2005

Graduate Advisor System

Richard Brian Pallow

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

Part of the Databases and Information Systems Commons

Recommended Citation


This Thesis 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.
GRADUATE ADVISOR 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
Richard Brian Pallow
December 2005
GRADUATE ADVISOR SYSTEM

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

by
Richard Brian Pallow
December 2005

Approved by:

Dr. David Turner, Chair, Computer Science

Dr. Arturo Concepcion

Dr. Kerstin Voigt

12/1/2005
ABSTRACT

The Graduate Advisor System is a Web-based application that allows a prospective student for the Master of Science in Computer Science at California State University San Bernardino to apply for the program online. The prospective student can create an account with the system which will enable them to log in and complete their application over multiple sessions. Recommenders for the student’s application will also be able submit their reference letters online via the application. The system automates a large portion of Department of Computer Science staff work for applications by generating reports and data entry.

This project is a revision of the original Graduate Advisor System. The goal was to update the software architecture and use state of the art software modules in the implementation. This required a complete rebuild of the application using the Model-View-Controller architecture pattern. This project has successfully brought this application up to date with today’s newest web applications and at the same time, has retooled the user interface for a cleaner, simpler look. This project is a solid building block for future iterations of the
Graduate Advisor System and could successfully be used for other Web-based applications with minimal changes.
ACKNOWLEDGMENTS

I would like to thank the faculty of Computer Science Department for giving me an opportunity to pursue my M.S. in Computer Science at California State University, San Bernardino. The support of the Natural Science Foundation under the award 9810708 is also gratefully acknowledged.

Specifically, I have to thank Dr. David Turner and Dr. Arturo Concepcion. Both have given me tremendous support on this project and presented me with life changing opportunities. I could not have done five percent of this project if it were not for the selfless help of Dr. Turner.

I also need to thank all the staff in the Computer Science Department. This includes Monica Gonzales who gave all the advice on the staff part of this project, Ken Hahn who always helped with setting up my server and every other staff person. I always received help with a smile from everyone in the office regardless of the question.

My time here at California State University San Bernardino has been wonderful and changed my life for the better. I thank all who help to make this an excellent university.
# TABLE OF CONTENTS

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

CHAPTER ONE: INTRODUCTION

1.1 Purpose of This Project ................................................................. 1
1.2 History of Project ................................................................................. 2
1.3 Definitions of Terms .............................................................................. 3

CHAPTER TWO: SOFTWARE REQUIREMENTS SPECIFICATION

2.1 Introduction ............................................................................................... 5
2.2 Overall Description .................................................................................. 7
2.3 Specific Requirements ............................................................................... 14

CHAPTER THREE: MODEL VIEW CONTROLLER SOFTWARE ARCHITECTURE

3.1 System Design .......................................................................................... 33
3.2 Database Design ....................................................................................... 35
3.3 System Implementation ............................................................................ 43
3.4 Database Implementation ......................................................................... 51

CHAPTER FOUR: SYSTEM TESTING

4.1 Unit Test ................................................................................................. 54
4.2 System Test .............................................................................................. 55
4.3 Robustness Test ....................................................................................... 56

CHAPTER FIVE: MAINTENANCE MANUAL

5.1 Software Installation and Basic Configuration ........................................ 60
LIST OF TABLES

Table 1. Definition of Terms ........................................... 3
Table 2. Page Flow Diagram Key ......................................... 14
Table 3. Users Table ..................................................... 37
Table 4. Application Table ............................................. 37
Table 5. Education Record Table ....................................... 40
Table 6. Reference Table ................................................ 40
Table 7. Data Access Object Unit Tests ............................... 54
Table 8. System Test ..................................................... 56
Table 9. Robustness Test Results ...................................... 56
LIST OF FIGURES

Figure 1. Block Diagram of Application Process ...... 8
Figure 2. Deployment Diagram ......................... 9
Figure 3. Applicant Use Case Diagram .................. 9
Figure 4. Staff Use Case Diagram ....................... 10
Figure 5. Recommender Use Case Diagram ............. 10
Figure 6. Admin Use Case Diagram ...................... 11
Figure 7. Applicant Registration Page Flow ............ 15
Figure 8. Applicant Home Page and Form Flow ........ 16
Figure 9. Applicant Home Page ......................... 17
Figure 10. Contact Form ................................ 18
Figure 11. Applicant View Page Flow ................... 19
Figure 12. View Recommenders Page .................... 21
Figure 13. Recommender Form Page ..................... 22
Figure 14. Application Submit Page Flow .............. 23
Figure 15. Recommender Page Flow ..................... 24
Figure 16. Recommender Email .......................... 25
Figure 17. Reference Form ............................... 26
Figure 18. Staff Page Flow ............................... 27
Figure 19. Staff Home Page .............................. 28
Figure 20. View Application Page ....................... 29
Figure 21. Staff Application Page Flow ............... 30
Figure 22. Application Print and Accept Page Flow .... 31
Figure 23. Model-View-Controller Diagram ............ 33
CHAPTER ONE

INTRODUCTION

1.1 Purpose of This Project

The purpose of this project is to update the architecture and design of the California State University San Bernardino Graduate Advisor System. This system allows potential students to the Master of Science degree program in computer science to complete their application online. Currently the system relies heavily on Java Server Pages for most business logic and does not follow a rigid architecture. This project updates the architecture of the Graduate Advisor System to the Model-View-Controller pattern. By doing so, the main goal of the project, to build a generic architecture other computer science programs can use and build upon, has been achieved.

Spring, a cutting edge web framework, is used to implement this pattern. Hibernate is a new, widely accepted Object Relational Mapping tool which is used for database access for this project. All other components, including the operating system, have been updated to their most current version. A logging system, unit and Web tests, and a build tool, Ant, have been added.
The second most important purpose of this project is to improve on the user interface. The main idea behind improving the interface was simplicity. The system tries to be very clean, straight forward, and simple. For all users, the system is now easier and hopefully less time consuming to use.

Great care was taken in tailoring the interface to the user. Hopefully this project results in a cleaner, more exciting, system which will attract more applicants to the master’s degree program in computer science.

1.2 History of Project

The Graduate Advisor System was first developed in Dr. Arturo Concepcion’s CSCI655 Software Engineering class in 2001 at California State University San Bernardino. It was then taken over by a graduate student who continued work on it until it was released into production. In 2003, the author migrated the system from the Windows NT operating system to Redhat Linux. He also added the Ant build tool for developing purposes and a secure socket layer for security of personal data. After this, Kyle Rotte and Tiffany Chang worked on new versions of the program. This project is a full revision of the system.
1.3 Definitions of Terms

Table 1 displays the definitions of terms which will be used throughout this document.

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>CD</td>
<td>Compact Disc</td>
</tr>
<tr>
<td>CS</td>
<td>Computer Science</td>
</tr>
<tr>
<td>CSS</td>
<td>Cascading Style Sheets</td>
</tr>
<tr>
<td>CSUSB</td>
<td>California State University San Bernardino</td>
</tr>
<tr>
<td>GRADS</td>
<td>Graduate Advisor System</td>
</tr>
<tr>
<td>HTML</td>
<td>HyperText Markup Language</td>
</tr>
<tr>
<td>JSP</td>
<td>Java Server Page</td>
</tr>
<tr>
<td>JSTL</td>
<td>Java Standard Template Library</td>
</tr>
<tr>
<td>MS</td>
<td>Master of Science</td>
</tr>
<tr>
<td>MVC</td>
<td>Model-View-Controller pattern</td>
</tr>
<tr>
<td>ORM</td>
<td>Object/Relational Mapping</td>
</tr>
<tr>
<td>PDF</td>
<td>Portable Document Format (Adobe)</td>
</tr>
<tr>
<td>Staff</td>
<td>California State University San Bernardino Computer</td>
</tr>
<tr>
<td></td>
<td>Science Department Staff</td>
</tr>
<tr>
<td>-------</td>
<td>-------------------------</td>
</tr>
<tr>
<td>SLQ</td>
<td>Standard Query Language</td>
</tr>
<tr>
<td>URL</td>
<td>Uniform Resource Locator</td>
</tr>
<tr>
<td>WAR</td>
<td>Web Archive</td>
</tr>
<tr>
<td>XML</td>
<td>Extendable Markup Language</td>
</tr>
</tbody>
</table>
CHAPTER TWO
SOFTWARE REQUIREMENTS SPECIFICATION

2.1 Introduction

The purpose of this chapter is to describe the intent, functionality, and general interface of the Graduate Advisor System software product. GRADS is designed based on the requirements set by California State University of San Bernardino Computer Science professors Dr. David Turner, chair Dr. Arturo I. Concepcion and Dr. Kerstin Voigt herein referred to as "Customer". Concisely this chapter reflects the agreement between the Customer and the supplier of the GRADS project, defining GRADS' interfaces and functionality, through detailed page flow diagrams adjoined with thorough explanations of the products interfaces. Communications between the Customer and supplier have been maintained at the highest possible level to facilitate the insurance of a quality product being delivered to the Customer.

2.1.1 Scope

GRADS is a Web based application designed using the object-oriented methodology to provide an online application for the MS degree program in computer science at CSUSB. This product is designed only for the purpose of automating the application process. System input will only consist of data
prevalent to applying to the MS program and courses in the program.

The goal of GRADS is to efficiently automate the application process to the MS program for the applicants, their recommenders and Department of Computer Science staff. It will provide a paperless application process to be displayed in simple, easy to use forms.

The GRADS project will be used by the following users:

1. **Staff** - Individuals within the Department of Computer Science at CSUSB in charge of the logistical aspects of completed and uncompleted applications to the MS program. These individuals will be referred to as "staff" for the remainder of this document.

2. **Applicant** - Individuals who are prospective students to the MS program in computer science at CSUSB who are looking to apply to the program.

3. **Recommender** - Individuals specified by the Applicant who will provide a recommendation for the Applicant.

4. **System Administrator** - will have full program access privileges to create, edit and view Staff accounts.
2.2 Overall Description

2.2.1 Product Perspective

Physically, GRADS is a self contained system which resides on its own server at the Department of Computer Science. It is, however, part of the larger CSUSB computer science department network. Users can access it via this network. A link is provided on the computer science website providing initial access.

Systematically, GRADS is a major component of the application process for the MS program in computer science. An applicant is able to complete the entire application form and allow their recommenders to submit letters of reference via the system. They will still have to have transcripts of records and GRE scores sent to the department to complete their application.

Staff will be able to receive applications and recommendations submitted via GRADS. They will also be instantly notified of submitted applications via email. GRADS does not handle any of the evaluation process of the application.

2.2.1.1 System Interfaces. Figure 1 illustrates the entire application process to the MS program in computer science. As the diagram shows, GRADS can handle a significant percentage of processes. The solid boxes are
functions GRADS can process and the dashed boxes remain unchanged.

Figure 1. Block Diagram of Application Process

Figure 2 is the deployment diagram which shows all the hardware and software systems needed to run GRADS on a server and for the client to access GRADS.
2.2.1.2 User Interfaces. Figures 3, 4, 5 and 6 show the logical interface of each user of GRADS respectively.
Figure 4. Staff Use Case Diagram

Figure 5. Recommender Use Case Diagram
2.2.1.3 Hardware Interfaces. The only hardware GRADS will interface with is the Microsoft Access database the Department of Computer Science maintains for student records. This will be indirect access through the staff. If an applicant using GRADS has been successfully admitted to the MS program, staff will be able to download a SQL script generated by the system. They can then import this script into the Microsoft Access database.

2.2.1.4 Software Interfaces. GRADS will require all of its users to access the system with a compatible Web browser. GRADS has been tested to work with Microsoft Internet Explorer 5.0 and higher and Netscape 6.0.

2.2.1.5 Communication Interfaces. GRADS will be sending notification emails to all users except the admin. The system communicates with the CSUSB mail server “gw.csusb.edu” in order to send mail securely.
2.2.1.6 Memory Constraints. GRADS does not have any known memory constraints.

2.2.2 Product Functions

The main function of GRADS is to provide a Web based version of the application to CSUSB computer science department's MS program for prospective applicants. All information required in a hardcopy version of the application can be completed via html forms and stored in a database. GRADS will also automate most of the application process. Automatic notification to recommenders allows a given recommender to complete a recommendation via GRADS. Staff can accept the application through GRADS and download a SQL script to import the application data into the computer science Microsoft Access database. Details of these functions will be found in section three of this chapter.

2.2.3 User Characteristics

All users of GRADS will need to possess the knowledge of using a Web browser listed in paragraph 2.2.1.4. As noted in section 2.2.1.2 the users of GRADS are the applicant, recommender, staff, and administrator.

The applicant is simply a prospective applicant to the MS in Computer Science at CSUSB. They should be familiar with the logistics of the application process, including the separate application to CSUSB for admission to the
University. GRADS assumes the applicant is aware of all application requirements.

The recommender is typically a former teacher or employer of the applicant chosen by the later to submit a letter of reference for their application. GRADS recommends to the applicant that he or she inform the recommender of the reference request prior to using GRADS so they will be expecting this request. The recommender only has one Web form to submit, so they are expected to have just the minimum skills of using a Web browser.

The staff user is expected to be an employee of the Department of Computer Science who is completely familiar with the MS in Computer Science application process. They should know all the requirements of the application and all the types of its status. Staff is also expected to have moderate training on the use of GRADS.

There is only one administrator user. This person typically would be a computer science student currently working and maintaining GRADS. The administrator is expected to have an extensive computer science background and familiarity with the implementation of GRADS.

2.2.4 Constraints

Constraints have been placed on the system by the customer for security reasons. File upload for the
application statement of purpose and letter of reference is not included in the system on the basis that its infrequent use does not justify the possibility of malicious file uploads.

2.3 Specific Requirements

2.3.1 External Interfaces

The external interface of GRADS is self contained, simple, and user friendly. Each page flow diagram, figures 7 through 21, will be followed by an explanation of the user interface. Table 2 shows the key for these diagrams.

Table 2. Page Flow Diagram Key

<table>
<thead>
<tr>
<th>Link</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[link]</td>
<td>Html link</td>
</tr>
<tr>
<td>Input:</td>
<td>Html input (text, select, radio, or text area)</td>
</tr>
<tr>
<td>![Button]</td>
<td>Html submit button</td>
</tr>
<tr>
<td>![Submit and Link Action]</td>
<td>Submit and link action, i.e. page flow</td>
</tr>
<tr>
<td>General Description</td>
<td>Static content</td>
</tr>
</tbody>
</table>

2.3.1.1 Applicant. Figures 6 through 13 detail the external interface for the applicant using GRADS.
Figure 7 illustrates the create account screen flow. The user will choose a user name and password. The user will have to confirm the password. The password must contain a minimum of six characters. The user will also have to supply a valid email address and their first and last name. Any validation error will send the user back to the register form with an appropriate error message. Upon submission of all these required fields, the user will be automatically logged in to GRADS with the role of applicant and taken to a welcome page which will provide static information on how they can use the system. When the user clicks "continue->" they will be taken to the applicant home page.
Figure 8 shows the applicant’s home page and illustrates the form page flow for the contact, general, biography and statement of purpose form pages. Figures 9 and 10 show the screen shots for the applicant home page and contact form page. The application form for the MS degree in computer science is divided into six pages in GRADS: general information, contact information, biography, education, tests, statement of purpose, and recommenders. Each page will have a link menu to each of the six pages. Four of the six (general, contact, biography, and statement of purpose) are form pages. Form pages consist of html text fields which will be pre-filled with applicant inputted data if
previously filled out and saved or blank if the applicant is new to the page. If the user clicks the “Save and Continue” button, GRADS will validate all form text fields. If empty or there is an error, then the user will be sent back to the same form page with error messages. If there is successful validation, the user will be taken to the next page as shown in the application menu. The data entered will be saved to the database.

![GRADS GRADUATE Adviser System](image)

Figure 9. Applicant Home Page
If a user clicks one of the links in the application menu, they will be taken straight to that page. No validation or saving will be done at this point.
Two of the six application pages, recommenders and education pages, are considered view pages. Figure 11 illustrates the recommender page which shares the same flow as the education page. Figures 12 and 13 show the corresponding screen shots of the recommender page flow. The applicant will get to this page either through the normal progression of the application process or directly through the application menu link.

Looking at the recommender page as an example, the applicant will first see a list of previously submitted recommenders along with their status, i.e. recommender has been notified or reference has been submitted. If no
recommender has been submitted a message will be displayed telling them they will need to add at least three recommenders.

They can do this by clicking the "Add Recommender" button. This will take them to the recommender form page. Here the applicant will enter contact information for the recommender and indicate whether the recommender will submit their recommendation through GRADS, i.e. online. If the recommender is online, then their valid email is required. Upon successful validation the applicant will then be taken back to the recommenders page. If the recommender has been indicated as online, an email notification will be sent immediately to him or her allowing them to submit their recommendation.

All submitted recommenders will be listed on recommenders page as links. If the applicant clicks one of the links, he or she will be taken to the recommender form which will be pre-filled with the recommender values. This will allow the applicant to edit any recommender information. No notification email will be sent to the recommender upon successful submission of the form.

The applicant can also send a reminder email or delete any previously inputted recommenders. Clicking any check box next to a recommender and then clicking the "Delete" or
"Resend Email" button will achieve this. The delete function will bring up a confirmation page and the resend email function will bring the applicant back to the same page with a message telling them an email has been sent.

![GRADS Graduate Advisor System](image)

**Figure 12. View Recommenders Page**
Figure 13. Recommender Form Page

Figure 14 illustrates the application submission page flow for the applicant. When the applicant has successfully completed all six application pages, the status of the application menu will display a message and button that the application is ready for submission. If the applicant clicks this button they will be taken to a confirmation page relaying the importance of formally submitting the application. The main point is that upon formal submission, the applicant will no longer be able to edit their application.
If they formally submit, the applicant will then be taken back to their home page with a thank you message. The home page will now change from the general form page to the recommenders page. The applicant will now only be able to view their application except for the recommenders page.

Figure 14. Application Submit Page Flow

2.3.1.2 Recommender. Figure 14 represents the external interfaces for the recommender using GRADS.
Email Sent To Recommender

From: CSUSB Computer Science Dept
To: recommender

Hello Mr. Recommender,
Joe Applicant has named you as a recommender for his/her application to the Master's degree program in CS at CSUSB. Please follow the link below to complete your recommendation.

[http://www.grads.com?key=123456]

Thank you,
Grads

Reference Form

Applicant Name: Joe App
First Name: 
Last Name: 
Address: 
Rate Intellect: 
Rate Teaching: 
Submit

Validation Error

Thank You Page

Thank you for your letter of reference!
The link we provided will no longer be active.

Figure 15. Recommender Page Flow

Figure 15 illustrates the simplicity of the recommender page flow. When the applicant submits a recommender to GRADS as being online, the system will send an email to the recommender. This will include a unique link which when clicked on will take the recommender straight to their reference form. This form is a letter of reference questionnaire for the applicant. The recommender has the choice of filling this form out and submitting it or selecting that they will email in their own custom letter. Either way, upon successful submission the recommender will be sent to a page which thanks them and informs them that the reference link they have accessed will no longer be valid. At this point, a notification email will be sent to the applicant. Figure 16 and 17 display corresponding screen shots of figure 15.
Dear Dr. Turner,

X x has named you as a recommender to his application for admission to the Master of Computer Science program at California State University San Bernardino. You can submit your recommendation online by following the link below. This link will remain active until your submission.

http://web1.csusb.edu:8080/ springgrades/visitor/reference?key=541280034bb661b56692ba79125c861

If you have any questions, please feel free to contact me at (909) 880-5326 or mgonzales@csd.csusb.edu

Regards,

Monica Gonzales

Figure 16. Recommender Email
2.3.1.3 Staff. Figures 18 through 22 illustrate the external interfaces for staff using GRADS.
Figure 18. Staff Page Flow

Figure 18 illustrates the page flow for a typical staff interaction. Staff can access their home page either through the login page or a direct link inside a new application email notification. The home page, shown in figure 18, is a list of all applications in GRADS sorted by three categories. Submitted and complete means the application is submitted and all online recommenders have submitted their references. Submitted and not complete means the application has been submitted, however not all online recommenders have submitted their references. Incomplete lists all
applications which have not been formally submitted. Each application in this list is a link that will take staff to the application page. The application page lists all the details for the application and is shown figure 20. Staff can also get to the application page by clicking the link in the applicant submission notification email referenced in section 2.3.1.2. If staff is logged into GRADS already, they will be taken straight to this page and if not logged in they will be required to log in first.

Figure 19. Staff Home Page
Figure 20. View Application Page
Figure 21 illustrates the edit and delete functions staff can use from the application page. The application is broken down into six sections as detailed in section 2.3.1.1. Each one of these sections which will have an edit link next to it. When clicked, staff will be taken to the corresponding form page with application data pre-filled from the database. Staff can then edit this information with normal validation rules of GRADS.

From the application page, recommenders names will be links to the corresponding reference page listing all recommenders contact information and reference. Staff can go to the reference form page by clicking the edit link. The
page flow for editing the reference is the same as editing the application.

To delete a reference of the entire application, staff can click the delete button which will take them to a confirmation page to double check this deletion.

Figure 22. Application Print and Accept Page Flow

From the application page, staff will have the ability to print any reference form or the application form. This
would be done by clicking the appropriate print link. This would generate either of the forms in PDF.

Staff can also accept an application by clicking the "Accept Application" link. Accepting an application means that it is now completed and ready to be evaluated for acceptance into the master's degree program in computer science. When this link is clicked and confirmed, the applicant will be sent an email notification informing them their application is being evaluated. The confirmation page will list any remaining items still needed for this application. The application will also be taken off of the list from the home page of staff, but will not be removed from the database.

2.3.2 Functional Requirements

GRADS shall have the following functional requirements:

1. Ability to enter, store and retrieve all data.
2. Validate all necessary inputs.
3. Ability to run remotely through the HTTP protocol.
4. Respond to errors with appropriate messages.

2.3.3 Performance Requirements

GRADS can handle up to thirty requests per minute with a maximum response time of two seconds.
CHAPTER THREE
MODEL VIEW CONTROLLER SOFTWARE ARCHITECTURE

3.1 System Design

GRADS is developed using the JSP Model Two Architecture. This is also referred to as the Model-View-Controller (MVC) pattern. Figure 23 displays the MVC pattern for GRADS. Client requests are first intercepted by the “controller” servlet. The servlet is implemented through the Spring framework which is detailed later in this document. The Spring controller servlet handles the processing of the request. The controller allows for front-end processing. This includes authentication, database access, and logging. The “model” is a Java Bean object which is instantiated and populated by the controller. Typically this is set in the request and forwarded to the next JSP to display to the user. This is the “view” of the MVC pattern.

![Model-View-Controller Diagram](image)

Figure 23. Model-View-Controller Diagram
3.1.1 Controller

GRADS uses the Spring framework as its controller. The design of Spring is called Inversion of Control. For the controller, this means that configuration of views and controls are done within the framework inside of XML files. The model and other collaborators of the different controller servlets are set as Java Bean properties. This allows the model and collaborators to be provided to the controller directly at runtime.

3.1.2 Model

The model design of GRADS is also defined by the Spring framework. All models are Java Beans. They are either business objects or data transfer objects. Business objects handle database access and business functions. Typically these will populate the data transfer objects. These are Java Beans with getter and setter methods and are used to display dynamic content within the view.

3.1.3 View

GRADS uses JSP and HTML for the design of the view. HTML pages are used to serve static content. JSP pages serve both static and dynamic content. The JSP pages display models in the GRADS architecture.
3.1.4 Web Tier

GRADS is considered a two tier application. The two tier application consists of a client tier and application tier. The client tier for this system is any of the users' web browsers. The web tier consists of the web container, database and all components which allow the web tier to communicate with the client tier. Figure 24 shows GRADS' as a two tier application.

![Two Tier Application Diagram](image)

3.2 Database Design

As detailed in chapter two, GRADS has three different types of users, applicant, recommender, and staff. The applicant has an application and the recommender has a letter of reference. GRADS' database design converts the
application and letter of reference into database entities. Because there can be multiple education records in the application, this also becomes a database entity. Figure 24 shows an overview of the entity relationships of GRADS.

Figure 25. Entity Relationship Diagram

Tables 3 through 6 map out the entire database design for GRADS. Every user has an entry in the User table which consists of their username, password, email and role. The username is a unique java string. The application’s primary key, username, corresponds to their entry in the User table. All other tables use a unique Long id and have foreign keys to their corresponding tables.
### Table 3. Users Table

<table>
<thead>
<tr>
<th>FIELD NAME</th>
<th>SQL TYPE</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>username</td>
<td>varchar(255)</td>
<td>Unique username.</td>
</tr>
<tr>
<td></td>
<td>primary key</td>
<td></td>
</tr>
<tr>
<td>Role</td>
<td>varchar(15) not null</td>
<td>applicant, recommender, staff, or admin.</td>
</tr>
<tr>
<td>password</td>
<td>varchar(255)</td>
<td>minimum 6 chars.</td>
</tr>
<tr>
<td>password_key</td>
<td>varchar(255)</td>
<td>used for forgot password function</td>
</tr>
<tr>
<td>email</td>
<td>varchar(255)</td>
<td>valid email.</td>
</tr>
<tr>
<td>discriminator</td>
<td>varchar(255)</td>
<td>used by hibernate for inheritance.</td>
</tr>
</tbody>
</table>

### Table 4. Application Table

<table>
<thead>
<tr>
<th>FIELD NAME</th>
<th>SQL TYPE</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>username</td>
<td>varchar(255)</td>
<td>Foreign key to Users table.</td>
</tr>
<tr>
<td></td>
<td>primary key</td>
<td></td>
</tr>
<tr>
<td>title</td>
<td>varchar(5)</td>
<td>Mr, Mrs, etc.</td>
</tr>
<tr>
<td>Last_name</td>
<td>varchar(255)</td>
<td>student's last name.</td>
</tr>
<tr>
<td>Column</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------</td>
<td>-----------</td>
<td>---------------------------------</td>
</tr>
<tr>
<td>first_name</td>
<td>varchar(255)</td>
<td>student’s first name.</td>
</tr>
<tr>
<td>phone1</td>
<td>varchar(255)</td>
<td>primary phone</td>
</tr>
<tr>
<td>phone2</td>
<td>varchar(255)</td>
<td>secondary phone</td>
</tr>
<tr>
<td>address1</td>
<td>varchar(255)</td>
<td>street address</td>
</tr>
<tr>
<td>address2</td>
<td>varchar(255)</td>
<td>apt number, etc.</td>
</tr>
<tr>
<td>City</td>
<td>varchar(255)</td>
<td></td>
</tr>
<tr>
<td>state</td>
<td>varchar(255)</td>
<td></td>
</tr>
<tr>
<td>Zip</td>
<td>varchar(255)</td>
<td>postal code</td>
</tr>
<tr>
<td>county</td>
<td>varchar(255)</td>
<td></td>
</tr>
<tr>
<td>country</td>
<td>varchar(255)</td>
<td></td>
</tr>
<tr>
<td>office_name</td>
<td>varchar(255)</td>
<td>student’s employee</td>
</tr>
<tr>
<td>office_address1</td>
<td>varchar(255)</td>
<td></td>
</tr>
<tr>
<td>office_address2</td>
<td>varchar(255)</td>
<td></td>
</tr>
<tr>
<td>office_city</td>
<td>varchar(255)</td>
<td></td>
</tr>
<tr>
<td>office_state</td>
<td>varchar(255)</td>
<td></td>
</tr>
<tr>
<td>office_zip</td>
<td>varchar(255)</td>
<td></td>
</tr>
<tr>
<td>office_country</td>
<td>varchar(255)</td>
<td></td>
</tr>
<tr>
<td>office_phone</td>
<td>varchar(255)</td>
<td></td>
</tr>
<tr>
<td>office_phone_ext</td>
<td>varchar(255)</td>
<td>phone extension</td>
</tr>
<tr>
<td>Year</td>
<td>varchar(8)</td>
<td>year student is applying for</td>
</tr>
<tr>
<td>Column Name</td>
<td>Data Type</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>----------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>quarter</td>
<td>varchar(8)</td>
<td>quarter student is applying for</td>
</tr>
<tr>
<td>citizenship</td>
<td>varchar(255)</td>
<td>citizenship of student</td>
</tr>
<tr>
<td>birth_date</td>
<td>date</td>
<td></td>
</tr>
<tr>
<td>birth_city</td>
<td>varchar(255)</td>
<td></td>
</tr>
<tr>
<td>birth_state</td>
<td>varchar(255)</td>
<td></td>
</tr>
<tr>
<td>birth_county</td>
<td>varchar(255)</td>
<td></td>
</tr>
<tr>
<td>birth_country</td>
<td>varchar(255)</td>
<td></td>
</tr>
<tr>
<td>gre_exam_date</td>
<td>date</td>
<td>date GRE test is taken</td>
</tr>
<tr>
<td>gre_verbal</td>
<td>varchar(255)</td>
<td>GRE verbal score</td>
</tr>
<tr>
<td>gre_quantitative</td>
<td>varchar(255)</td>
<td>GRE quantitative score</td>
</tr>
<tr>
<td>gre_analytical</td>
<td>varchar(255)</td>
<td>GRE analytical score</td>
</tr>
<tr>
<td>toefl_exam_date</td>
<td>date</td>
<td>date TOEFL is taken</td>
</tr>
<tr>
<td>toefl_score</td>
<td>varchar(255)</td>
<td>TOEFL score</td>
</tr>
<tr>
<td>Sop</td>
<td>text</td>
<td>statement of purpose</td>
</tr>
<tr>
<td>create_date</td>
<td>date</td>
<td>date user account created</td>
</tr>
</tbody>
</table>
## Table 5. Education Record Table

<table>
<thead>
<tr>
<th>FIELD NAME</th>
<th>SQL TYPE</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>Id</td>
<td>integer, primary key</td>
<td>unique id, java Long value</td>
</tr>
<tr>
<td>application_id</td>
<td>varchar(255), foreign key</td>
<td>foreign key to application username</td>
</tr>
<tr>
<td>school</td>
<td>varchar(255)</td>
<td>school name</td>
</tr>
<tr>
<td>years_from</td>
<td>varchar(255)</td>
<td>year school started</td>
</tr>
<tr>
<td>years_to</td>
<td>varchar(255)</td>
<td>year school ended</td>
</tr>
<tr>
<td>Gpa</td>
<td>varchar(255)</td>
<td>grade point average</td>
</tr>
<tr>
<td>degree</td>
<td>varchar(255)</td>
<td></td>
</tr>
</tbody>
</table>

## Table 6. Reference Table

<table>
<thead>
<tr>
<th>FIELD NAME</th>
<th>SQL TYPE</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>Id</td>
<td>integer, primary key</td>
<td>unique id</td>
</tr>
<tr>
<td>application_id</td>
<td>varchar(255)</td>
<td>references</td>
</tr>
<tr>
<td>Field</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>------------</td>
<td>-------------------------------------------------------</td>
</tr>
<tr>
<td>foreign key</td>
<td>application username</td>
<td></td>
</tr>
<tr>
<td>the_key</td>
<td>varchar(255)</td>
<td>unique key required for recommender to access reference</td>
</tr>
<tr>
<td>title</td>
<td>varchar(5)</td>
<td>Mr, Mrs, etc.</td>
</tr>
<tr>
<td>Last_name</td>
<td>varchar(255)</td>
<td>recommender's last name</td>
</tr>
<tr>
<td>first_name</td>
<td>varchar(255)</td>
<td>recommender's first name</td>
</tr>
<tr>
<td>phone</td>
<td>varchar(255)</td>
<td>phone number</td>
</tr>
<tr>
<td>address1</td>
<td>varchar(255)</td>
<td>street address</td>
</tr>
<tr>
<td>address2</td>
<td>varchar(255)</td>
<td>apt number, etc.</td>
</tr>
<tr>
<td>City</td>
<td>varchar(255)</td>
<td></td>
</tr>
<tr>
<td>state</td>
<td>varchar(255)</td>
<td></td>
</tr>
<tr>
<td>Zip</td>
<td>varchar(255)</td>
<td>postal code</td>
</tr>
<tr>
<td>office_name</td>
<td>varchar(255)</td>
<td>recommender's employee</td>
</tr>
<tr>
<td>office_position</td>
<td>varchar(255)</td>
<td>recommender's work title</td>
</tr>
<tr>
<td>relationship</td>
<td>varchar(255)</td>
<td>relationship to student</td>
</tr>
<tr>
<td></td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>----------------</td>
<td>-----------------------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>length_known</td>
<td>varchar(255)</td>
<td></td>
</tr>
<tr>
<td>notify_date</td>
<td>date</td>
<td></td>
</tr>
<tr>
<td>submit_date</td>
<td>date</td>
<td></td>
</tr>
<tr>
<td>online</td>
<td>tinyint</td>
<td></td>
</tr>
<tr>
<td>intellect</td>
<td>varchar(5)</td>
<td></td>
</tr>
<tr>
<td>imagination</td>
<td>varchar(5)</td>
<td></td>
</tr>
<tr>
<td>general</td>
<td>varchar(5)</td>
<td></td>
</tr>
<tr>
<td>Oral</td>
<td>varchar(5)</td>
<td></td>
</tr>
</tbody>
</table>

- **length_known**: varchar(255) - length recommender has known student
- **notify_date**: date - date notification email is sent to recommender
- **submit_date**: date - date recommender submitted reference through GRADS
- **online**: tinyint - converts to boolean in code. Online recommender or offline.
- **intellect**: varchar(5) - rate of student intellect
- **imagination**: varchar(5) - rate of student imagination
- **general**: varchar(5) - rate of student general ability
- **Oral**: varchar(5) - rate of student oral skills
<table>
<thead>
<tr>
<th>writing</th>
<th>varchar(5)</th>
<th>rate of student writing skills</th>
</tr>
</thead>
<tbody>
<tr>
<td>maturity</td>
<td>varchar(5)</td>
<td>rate of student maturity</td>
</tr>
<tr>
<td>teacher</td>
<td>varchar(5)</td>
<td>rate of student teacher potential</td>
</tr>
<tr>
<td>professional</td>
<td>varchar(5)</td>
<td>rate of student as a professional</td>
</tr>
<tr>
<td>overall</td>
<td>varchar(5)</td>
<td>rate of student overall</td>
</tr>
<tr>
<td>comments</td>
<td>text</td>
<td>optional comments to staff</td>
</tr>
</tbody>
</table>

### 3.3 System Implementation

Apache Jakarta Tomcat version 5.0.27 is the backbone of GRADS' system implementation. Tomcat is a servlet container in the web tier of the application. It provides a hosting environment for the GRADS' MVC architecture. The MVC architecture is implemented through the Spring framework, version 1.1.2. The framework provides components which interact with the Java API, Java Standard template library, Java Mail, Log4J, and Hibernate. All of these combined
implement the MVC architecture of GRADS. Figure 25 shows an overview of this implementation.

![Deployment Diagram](image)

**Figure 26. Deployment Diagram**

### 3.3.1 Spring Framework

GRADS uses five of seven Spring modules to implement its architecture. Figure 27 illustrates the usage scenario of these five modules. The core module provides a bean factory which allows for most of the configuration of GRADS to take place within XML files. This decouples the configuration from the business logic. On top of the core module is the context module which allows for access in code to the configured beans. GRADS uses this extensively for validation. The web module provides basic utilities to
access HTTP request parameters and attributes. The web MVC module implements the controller function for GRADS. The ORM module allows Hibernate mapping to the database. This will be discussed in the next section.

![Diagram](image)

**Figure 27. Usage Scenario**

### 3.3.2 View Implementation

GRADS uses the Spring web and core modules, JSP pages, CSS and JSTL to implement the view. The web module contains a class, ResourceBundleViewResolver. This allows for all JSP files to be mapped into one XML file and read by the core package. GRADS extends the JstlView of Spring to allow for templating of JSP files. The JstlView then provides all the JSTL functions within the JSPs for dynamic content. Figure 27 shows the two views GRADS uses, plain view for staff and recommenders and fancy view for the applicant.
3.3.3 Model Implementation

GRADS uses Java beans for the model implementation. Two are defined. First, the regular beans are Java implementations of the data entities described in section 3.2. The second bean is the form bean. These are data transfer objects used in GRADS. A form bean contains its corresponding regular bean and added String attributes for conversion of Strings to other Java types. For example, a String from a form needs to be converted to a Date object. Figure 29 is a class diagram showing the models of GRADS.
3.3.4 Controller Implementation

GRADS relies on three different classes of the Spring framework to implement the controller. These are the Controller, MultiActionController, and SimpleFormController.
3.3.4.1 Controller. GRADS implements the Controller interface of Spring when database access is necessary to return a model and view. The handleRequest function returns a Spring ModelAndView object which is the combination of the view and model. In the example in figure 30, ViewAccountController implements this function. The UserService provides model data from the database.

```
<<interface>>
org.springframework.web.servlet.mvc.Controller

 VIEWACCOUNTCONTROLLER

+handleRequest(request:HttpServletRequest,

UserService userService

Figure 30. Controller Class Diagram
```

3.3.4.2 MultiActionController. The MultiActionController is the same as the Controller except that it allows for multiple functions to be declared in a single file each returning a different ModelAndView. GRADS uses this for returning views and models that are all related. For example, figure 31 shows the class diagram for
StaffViewsController. This controls the generation of views for a single staff or multiple staffs.

![Diagram of StaffViewsController](image)

Figure 31. MultiAction Controller Class Diagram

3.3.4.3 AbstractParentFormController. GRADS inherits the Spring SimpleFormController with the creation of the AbstractParentFormController class. Form submission pages within GRADS inherit from this controller. Basically the form view is provided a model object by the formBackingObject function. If this is an edit form, then the model is populated through the database. Upon submission, the SimpleFormController calls its validate function to check all the data. If the form passes validation, then the onSubmit function is called. This saves
the form data to the database and returns the success view.
Figure 31 shows an example class diagram and figure 32 shows a sequence diagram for an implementation of the AbstractParentFormController.

Figure 32. AbstractParentController Class Diagram
3.4 Database Implementation

GRADS uses the MySQL version 4.0.20 database. The Spring framework version 1.1.4 and the Hibernate version 2.1 ORM to implement the models access to the database.

3.4.1 MySQL

GRADS uses a JDBC datasource to connect to the MySQL database. The datasource is configured as a bean within a Spring configuration file.

3.4.2 Hibernate

Hibernate is an object/relational mapping between the business and model object in GRADS to the database. It generates its own SQL statements for retrieval and query. Configuration files are used to map database tables to the models detailed in section 3.3.3.
3.4.3 Spring and Hibernate

Spring allows GRADS to configure Hibernate sessions as beans in the application context file. Business objects get references to these beans to access data from the database. GRADS uses a Spring class, HibernateDaoSupport, which manages the Hibernate sessions automatically and provides a very simple way to make database calls.

GRADS breaks down database access into three parts. On the top level are service objects. These are used within controllers to retrieve required data and to decouple business logic out of the controllers. Figure 34 shows an example of this with the ReferenceService. The ReferenceService has its own ReferenceDAO. The DAO is the second layer of the GRADS database access. These are interfaces which, through Spring configuration, decouple the DAOs from the implementation. This makes it easier for any other ORM to be used with GRADS. No changes would have to be made to the DAO; only a new implementation would need to be added. The third level is the ReferenceDAOImpl which is the implementation of ReferenceDAO. This is the class which inherits functions from the Spring class, HibernateDaoSupport.
Figure 34. Reference Service Class Diagram
CHAPTER FOUR

SYSTEM TESTING

The system uses two different types of testing, unit test and system testing. Both tests showed GRADS to be highly reliable.

4.1 Unit Test

GRADS uses JUnit version 3.8.1 for its unit testing. JUnit is a testing framework which the system used for all unit tests. A unit is a DAO object. The units tests were all deployed through the Ant build tool. This allowed for regressive tests of all units even with the smallest code changes. For example, after any code changes the developer only needs to run the command “ant unittest” and the test will be performed instantly. Code was fixed until all unit tests performed one hundred percent correct. Table 7 shows the results of the unit tests.

Table 7. Data Access Object Unit Tests

<table>
<thead>
<tr>
<th>Unit</th>
<th>Tests</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>ApplicationDAO</td>
<td>1. String username#1=app#1.getUsername()</td>
<td>Pass</td>
</tr>
<tr>
<td></td>
<td>2. saveOrUpdate(app#1)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3. app#2=find(username)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>4. String username#2 = app#2.getUsername()</td>
<td></td>
</tr>
<tr>
<td></td>
<td>5. assertEquals(username#1, username#2)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>6. delete(app#1)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>7. app#1=find(app#1.getUsername())</td>
<td></td>
</tr>
</tbody>
</table>
8. `assertNull(app#1)`

| EdRecord | 1. `Long id#1=edRecord#1.getId()`  
| 2. `saveOrUpdate(edRecord#1)`  
| 3. `edRecord#2=find(id#1)`  
| 4. `Long id#2=edRecord#2.getId()`  
| 5. `assertEquals(id#1, id#2)`  
| 6. `delete(edRecord#1)`  
| 7. `edRecord=find(edRecord#1.getId())`  
| 8. `assertNull(edRecord)`  | Pass |

| ReferenceDAO | 1. `Long id#1=ref#1.getId()`  
| 2. `saveOrUpdate(ref#1)`  
| 3. `ref#2=find(id#1)`  
| 4. `Long id#2=ref#2.getId()`  
| 5. `assertEquals(id#1, id#2)`  
| 6. `delete(ref#1)`  
| 7. `edRecord=find(ref#1.getId())`  
| 8. `assertNull(ref)`  | Pass |

| UserDAO | 1. `String username#1=user#1.getUsername()`  
| 2. `saveOrUpdate(user#1)`  
| 3. `user#2=find(username)`  
| 4. `String username#2 = user#2.getUsername()`  
| 5. `assertEquals(username#1, username#2)`  
| 6. `delete(user#1)`  
| 7. `user#1=find(user#1.getUsername())`  
| 8. `assertNull(user#1)`  | Pass |

4.2 System Test

System testing is the testing process that uses real data to test the system. All components of the system will be integrated together. After the entire system is up and running a complete web test is run. Table 9 shows the result of the system test.
4.3 Robustness Test

Robustness testing of the system consisted of inputting erroneous data into forms. Examples would be entering an invalid date into a date field or malicious code into these fields. The system should not crash under these conditions, but should gracefully show an error message. Validation of these fields were found to reject all of these attempts. Table 10 shows the robustness test results.

Table 9. Robustness Test Results

<table>
<thead>
<tr>
<th>Test</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Invalid date value inputted into date field</td>
<td>Pass - upon submission, user returned to previous screen with error message displaying that date needs to be in valid format.</td>
</tr>
<tr>
<td>Invalid email value inputted into email field</td>
<td>Pass - upon submission, user returned to previous screen with error message displaying that date needs to be in valid format.</td>
</tr>
<tr>
<td>Invalid parameters passed through URL</td>
<td>Pass - user logged out of system with error message displaying</td>
</tr>
<tr>
<td>Code inserted into String parameters</td>
<td>illegal access.</td>
</tr>
<tr>
<td>--------------------------------------</td>
<td>-----------------</td>
</tr>
<tr>
<td>Pass - Code will not be run through browser because of JSTL “c:out” function security.</td>
<td></td>
</tr>
</tbody>
</table>

Figures 35 through 38 are screen shots of the above robustness tests.

![Figure 35. Invalid Date Example](image_url)
Please fix errors!

My Account

Username: x
*Email: rpallow@hotmail
  Invalid email
*Email(ReType): rpallow@hotmail.com
  Emails do not match

Save  Cancel

Figure 36. Invalid Email Example

HTTP Status 403 - Access to the requested resource has been denied

TYPE Status report
message Access to the requested resource has been denied
description Access to the specified resource (Access to the requested resource has been denied) has been forbidden.

Figure 37. Invalid Parameter Example
Statement of Purpose

*Write a brief statement(no more than 400 words) on your reasons for pursuing the M.S. Degree in Computer Science. Include any additional information concerning your preparation pertinent to the M.S. Degree.

```javascript
<script type="javascript">
alert('Hello World');
</script>
```

Figure 38. Code Inserted Example
CHAPTER FIVE
MAINTENANCE MANUAL

The following manual is a step by step guide for installing GRADS on a single server. Network configurations are not part of this manual. All software is included with the maintenance CD included.

5.1 Software Installation and Basic Configuration

GRADS uses the Gentoo operating system, Java SDK, Java mail, MySQL database, Ant build tool, Log4J logging tool, Tomcat container, Spring framework, and Hibernate ORM tool. Below are general notes on installation for GRADS that would be in addition to the normal installation guidelines provided by the developers. Most software was installed via Gentoo.

5.1.1 Gentoo

A Gentoo Live CD is required for installation. This can be found at http://www.gentoo.org.

1. Install Gentoo per their instructions.

5.1.2 Java SDK

1. As root, type "emerge jdk" to install the Blackdown version of the Java development kit.
5.1.3 Java Mail

1. All that is needed for Java mail are the two files mail.jar and activation.jar. These can be found inside the download package at http://java.sun.com/products/javamail.

2. Place these in the classpath of the application.

5.1.4 MySQL

1. As root, type "emerge mysql".

2. Type "mysql_install_db --user=mysql" to create the grant tables.

3. Add mysql to the default run level by typing the following commands:
   "rc-update add mysql default"
   "/etc/init.d/mysql start".

4. Set the root password by typing "mysqladmin -u root password 'root-pass'".

5. Remove default accounts for security reasons by typing the following:
   "mysql -u root -p"
   "mysql> use mysql;"
   "mysql> delete from user where password = '';"
   "mysql> flush privileges;"

6. Create the "grads" database by typing the following:
   "mysql -u root -p"
"mysql> create database grads;"
"mysql> grant all on grads.* to username identified by 'password';"

5.1.5 Ant Build Tool
1. Type "emerge dev-java/ant"

5.1.5 Log4J
1. Download log4j distribution from
   http://jakarta.apache.org/log4j/docs/download.html
2. Copy the file dist/lib/log4j-1.2.6.jar to the classpath.

5.1.6 Tomcat
1. Type "emerge tomcat".
2. Set tomcat at run level by typing "rc-update add
tomcat5 default".
3. To allow GRADS to read tomcat jar files type "chmod
   755 /opt/tomcat5".
4. Download the JSTL jar files, jstl.jar and
   standard.jar from http://jakarta.apache.org/taglibs. Place
   these in tomcat/common/lib directory.

5.1.7 Spring Framework
1. Download spring.jar from http://springframework.org
   and place in classpath.
5.1.8 Hibernate

1. Get the JDBC driver by typing "emerge dev-java/jdbc-mysql".
2. Copy mysql-connector-java-3.0.11-stable-bin.jar to the /opt/tomcat/common/lib directory.

5.2 Installation and Configuration

All GRADS files are located in the zip file grads.tar.gz located in the maintenance CD. Unzip these file and follow the steps below to get GRADS up and running.

1. Edit the build.properties to so the variable “userPath” matches the path of the GRADS directory.
2. Edit conf/machines/web1/springgrads_context.xml so docBase reflects the proper directly along with the parameters log4j.properties and url.prefix.
3. As root, copy springgrads_context.xml from step 2 into /opt/tomcat5/conf/Catalina/localhost directory.
4. Edit conf/machines/web1/classes/log4j.properties so the variable log4j.appender.r.File equals the proper directory.
5. Edit web/WEB-INF/jdbc.properties to the username and password match the MySQL username and password.
6. Edit web/WEB-INF/web.xml so the parameter url.prefix matches the correct url.

5.3 Deployment

Deploy GRADS from the top level directory and typing "ant initdb start". This will deploy GRADS into the Tomcat container and create the database tables. An optional deployment method would be to type "ant initdb war". This will create the database tables and then create a WAR file. This can be placed into the Tomcat/webapps directory for deployment.

5.4 Hints for Maintenance

5.4.1 Configuration

The core of the GRADS configuration files are found under the web/WEB-INF directory. These are all XML files. Each controller servlet has its own XML file. These contain the controller mappings.

The application-hibernate.xml file in the WEB-INF directory contains datasource and business object mappings. These are all the configuration files for data access. The hibernate mapping files are found under the conf/machines/web1/rpallow/classes directory. These are set up on a per DAO basis. For example, the UserDAO has a user.hbm.xml mapping file.
View configuration is found in the classes directory in the file, views.properties. This file is a mapping file to all JSPs. Error messages are under the same directory in the file error.properties.

5.4.2 Logging

GRADS has an excellent logging system set up. Logging is configured in the conf/machines/web1/rpallow/log4j.properties file. Consult the log4j documentation for adjustments to logging levels. Log files are created and placed in the logs file. The date file, i.e. 2005-5-12.log, is created by Tomcat and displays its log file. This is similar to the Tomcat catalina.out file. The file, grads.log, are log statements within code of the GRADS system.

5.4.3 Testing

Testing files are found under the directory src/tests. Configuration for the testing is found in the build.xml file and java files under the tests directory. Please note that testing uses its own log4j log file named log4j_testing.properties. This is found in the conf/machines/web1/rpallow/ directory.
CHAPTER SIX
CONCLUSION AND FUTURE DIRECTION

6.1 Results

The GRADS system has resulted in an easier to navigate user interface, brighter look and is more error free. This is a solid improvement over the existing system in all aspects.

The number of pages for the user has declined from the previous version of the system. At the same time more user functions have been added. The user is able to navigate to almost any interface of the system within two clicks of the mouse. This is achieved by a global menu bar with links to most of the system for the applicant and staff. The recommender interface has been reduced to two pages and takes the recommender straight to his or her reference form.

Through the use of CSS and consulting with artists, GRADS presents a bright and clean interface to the user. A color palette was selected and strategically displayed to accent certain parts of the interface. For example, the applicant application menu bar was considered a vital part of the system. Through strategic use of color, this was accented with the effect of being the most prominent item on each applicant page. For every page a decision was made as
to which interface item needed to be highlighted. This system has successfully achieved this, resulting in a clean and bright look.

The validation system for GRADS allows for precise error messages to be displayed to the user. Errors are placed at the top of the page and next to the field where the error occurred. This has also added to the simplicity of the system.

6.2 Benefits

The new implementation of GRADS in the MVC pattern has bought this system inline with the latest Java Web applications. The Spring framework allows for software components to be plugged into the system very easily. Now GRADS is using the latest software tools for configuration, validation, logging, PDF generation, testing, and overall development. Everything is more modularized.

Configuration and new components can be changed or added without significant change to code. If a Web designer wants to change the menu bar for all applicant pages, there is only one file to edit. If a database programmer needs to implement a new ORM mapping tool, only new DAO implementations need to be written. Controller and model objects do not need to be modified. These are just two
examples of the solid building block of the new GRADS system.

The Ant build tool, log4j logging and JUnit testing have added a huge benefit to the development process of GRADS. In the previous system, there was no testing, logging was only done through Tomcat, and JSP and controller changes required that Tomcat would have to be restarted. This is a time consuming process. Now, the Ant build tool allows for redeployment of the system, without restarting Tomcat. Also, JSP files can be edited and changes viewed instantly. All log files are placed inside the GRADS file system and are broken down into different files and debugging levels at the will of the developer.

6.3 Future Work

This new building block should allow for easier upgrades to the current system in the future. One upgrade that is absolutely necessary is the implementation of an interface for the graduate coordinator for the MS program in Computer Science. She should be notified of submitted applications and be able to evaluate them through GRADS.

A second necessary upgrade which would help the staff would be some type of automatic import of applicant data to the student computer science database. This system reduces
the amount of physical labor by generating an SQL script, but it still needs to downloaded by staff which takes time. A Web Service would be the best way to implement this feature.

Because of the flexibility of the MVC architecture and the Spring framework, GRADS can be a strong building block for future versions of the system and could be used for other department’s Master Degree applications. Many style attributes of GRADS can be changed in one program file. The whole system could be modified to this style in a future version. This would allow for GRADS to be used for other applications with minimal work. GRADS is already set up to use any database. Only new objects would have to been coded for a new application.

Due to time constraints, design and help menus where kept to a minimum. This should also be considered for future development.
REFERENCES


