure 2. Use Case Diagram - Administrator .................... 11
Figure 3. Use Case Diagram - Officer ............................. 12
Figure 4. Use Case Diagram - Staff ............................... 13
Figure 5. Use Case Diagram - Applicant .......................... 14
Figure 6. Use Case Diagram - Evaluator ........................... 15
Figure 7. Solicitation Management System Login Page ....... 18
Figure 8. Registration Page .......................................... 19
Figure 9. Administrator Home Page ............................... 20
Figure 10. Officer Home Page ........................................ 21
Figure 11. Staff Home Page .......................................... 22
Figure 12. Evaluator Home Page .................................... 23
Figure 13. Applicant Home Page .................................... 24
Figure 14. Class Diagram .............................................. 28
Figure 15. Persistent Objects Class Diagram ..................... 30
Figure 16. Data Access Objects Class Diagram .................. 32
Figure 17. Sequence Diagram ........................................ 35
Figure 18. Handler Class Diagram .................................. 38
Figure 19. Handler Sequence Diagram ............................. 39
Figure 20. ViewObjectHandler Class Diagram ................... 40
Figure 21. ViewObjectHandler Sequence Diagram ............... 41
Figure 22. ActionHandler Class Diagram ........................ 43
Figure 23. Validator Class Diagram ................................ 44
Figure 24. CreateActionHandler Sequence Diagram ............. 46
Figure 25. EditActionHandler Sequence Diagram ..........48
Figure 26. DeleteActionHandler Sequence Diagram ..........49
Figure 27. PageSequenceActionHandler Sequence Diagram ....52
Figure 28. LoginActionHandler Sequence Diagram ..........55
Figure 29. LogoutHandler Sequence Diagram ...............56
Figure 30. Upload Handler Sequence Diagram ...............58
Figure 31. ViewProposalHandler Sequence Diagram ..........59
Figure 32. Word2PdfJob Class DIagram ...................61
CHAPTER ONE

INTRODUCTION

1.1 Purpose

For the purpose of enriching its teaching and learning environment, faculty of the computer science department at California State University San Bernardino (CSUSB) have undertaken several commercial software development projects with graduate students in the computer science masters degree program. One of these projects has been to develop and operate a Web-based system to facilitate the running of grant proposal solicitations by the Office of Technology Transfer and Commercialization at CSUSB. In order to produce a production quality system, we re-engineered the internal architecture to reduce the code size, clean up the unnecessary lines of code, and increase the separation of concerns and consistency. By doing that, we believe the system will be more maintainable and more efficient for the preparation of future expansion.

1.2 Scope

The scope of this project includes a re-engineering of the internal architecture of the Solicitation Management System. The previous version of the system was written to rely on the Spring and Hibernate frameworks. Included in
this project is the goal to remove reliance on the spring framework, but maintain reliance on Hibernate. Also, the current code base for the system contains a lot of inconsistencies in how similar components are implemented. Another goal of this project is to increase consistency in the code, so that the system is easier to understand, maintain and extend. This will not change the functioning of the system as it appears to the end user, except to reduce delay that occurs from weaknesses in the architecture of the current system. Some of the user interface may be modified slightly to improve usability, but the user interface will essentially remain unchanged. Additionally, the current database design will not be modified, because it is adequately well designed. Thus, the project proposed here is going to result in an updated version of the Solicitation Management System.

1.3 Definition and Abbreviations

**SMS** - Solicitation Management System.

**OTTC** - Office of Technology Transfer and Commercialization.

It is an office that assists in transitioning promising new technologies from government and academic laboratories alike into full commercialization.
**Hibernate** - Hibernate is an Object-relational-mapping (ORM) framework for Java. It lets developers develop persistent classes following common Java idiom.

**Apache** - A public-domain open source Web server developed by a loosely-knit group of programmers. The source code is freely available, anyone can adapt the server for specific needs, and there is a large public library of Apache add-ons. In many respects, development of Apache is similar to development of the Linux operating system. The original version of Apache was written for UNIX, but there are now versions that run under OS/2, Windows and other platforms.

**API** - Application Programming Interface.

**HTML** - Hyper Text Markup Language.

**Java** - An object oriented language developed by Sun Microsystems, Java programs are capable of running on most popular computer platforms without the need for recompilation.

**Java Servlet** - A Java application that runs in a web server or application server. It provides server-side processing which usually is used to access a database or perform e-commerce processing.
JSP - Java Server Page, an extension of the Java server technology from Sun that provides a simple programming vehicle for displaying dynamic content on a web page.

JDBC - (Java Database Connectivity) is a programming interface that allows Java applications to access database through the SQL language.

MySQL - MYSQL is a software that delivers a very fast, multi-threaded, multi-user, and robust SQL (Structured Query Language) database server.

Tomcat - Tomcat is an application server from the Apache Software Foundation that executes Java servlets and renders Web pages that include Java Server Page coding.

UML - Short for Unified Modeling Language, a general-purpose notational language for specifying and visualizing complex software, especially large, object-oriented projects.

Use Case Diagrams - A diagram provided by the UML to facilitate the process of requirements gathering. The use-case diagram models the interactions between the system's external clients and the use cases of the system. Each use case represents a different capability that the system provides to its clients.

Use Sequence Diagrams - A diagram provided by the UML to show the sequence between classes.
1.4 Overview

This paper reports on our experience using recently established, popular open source tools to re-write a small-to-medium-sized Web application for a government agency. Therefore, we create a new architecture by using Java Servlet API to simplify the implementation of the old system, and Hibernate framework to simplify the management of persistent data. In addition, the JSP and persistent objects architectures will be almost the same because the application needs only to provide the access to basic functions, and does not need to be embellished with advertising. Thus, we re-engineered the internal architecture to have a simple, consistent and flexible solution for the system.

The Solicitation Management System (SMS) was built to facilitate the processing of grant proposal solicitations. The system comprises the four versions of SMS, and was developed collaboratively with Dr. Arturo Concepcion, Dr. David Turner and several graduated students as follows. Rajiv Sadagopan and Sathya Prasad helped to develop version one of the system. Rick Pallow and Jonathan Wang helped to develop version two. Christina Wu, Alan Lin, and Robert Chen helped to develop version three. The fourth version of the SMS was developed within this project.
2.1.2 User Interfaces

All the users use the same login page whatever their role type. Each user logs in by using the personal username and password, and is redirected to his home page according to role type. From his home page, the user can access functions of the system that are specific to his role. The user interface consists of roughly 150 web pages due to the large scale of this system. Each page is provided with several components that guides the user throughout the system. These components include a static menu, a trail of breadcrumbs, and a dynamic sub-menu.

A static menu is provided for each page and is located of the left of the page. The content of the submenu differ
from role to role. The user can use the static menu to navigate to the pages or functionality that he desires.

In addition, there is the trail of breadcrumb on each page. The purpose of the breadcrumb is to display the pages that users have been through that can be the information of the level of submenu and the trace from the beginning.

Finally, the dynamic sub-menu displays the functionality relative to this page. Since each page has different functionalities and serves different purposes, the sub-menu changes dynamically.

2.1.3 Hardware Interfaces

The system is an on-line system. Thus, the requirement is a computer with Internet access. Hence, any operating system with a browser is available to access the system.

2.1.4 Software Interfaces

The project software interface will be viewed over the web. It should run on any web browser for Windows, Linux, or Mac OS. The language and applications used in writing this interface are Java, JSP, and CSS. The server runs Linux. The main part of the system runs on the Linux server and the file converting part of the system runs on Windows XP Professional with Microsoft Office 2003 installed.
2.1.5 Communications Interfaces

The project is a web-based program, and the interface could be open from any web browser such as Mozilla, Firefox, Internet Explorer, etc. There are no local network protocols needed. Latest versions of the browsers are recommended because outdated version may lead to unexpected results in accessing certain functions in the system.

2.1.6 Memory Constraints

The primary memory for the solicitation management system should be at least 256 MB. The size of the secondary storage depends on the software installed and the database that will hold all records. The initial estimate indicates that at least 20 GB will be needed.

2.1.7 Operations

The system will operate 24/7. Backups can be done by using the dump of the database once a month. Maintenance will be done on call, and mostly remote.

2.1.8 Site Adaptation Requirements

The solicitation management system does not have any site adaptation requirements. However, a screen resolution of at least 600 X 800 is encouraged.

2.2 Product Functions

This product has three main functions which are to send the submission of the grant solicitation, process the
applicant's submission and evaluate the solicitation. All of those actions are handled on-line which can save the amount of process time.

2.3 User Characteristics

The SMS system facilitates the processing of a solicitation. It serves users from five different role groups including applicant, evaluator, admin, officer, and staff. Each user role has a home page to perform different functions and modify the personal profile.

The administrator can manage officers and staff members through various operations. These operations include viewing a list of officers and staff members, viewing the detailed information of a specific officer or staff member, creating a new account for an officer or staff member, editing information for a specific officer or staff member, and finally deleting an account of an officer or staff member. The use case diagram is shown at next page.
An officer can execute create, edit and delete operations. These operations can apply to solicitations, applications, awards, evaluators, applicants, and evaluations. The use case diagram is shown below.
Figure 3. Use Case Diagram – Officer
The staff member can list and view many components of the system. These components include the solicitation, applications, awards, evaluators, applicants, and evaluations. The diagram is shown below.

Figure 4. Use Case Diagram - Staff
The applicant can submit his application and proposal online and manage the personal profile. This system accepts proposals in either PDF file or Word file. The use case is shown below.

![Diagram](image)

Figure 5. Use Case Diagram - Applicant

The evaluator can evaluate applications assigned to him. He can start this process by reading the applicant proposal and then answering the evaluation questions assigned to the specific solicitation.
2.4 Constraints

The system has some mainly Constraints. First, the user information for all role type regards the email and password. The email need to be the form with “@” characters and the password has to be at least six characters. Second, it is the individual ability constraints for the applicant, evaluator and officer. For example, the applicant can only upload his proposal with the PDF or Word file and edit his application only before the submission deadline. In addition, the evaluator can only evaluate after the submission deadline and before the evaluation deadline and the evaluation can not be changed after finished.

For the officer, he can not delete a solicitation which one or more applications submitted and can not delete an
applicant who has a submitted application and an evaluator with the evaluation.
CHAPTER THREE
SOFTWARE REQUIREMENTS SPECIFICATION

3.1 External Interface Requirements

3.1.1 User Interfaces

The screen format is almost similar with the previous version. Thus, the discussion will only mention the most frequently use pages.

3.1.1.1 Login Page. The login page for all the user is shown in Figure 7. The content includes the banner and the instructions. Users input the username and password to login the system. After passing the validation, users are redirected to their homepage according to user role. The forgot-password link is used to retrieve the password if user forgot password.
3.1.1.2 Registration Page. Registration page is shown in Figure 8 where user can do the registration process to be the applicant or the evaluator. After users complete the registration process successfully, they will be redirected to his home page.
3.1.1.3 User Home Page. The following figures show the home page of each role respectively. In administrator home page, The administrator can create the new staff or new officer and edit delete their account form the page and the links.
In officer home page, all the solicitations are listed with the link to display the detail information. The other links include management the applicants, evaluators, awards and application groups.
Figure 10. Officer Home Page

Figure 11 shows the staff home page. It’s almost the same with the officer home page except it can only execute the list and view actions.
In evaluator home page, it lists the applications that need to be evaluated by this evaluator. In addition, evaluator can click the link to display the proposal from this application or view the evaluation after he or she has finished.
Figure 12. Evaluator Home Page

Figure 13 shows the applicant home page. Applicant can choose the open solicitation to apply or view the application he already applied and submit the latest version of the proposal before the deadline.
3.1.2 Hardware Interfaces

Two Dell computers are used as servers.

Each server has at two network connectors and four USB connectors. The servers are connected to the Internet. Their IP addresses are: 139.182.139.150 and 139.182.139.152.

3.1.3 Software Interfaces

The main system is installed on a Linux RedHat Enterprise 3.0 platform. To support the system, Tomcat 5.0.28 and Oracle 9i are installed.
The connection between Web application and document conversion server is done using an application-specific protocol over TCP. A single-threaded server client running in the Web application makes a single connection to the document conversion server.

3.1.4 Communications Interfaces

The System uses TCP/IP (Transmission Control Protocol/Internet Protocol) to communicate over the Internet.

3.2 Functions

The System administrator is responsible for managing the staff and officer user. He can create, view, edit and delete the accounts and information of those two roles of users.

The officer can operate most of functions in the system including management of applicant and evaluator accounts, the whole operation of solicitations, awards and application groups, the ability of uploading applicant’s proposal, the selection of assigning evaluator and globe report generations.

The staff only can do the view and list functions that an officer can do and generate reports for a specific solicitation.
The evaluator can evaluate the applications that are assigned to him and read proposals during evaluation and view the evaluations after finished.

An applicant can submit applications according to open solicitations, upload the latest proposal to replace the old one and perform view or delete action of his own applications.

3.3 Performance Requirements

The loading of most Web pages should take less than 5 seconds. The exceptions to this include generating reports, document conversion, and file uploading.

3.4 Local Database Requirement

The local database requirements for this system are the requirements for installing and running the Oracle database. Oracle 9i database requires at least one gigabyte memory.

3.5 Design Constraints

The programmer, during its design, should use only currently available computers as a server. The developer should use the latest and/or most mature technology to design the system and JSP to write the view component of the web application.
3.6 Software System Attributes

3.6.1 Reliability

The system should be reliable and crashes should not occur.

3.6.2 Availability

The system should be available to agency staff and other potential users who wish to register themselves into the system.

3.6.3 Security

This system is designed with security in mind such that a user cannot access information of another user.

3.6.4 Maintainability

This system is built with most configurations allocated in one place so that it will be easy to make changes and maintain when needed.

3.6.5 Portability

This system is built with as less dependencies as possible to gain portability.
CHAPTER FOUR
SYSTEM ARCHITECTURE

An overview of the new system architecture is presented in the class diagram in Figure 14. The details will be discussed in the following sections.

Figure 14. Class Diagram
4.1 Object-Oriented Persistence

4.1.1 Persistent Objects

The persistent object is like a box that can hold the information data and is divided into several sections to fit different data. In addition, there must be several different kinds of boxes for different purposes. The data need to be placed in exactly the right box. Then, those boxes will be saved into the storage called a database for further usage.

The classes that represent persistent objects in this system include User, Applicant, Evaluator, Application, Evaluation, Solicitation, Award, ApplicationGroup, Proposal, TechArea and BusArea. The class diagram is shown below. It is obvious that all the persistent objects are inherited from the PersistentObject class which has the "id" attribute. This attribute is a Long variable, which is used for identification. The diagram is used for the description of the dependencies. Therefore, those classes do not contain all the attributes and methods in detail.

The User class is used to hold the information of basic users, including: admin, staff and officer. The information includes the username, password, email and role type.

This diagram shows that the Applicant and Evaluator classes are associated with the User class. Both of these classes have a one-to-one relationship with the User class,
because they have a User variable inside to save the basic user information. Figure 15 shows the relationship of those classes.

The Solicitation class holds the information of the solicitation including the title, deadline, etc.

The Application class is used for applicant's application. It includes proposal, evaluation, etc. and associated with many classes.

Figure 15. Persistent Objects Class Diagram
Table 1. Persistent Objects Association Table

<table>
<thead>
<tr>
<th>Class</th>
<th>Classes</th>
<th>Association</th>
</tr>
</thead>
<tbody>
<tr>
<td>User</td>
<td>Applicant, Evaluator</td>
<td>1:1</td>
</tr>
<tr>
<td>Evaluator</td>
<td>Evaluation, BusArea, TechArea</td>
<td>1:M</td>
</tr>
<tr>
<td>Applicant</td>
<td>Application</td>
<td>1:M</td>
</tr>
<tr>
<td>Solicitation</td>
<td>Award, Application, ApplicationGroup</td>
<td>M:M</td>
</tr>
<tr>
<td>Application</td>
<td>Award, ApplicationGroup, Evaluator,</td>
<td>M:M</td>
</tr>
<tr>
<td></td>
<td>Evaluation, BusArea, TechArea</td>
<td></td>
</tr>
</tbody>
</table>
4.1.2 Persistent Object DAOs

Hibernate is a mapping tool used for Java environments. Hibernate can map a data representation from an object model to a relational data model in the database.

The data access objects (DAOs) use the data query and retrieval facilities to implement the business logic in the database. The main actions include create, edit, find, list and delete purpose of logic. It can significantly reduce the time and simplify the implement instead of writing SQL and JDBC script to access the database.

All the DAO classes inherit form the ParentDAO class, which provides the basic methods to perform the actions mentioned before, and each subclass can override these methods for special purposes. Figure 16 illustrates the class hierarchy for the data access objects.

![Figure 16. Data Access Objects Class Diagram](image.png)
4.2 Controller

The controller is the only class in this architecture that is inherited from the HttpServlet class. The J2EE web container (tomcat in this case) creates a single instance of the Controller class to process all incoming HTTP get and post requests from browsers. The controller servlet implements functionality that is common to the handling of all (or many) requests. After performing these global actions, the controller passes execution to a Handler in order to carry out specific functionality needed to process the given request.

The sequence diagram in Figure 17 illustrates how the controller servlet operates. First, the Controller gets the request from the browser when the web container invokes either its doGet or doPost methods. The controller does not distinguish between get or post requests; it simply calls the processRequest method from both doGet and doPost methods.

The controller then obtains a Hibernate session object by static method call on the HibernateUtil class. The Controller uses the session instance to open a database transaction by calling its beginTransaction method, which means a connection is created between the database and Controller. As a result, we will be able to operate on the
persistent objects in the database by using the Hibernate session.

After establishing a database transaction, the Controller uses the HandlerMap to locate the Handler class to service the request based on the request URL. If no Handler class is matched, the Controller logout the user to the login page for security reasons. After obtaining the correct handler to process the request, the Controller calls the authorized method of Handler class, which returns true if the user is authorized to perform the operation that the Handler implements. If the user tries to access an unauthorized resource, the system will redirect the user to the login page.

After the controller verifies that the user is authorized to access the requested resource, the Controller places the request object one or more references into objects that represent the logged-in user, because many handlers or their corresponding JSP will need access to this information. Note that we do not place these references in the session object, because they refer to lazy loaded persistent objects whose state is accessible only within the database transaction that is started and ended by the controller for each given request.
The controller then calls the process method on the Handler to carry out the remaining details needed for servicing the given request. Most of the implement is made in this method according to different functions needed. After the process method returns, the controller commits the database transaction, which will cause all persistent objects whose states have changed to be written to the database.

Figure 17. Sequence Diagram
4.3 Generic Request Handlers

The classes in the SMS system that are used to handle incoming requests, which are user operations such as login, change password, upload proposal, etc. All of these classes subclass a class called Handler. The Handler class itself is used to simply execute JSP scripts in the case that no other processing is needed to service the request. If more processing is needed before generating an HTML response, a subclass is defined, and either the process method and/or the authorized method is overridden in order to implement the needed functionality. Basically, the function of Handler classes is to process requests from the browser and then forward execution to a JSP in order to render the view to the user.

There are many browser requests that can be handled by the same handler code. In order to avoid writing redundant code, we group together these requests and service them with a single generic handler. The main functionalities provided by these generic handlers include listing, viewing, creating, editing and deleting persistent objects (objects whose states are stored in the database, and thus survive restarts of the system). The Hander class is the super-class and all these generic handlers, which include ViewObjectHandler, CreateActionHandler, EditActionHandler and
DeleteActionHandler. The follow sections will describe these handler classes in detail.

4.3.1 Handler

In addition to serving as the parent of the handler hierarchy, the Handler class is used for viewing the simple page without passing any persistent object. The class diagram and sequence diagram for the Handler class are shown in Figure 18 and Figure 19, respectively.

The class diagram shows Handler is associated with the Controller and the HandlerMap. The sequence diagram shows how Handler renders a simple JSP to the client. First, it calls the doJsp method after the Controller calls the process method. Then, doJsp method sets the HTML header information and creates a RequestDispatcher instance by assigning the path of JSP. Its function is to display the view according the request URL from browser because we make the JSP file architecture like the request URL. For example the URL is “https://localhost:8443/sms2/visitor/login.html,” then the JSP file path is “/WEBINF/jsp/visitor/login.jsp”. Finally, the doJsp method calls the forward method of the RequestDispatcher to shift the control to JSP. Then, The JSP generates the HTML syntax and Tag class if there is any generates the HTML syntax to the OutputStream. Then,
OutputStream dumps all the HTML syntax to generate the page to the browser.

Tag class is act like JavaServer Pages Standard Tag Library (JSTL) which is a component technology to write dynamic JSP pages. We implement the javax.servlet.jsp.tagext.TagSupport class to create the special tag classes to fit the system requirement that includes the AreaTreeTag and CheckboxesTag. Using those tag class can simplify the JSP and reduce the code of line.

Figure 18. Handler Class Diagram
4.3.2 ViewObjectHandler

The ViewObjectHandler is also used for viewing the page like handler class but the difference is that it needs to pull out the persistent object from the database and then uses it to generate the view. Figure 20 shows the class diagram for the ViewObjectHandler. The ViewObjectHandler would operate on the persistent domain objects through DAOs.

In this way, we further isolated SQL code needed for database interaction. Figure 21 shows the sequence diagram. First, Controller calls the process method of ViewObjectHandler. Then, ViewObjectHandler calls the find method of ParentDAO by passing the type of persistent
objects and the id. Then, ParentDAO uses Hibernate session to find the exactly persistent object and return the object. Finally, ViewObjectHandler calls doJSP method to display the HTML page.

Figure 20. ViewObjectHandler Class Diagram
4.3.3 ActionHandler

ActionHandler is the parent handler class used for processing HTML form submission. The Class diagram is shown in Figure 22. The ActionHandler is not instantiated directly; only its subclasses are instantiated. For this reason, we override the process method of Handler, but we declare it as abstract. The subclasses override the process method in order to perform create, edit, and delete functions. All the subclasses of the ActionHandler class need to associate with the ParentDAO class in order to access persistent objects from the database. The CreateActionHandler and EditObjectAction classes are associated with the Validator class which is used for validating submitted data. The

Figure 21. ViewObjectHandler Sequence Diagram
details of the subclasses will be described in following sections.

The ActionHandler contains methods that are common to all handlers that process form submission. Those are setSuccessUrl, setCancelUrl and hasErrors. The setSuccessUrl method is used to set the page URL for the current request has been handler correctly. The subclasses will override the method with some special required. For example, the id parameter needs to be attached in the URL to redirect page. The setCancelUrl method is used to set the page URL for undo action which will redirect the current page to the action start page. The hasErrors method is used to detect if there is the validation error and It will return a Boolean value if that happens.
Figure 22. ActionHandler Class Diagram

4.3.4 Validator

There are many Validator classes and all of them are inherited from the Validator class and implement the ValidateAndBind method. The class diagram below just lists part of them. The Validators are used by the handler classes to validate the data. Therefore, they need to be reusable instead of generating the duplicate code. For example, the CreateActionHandler class and the EditActionHandler class
can use the same validator because the form they get from the user request are almost the same. Therefore, they can use the same Validator to validate the data.

The other methods in the Validator are addError and finOrCreate. The addError method is used to add the error message to the JSP page if validation errors happen. The findOrCreate method is used to find the persistent object form the database. The detail will describe in the CreateActionHandler session.

Figure 23. Validator Class Diagram

4.3.5 CreateActionHandler

CreateActionHandler is used to handle the incoming request with the action of creating the new persistent object. The sequence diagram is shown in Figure 24. First, Controller calls the process method. Then, the process will
run the conditional logic if the cancel button is clicked which calls the redirectCancelUrl method to redirect the page to cancel page. If not, the validateAndBind method of the Validator is called and it calls the findOrCreate method. FindOrCreate method returns a persistent object whatever it can be found in the database. If the persistent object cannot be found, the method will create a new instance and return it.

In addition, the persistent objet will be set by the Validator to HttpServletRequest instance. Then, CreateActionHandler can get the object from the HttpServletRequest object and call the hasErrors method. If validation errors happen, the CreateActionHandler will call doJSP method to redisplay the form page with the error messages. If not, the ParentDAO will be called to save the new persistent object to database. Finally, CreateActionHandler calls the redirectSuccessUrl method to redirect the user to the successful page.
4.3.6 EditActionHandler

The sequence diagram of EditActionHandler shown below is almost the same with CreateActionHandler and they use the same Validator to verify the form data. The difference is that Validator calls the ParentDAO to find the existing
persistent object form the database instead of creating the new one. Furthermore, EditActionHandler calls the cancelUpdate metod of ParentDAO if the validation errors happen. It is because the session cache already has the change of wrong data. In order to avoid hibernate save those changes into the database, the session cache has to be clear by calling this method. Therefore, EditActionHandler do not call create method of ParentDAO because those changes are save automatically if there is no validation errors.
4.3.7 DeleteActionHandler

This class is used to delete the persistent object. Most of the logic is like CreateActionHandler. The difference is that it calls the isErasable method of PersistentObject before delete action. The purpose is
because some PersistentObjects could contain some other PersistentObjects and the upper level one can not be removed if the lower level one are exist. For example, the one of the applicant object can not be deleted if there are some applications related with it. Therefore, the delete action can be run if the isErasable method returns the true value. Then, DeleteActionHandler calls delete method of ParentDAO delete the persistent object from the database.

Figure 26. DeleteActionHandler Sequence Diagram
4.4 Page Sequence Action Handlers

4.4.1 PageSequenceActionHandler

PageSequenceActionHandler is used to handle multi-pages data input. For example, the registration of the applicant or evaluator needs to go through several pages to complete the action. In order to complete action, the same incoming request is handled by this handler several times to do the same action.

The sequence is shown in Figure 27. First, Controller calls the process method like calling other handlers. Then, it tries to get the attribute “page” from the HttpServletRequest object. This attribute is used to decide which Validator needs to be called in the following sequence. In this multi-page data input sequence, every page needs to be validated so we put those Validators in a Array that we can use the page value to find the Validator corresponded.

In addition, before validating the page, the checking of cancel-button and previous-button clicked has to be done. If user hits the cancel-button, the page will redirect to the cancel page. If user hits the previous-button, the page value needs to be decrease one in order to go back to the previous page by calling the doJSP method. Then, the Validator will be called according to the page value if it
passes the checking. The rest of validation action is the same with CreateActionHandler.

After passing the Validation, it means user clicks the next-button. The page value will be increased one and Controller calls the doJSP method to display the next page. This process repeats several times until the submit-button is clicked. Then, Controller calls the create method of ParentDAO and redirect the successful page.
Figure 27. PageSequenceActionHandler Sequence Diagram
4.4.2 RegistrationSequenceHandler

This handler is used to handler the registration of new applicant and evaluator. It is the subclasses of PageSequenceActionHandler and it overrides the redirectSuccessUrl method. It is because the new user needs to be redirected to the home page according to whose role type. For example, new applicant needs to go to applicant home page after the registration done. Therefore, redirectSuccessUrl method calls ParentDAO to find the User Object because it is called after Hibernate already saved the User object into the database. Then, get the id and the role value form this User object and save them to the HttpSession Object. Finally, it redirects the page to successful page.

4.4.3 CreateApplicationActionHandler

The handler is used for the action of the applicant creates the new application. This class is also inherited form the PageSequenceActionHandler and overrides the redirectSuccessUrl and redirectCancelUrl methods. It is because the page needs to be redirected to the applicant’s home page if the action is cancel and to the view page after new application is done. Therefore, the redirectSuccessUrl method gets the id value of the new application object and attaches it to the URL. The redirectCancelUrl method gets
the userId value from the User object and attaches it to the URL. Finally, both methods use the new URL to redirect the page.

4.5 Special Purpose Action Handlers

4.5.1 LoginActionHandler

LoginActionHandler is used to validate the user login action with the username and password. It is inherited from the handler and overrides its process method. First, it gets the username and password attributes form the HttpServletRequest object. Then, it accesses the database to find the User object by call findByUsername method with the username. If the user does not exist, the ParentDAO will return a null value and LoginHandler will check for that statement and calls the doJSP method to render the login page again. After the LoginActionHandler find the User object, it will check the user password by calling checkPassword method which calls digest method of the PasswprdDigester Class. It's because the password in the database is encrypted so the password input by the user needs to be encrypted again that it can be compared with the value in the database. If they are not matched, the doJSP method gets called to display the login page with the error message. Finally, the LoginActionHandler saves the userId
and role value of the User Object to the HttpSession for the further usage and redirect the user to the home page according to the role value.

Figure 28. LoginActionHandler Sequence Diagram
4.5.2 LogoutHandler

The Handler is simple but important because it needs to clear the session cache before user logout the system. By calling invalidate method of HttpSession, it can remove all the attribute in the session such as userId, role, etc. Then, it redirects the page to the login page.

![LogoutHandler Sequence Diagram](image)

Figure 29. LogoutHandler Sequence Diagram

4.5.3 UploadProposalActionHandler

The UploadProposalActionHandler is used to upload the proposal which can be the PDF file or the Word file. First, it calls parseMultipartRequest method to parse the multipart
request by passing the max file size, the memory limit and the location to store files temporarily. Then, this method will save the fileItem and formFields attributes to the HttpServletRequest object for further usage. Then, This Handler gets the formFields attribute from the HttpServletRequest object which includes all the attributes from the JSP such as id, cancel-button, etc. It can not call getParameter method to get those attribute directly because all of them is encrypted by setting the encrypted attribute of HTML table to "multipart/form-data". Then, the cancel-button will be checked if user clicks that.

Second, the handler calls ParentDAO to find the Application object in order to get the Proposal object for this application. Then, it gets the fileItem which includes the information and the data of the file uploaded. Then, it will check for the file type, filename and the size. If everything is correct, the write lock will be set which means this proposal is not writable currently. Then, the handler calls the store method of the Proposal object to save the file into the database and redirect to the successful page. The process mentioned before is for the PDF file. If the file type is Word file, the handler will call the Word2PdfJob class to convert the file to PDF file. This will be described in detail in the Utility Classes section.
4.5.4 ViewProposalHandler

This handler is used to display the applicant’s proposal which is PDF file. First, it calls ApplicationDAO to get the Application object and then calls the getProposal of the Application object method to get the proposal of this application. Then, this handler will check the status of the application. If the status is incomplete, the page will display the message of incomplete message and stop all the
process. After passing the checking, the Locker class will set the read lock by calling setReadLock method. Then, the handler will call the setHead method to set the header of the HTTP response. Then, it calls the retrieve method of the Proposal object to insert the application referenceNumber into the content and generate the page. Finally, it unlocks the read lock.

Figure 31. ViewProposalHandler Sequence Diagram
4.6 Utility Classes

4.6.1 Job, JobQueue and Word2PdfJob

Those classes are used to convert the Word file to PDF file. First, UploadProposalActionHandler will create a Word2PdfJob object and call the addJob method of the JobQueue class with this object. Then, JobQueue will run the process method of Word2PdfJob to convert the file. There is an important part needed to be noticed here. JobQueue is inherited form the Runnable class and it can handler the different Threads at the same time. Thus, its run method is running at the beginning when the Init class calls start method to start the Thread, and it will be end after the stop method is called. In addition, JobQueue has Job LinkedList and it will check it every 2 seconds. If there is a job object passing into this container, JobQueue will call the process method of the Job Object. This is how the conversion gets call.

Inside the process method, Word2PdfJob will Sent the Word file to the conversion server and it will send back a PS file. Then, Word2PdfJob will call the external software to convert the PS file to PDF file and save it to the Proposal object to finish the job.
4.6.2 ProposalLocker

This class assumes that objects passed in will not change their identity. For this purpose, it will set the read and write locks which are represented by HashMap class. The write lock will be set when the PDF file is saved into the database, the Word file is converted or it is being reading. If the users try to read or write the proposal, the id of the Proposal object will be passed into this method and the hasLock method will check for the HashMap if it contains this id value. The read request will be reject if the write lock is set.
The read lock is set if the users are reading the proposal. If the applicant and the officers try to upload the updated proposal, the request will be rejected because the lock is set. After the users finish the reading, the lock will be unlocked and then the applicant or the officers can upload the new proposal.

4.6.3 CheckboxItem, CheckboxesTag, AreaTreeTag and ErrorTag

CheckboxItem is an interface including getId and getName methods. Award and ApplicationGroup classes implement this interface for the usage by CheckboxesTag class.

Those classes are used to simply the HTML syntax to only one line statement in JSP file. The CheckboxesTag is to display the checkboxes with the Award and ApplicationGroup selections. The AreaTreeTag is used to show the tree of business and technical background information. The ErrorTag is to show the validation error message. All the tag classes are inherited from the TagSupport class and implement doStartTag method. By using them, the logic can be implemented first and be displayed simply in the HTML page.
CHAPTER FIVE
SECURITY RELATED ISSUES

The main focus for security of the SMS is to keep the information related to the user account intact from uninvited access. This would include protecting the user password and blocking information access from unauthorized personnel.

5.1 Protecting User Password

The mechanism for protecting the user password is to hash the password before saving it to database. Thus even if a person can access the database directly, he cannot retrieve the original user password, all he can get is the hashed password.

5.2 Blocking Unauthorized Access

The authorized method of Hanlder class can check the authorized access by comparing the request URL with the user role type. If the URL does not start with the role type, the page will be redirected to login page. Thus, the users just can access the page of their role type.

In addition, if an evaluator tries to read the proposal from other application by changing the id vale of the URL, the system compares the requested proposal to all of the
assigned proposals. If there is no match, the system will redirect the user back to the evaluator homepage.

If the applicant tries to operate other applicant’s application, the system compares the requested application to collection of his own applications. If there is no match, the system will refuse to render the requested page and redirects the applicant back to the applicant homepage.
CHAPTER SIX
CONCLUSIÓN AND FUTURE DIRECTIONS

6.1 Conclusion

Our experience in rewriting the SMS system using Java Servlet API and Hibernate frameworks demonstrates a significant reduction in lines of code. First, we get rid of the configuration code for the Spring framework. Though declarative programming like Spring can make the code simple by putting settings in the configuration files, the disadvantage comes right away. The system’s maintainability is reduced by maintaining too many configuration files. In addition, shifting away from declarative programming makes it easy to catch problems earlier during the compilation step instead of later in the runtime environment. The lines of code across the Java source code files for version 3 is 18,160 lines, and for version 3 is 6,358 lines, which is a decrease of 65%. Furthermore, the LOC of configuration has decreased by 70%, from 3764 lines in version 3 to 1115 lines in version 4. We attribute this reduction to the discard of Spring framework.

In addition, we achieve another goal of creating the generic classes to handle the creation, editing, deletion and display actions. The previous version has lots of
classes to handle each action of each persistent object, which makes the system very huge. Furthermore, we cleaned up the JSP scripts and created some tag classes to make it shorter. By achieving the two goals, we believe that the system is smaller, simpler and more maintainable.

6.2 Future Directions

The future direction is to simplify the existing server architecture, which means the operation should be all in one machine instead of processing the doc conversion and accessing the database in different machines. In addition, the doc conversion process needs to be re-examined and improved in order to make document conversion more efficient and less likely to generate infinite locks on proposals when subjected to abnormal conditions.

Finally, the future direction is to add new functions and new persistent objects to fit different purposes for potential customers.
REFERENCES

[1] Hibernate Framework
Hibernate 2005
(http://www.hibernate.org/)

[2] MySQL Homepage
MySQL AB 2006
(http://www.mysql.com/)

The Apache Software Foundation 2006
(http://jakarta.apache.org/tomcat/)

IEEE Recommended Practice for Software Requirements
Specifications.

[5] F. Ortin, B. Lopez and J. Baltasar Garcia Perez-
Schofield, "Separating Adaptable Persistence
Attributes through Computational Reflection," IEEE

The Apache Software Foundation 1996
(http://httpd.apache.org/docs-2.0/)Java Servlet
Technology
Sun Microsystems, Inc 2006
(http://java.sun.com/products/servlet/)