Committee Tracker for Cornell College

A Manuscript

Submitted to

the Department of Computer Science

and the Faculty of the

University of Wisconsin-La Crosse

La Crosse, Wisconsin

by

Walter Seubert

in Partial Fulfillment of the

Requirements for the Degree of

Master of Software Engineering

May, 2010

Committee Tracker for Cornell College

By Walter Seubert

We recommend acceptance of this manuscript in partial fulfillment of this candidate's requirements for the degree of Master of Software Engineering in Computer Science. The candidate has completed the oral examination requirement of the capstone project for the degree.

_____	_____
Dr. Kasi Periyasamy Examination Committee Chairperson	Date
_____	_____
Dr. Tom Gendreau Examination Committee Member	Date
_____	_____
Dr. Mark Headington Examination Committee Member	Date

ABSTRACT

SEUBERT, WALTER, T., "Committee Tracker for Cornell College", Master of Software Engineering, May 2010, Kasi Periyasamy.

Committee membership is considered to be an important part of being on the faculty at Cornell College in Mount Vernon, Iowa. Approximately two thirds of the faculty members serve on a standing committee every year. The Committee on Committees is responsible for the lifecycle of all committees along with managing the election and nomination process for the standing committees. Managing the election and nomination process is a time consuming manual task from January through April every year. This manuscript describes the development of a web based application that enables the faculty at Cornell College to manage their committee system and to automate their voting system.

ACKNOWLEDGEMENTS

I would like to express my sincere thanks to my project advisor Dr. Kasi Periyasamy for his valuable guidance and gentle encouragement. I would like to thank the project sponsor Leon Tabak from Cornell College who initiated this project and provided the support for this project. I would like to thank Diep Mai for acting as a software tester. Finally, I wish to thank my wife and children for their patience and encouragement during the tenure of this project.

TABLE OF CONTENTS

ABSTRACT iii
ACKNOWLEDGEMENTS iv
TABLE OF CONTENTS v
LIST OF FIGURES vi
GLOSSARY vii
1 Introduction 1
2 Life Cycle Model 3
2.1 Program Constraints 3
2.2 Program Risks 3
2.3 Project Phases 4
2.4 Resulting Life Cycle Process Model 4
3 Requirements 6
3.1 Requirements Capture Tool 6
3.2 Requirements Capture Tool Output Example 8
4 Application Architecture 10
5 User Interface 11
5.1 Explicit Navigation 11
5.2 Consistent Look and Feel 11
5.3 Data Validation 14
6 Data Access 16
6.1 Database Access 16
6.2 Alternative Database 17
7 Committee System Management 20
7.1 Initial Design 20
7.2 Committee Model 21
7.3 Committee and Position Constraints 22
7.4 Automatic Population of Committees 23
7.5 Nominations and Elections 23
8 Verification 30
9 Tool Selection 32
10 Limitations 33
11 Continuing Work 34
11.1 Reevaluate the use of MySql 34
11.2 Reevaluate the use of Apache Tomcat 34
11.3 Performance Tuning 34
11.4 Security 35
12 Bibliography 36
Appendix 37

LIST OF FIGURES

Figure 1: System Requirements Example 8
Figure 2: Software to System Requirements Mapping Example 8
Figure 3: Software Requirements Example 9
Figure 4: System to Software Requirements Mapping Example 9
Figure 5: Visual JSF Navigation Flow 11
Figure 6: User Interface Highlighting the Navigation Bar and Header 12
Figure 7: Adding and Editing Users Interface 13
Figure 8: Committee ER Diagram 16
Figure 9: Database Connection Design 18
Figure 10: Committee Constraints Table 22
Figure 11: Committee Manager Use Cases 25
Figure 12: Voting Member Use Cases 26
Figure 13: Design for Election Management Buttons 27
Figure 14: Design for Election Management Member States 27
Figure 15: Election GUI for Committee on Committees Members 28
Figure 16: Election GUI for Users 29
Figure 17: Committee Model 37
Figure 18: Committee Position Constraint Container 38
Figure 19: Committee Constraint Container 38
Figure 20: Entire Committee Model with Constraints 39
Figure 21: Code Sample of a JUnit Test 40

GLOSSARY

Apache Ant (ANT)

Apache Ant is a Java-based build tool with many integrated tasks such as JUnit, Jar, and Javac.

Apache Tomcat

Apache Tomcat is an open source application server that implements the Java Server Pages and Java Server Faces Specifications.

Business Intelligence and Reporting Tools (BIRT)

BIRT is an open source Eclipse-based reporting system that integrates with Java/J2EE application to produce compelling reports.

Cascading Style Sheets (CSS)

CSS is a style sheet language used to separate HTML document content from presentation including layout and color.

Composite Pattern

The Composite Pattern is a design pattern that allows a developer to compose objects into tree structures to represent part-whole hierarchies. Composite Pattern lets clients treat individual objects and compositions of objects uniformly.

Connection Pool

A connection pool is a cache of connections that can be reused. Connection pools are commonly used to enhance the performance of executing commands on databases.

Decorator Pattern

The Decorator Pattern is a design pattern that attaches additional responsibilities to an object dynamically. Decorators provide a flexible alternative to subclassing for extending functionalities.

Eclipse

Eclipse is an open source software development environment consisting of an IDE and a plug-in system that allows it to be extended.

Factory Method Pattern

The Factory Method Pattern is a design pattern that defines an interface for creating an object while letting the subclasses decide which class to instantiate.

Firefox

Firefox is a free and open source web browser descended from the Mozilla Application Suite and managed by Mozilla Corporation.

Google Chrome Browser

Google Chrome is a web browser developed by Google .

Java Server Faces (JSF)

JSF is a MVC framework for building user interfaces for web applications that typically utilize JSP.

Java Server Pages (JSP)

JSP is a server side Java technology that can be used to dynamically generate web pages.

JUnit

JUnit is a unit testing framework for Java. Software developers can create tests by extending the class TestCase which is provided by the JUnit framework.

Microsoft Access

Microsoft Access is a relational database distributed by Microsoft.

Microsoft Internet Explorer

Microsoft Internet Explorer is a web browser developed by Microsoft and typically distributed with the Microsoft Windows operating systems.

Model View Controller (MVC)

MVC is an architectural design pattern that divides an interactive application into three components. The Model contains the domain logic and data. Views display information to the user. Controllers handle the user input. Views and Controllers together comprise the user interface.

MyEclipseIde

MyEclipseIde is a low cost subscription based Eclipse IDE that provides an integrated Java EE environment that integrates with databases and application servers. It is a collection of open source and proprietary components and tools.

MySql

MySql is an open source relational database.

Observer Pattern

The Observer Pattern is also known as and can be thought of as having the roles Publisher and Subscriber. It defines a one-to-many relationship between a set of

objects. There is one Publisher and many objects can subscribe to its contents. When the data being published changes, the subscribers are notified that a change occurred.

PostgreSQL

PostgreSQL is an open source object-relational database management system.

Realm Managed Security

A Realm is a "database" of usernames and passwords that identify valid users of a web application, plus an enumeration of the list of roles associated with each valid user. You can think of roles as similar to groups in Unix-like operating systems, because access to specific web application resources is granted to all users possessing a particular role. A particular user can have any number of roles associated with their username.

Singleton

The singleton pattern is a design pattern that is used to restrict instantiation of a class to one object. This is useful when exactly one object is needed to coordinate actions across the system. The typical implementation of a Singleton consists of a private constructor, a private static reference to the Singleton class initialized to null, and a static access method that returns the reference.

The constructor is private to ensure that the constructor cannot be called by other classes to prevent unintended instantiations of the object from being created. The private static reference is usually initialized to null. The static access method creates an instance of the object if the reference to the Singleton class is null and then returns the value of the reference.

Subversion (SVN)

SVN is an open source version control system that utilizes a command line user interface. The Subversion solution includes a data repository and a command line client.

TortoiseSVN

TortoiseSVN is a SVN client implemented as a Microsoft Windows shell extension.

Unit Testing

Unit Testing is a software verification and validation method in which a programmer can test units of code. The units can be as small as an individual function or as large as an entire subsystem.

Validation

Validation is a process that confirms that the product (or partial product) meets the users' expectations.

Verification

Verification is a process that confirms a development process or activity or task to be correct.

1 Introduction

Committee membership is considered a very important part of being on the faculty at Cornell College. Per Leon Tabak, the project sponsor, "Service on committees is important to individual members of the faculty and to the faculty as a whole. When the college considers an application for tenure or promotion, it evaluates the quality of the candidate's teaching, service, scholarship, and commitment to the liberal arts. Faculty self-governance is an organizing principle of the college. Service on committees provides one of the most important opportunities for members of the faculty to work with colleagues outside of their own academic departments. We find satisfaction in the social engagement with our peers and in the work of shaping the college's programs."

Approximately two-thirds of the 80 faculty members serve on a standing committee every year. Some of the committees are intended to be short term advisory committees and are handled by appointment and nomination only. There are also 13 elected posts for standing committees in the committee system.

The Committee on Committees is responsible for the lifecycle of all committees along with managing the election process for the standing committees. Managing the election process is a time consuming manual task from January through April every year.

Managing the election process involves three steps:

1. Determine the eligibility requirements for each position on each specific committee.
2. Determine who meets the eligibility requirements for the current committee being filled.
3. Fill up each position on the committees.

Currently, all three steps of the election process are done manually by a member of the Committee on Committees. This involves the following manual activities:

- tracking of the eligibility status of every faculty member for every available position,
- tracking the committee participation of every faculty member,
- creation of election announcements for nominations and selection rounds,
- tallying of votes, and
- creation of an updated roster showing the current makeup of every committee.

The primary objective of this project is to develop a web based application that automates the three steps of the election process.

This will require developing a user interface that enables:

- Configuration of rank, length of service, division, department, and teaching responsibility for all faculty members.
- Creating and configuring the requirements and constraints for each position on each committee.
- Initiation and management of the committee voting process.
- Entry of legacy committee appointments to provide an accurate picture of a faculty member's service to Cornell College.

The server that will be hosting the application is only available on Cornell College's intranet and so security is not a high priority. However, there is a desire to have a growth path to having the application hosted on a server that is on the internet.

2 Life Cycle Model

2.1 Program Constraints

The Cornell College Faculty Handbook clearly defines how the committee system works as well as the rules for the nomination and election process. Since the faculty handbook defined the domain logic, they only real point of variability was the user interface. Through customer interviews, it was learned that the customer did not have any particular desire to have specific input on the user interface with the exception that the application would be web based utilizing JSF.

The customer's primary motivation for a web based application was simple deployment of the application. The customer's secondary motivation was from the computer science department; they are seeking practical examples and problems for their students. This application will serve as a practical example in many areas including verification, user interface design, database design, and developing domain logic as the Cornell College students deploy, maintain, and grow this application.

2.2 Program Risks

Upon completion of customer interviews, it was realized that a significant risk to the project was with misunderstanding the committee system as it was defined in the Cornell College Faculty Handbook because there are many overlapping requirements that can at times appear to be contradictory. To mitigate this risk, it was decided to create a technical baseline from which the application could be developed with confidence.

The customer constraint that required a web based application utilizing JSF presented a different challenge due to the developers minimal experience with web based applications and complete lack of exposure to JSF.

2.3 Project Phases

While investigating the project and performing the high level design, the opportunity to decompose the work into the following three separate phases became clear:

1. User entry including user permissions and roles.
2. Committee entry including committee and position constraints.
3. The election process and nomination process.

2.4 Resulting Life Cycle Process Model

To mitigate the program risks, a hybrid of the waterfall and evolutionary prototyping processes were utilized. The waterfall process was used to capture the requirements and the evolutionary prototyping process was used for the design, code, and testing aspects.

The waterfall process relies on very distinct development phases with the previous step's completion as a prerequisite. The only part of the waterfall process used was to create a complete technical baseline by completing all of the requirements at once. The Cornell College Faculty Handbook was mapped into a system level requirements document. This allowed the developer to create a software requirements document with complete tracing to the system level requirements. This is discussed in more detail in section 3, "Requirements".

The evolutionary prototyping model involves creating a prototype and then continually evolving it until the final design is reached. The choice of this process allowed the developer to learn everything needed to develop the bare bones proof of concept for each phase and then enhance it until it was complete. Additionally, the use of the evolutionary prototyping model shortened the feedback cycle of the design choices which accelerated the learning process.

Using the waterfall process on its own would not have been suitable for this project because it would have loaded all the risk for the project to the later stages

due to the developer not having any exposure to JSF and minimal experience with web based applications. The decision to use the evolutionary prototyping model supported the need of the developer to learn the technology and tools required for this project.

3 Requirements

Customer interviews were utilized to get introduced to the Cornell College committee system and gain an understanding of the intended scope of this project. During these customer interviews the developer listened, took notes, and asked many questions. The developer was provided a copy of the Cornell College Faculty Handbook which allowed the developer to gain an in-depth understanding of the committee system, the structure and constraints for committees and positions, and the faculty nomination and election process.

The Cornell College Faculty Handbook served as the primary source of system level requirements for this application. These requirements were enhanced and expanded by the information from customer interviews. These high level system requirements clearly defined what the application must do but were often too abstract to be directly implemented and verified. Additionally, there are many identical or nearly identical system requirements particularly in the area of committee structure and constraints.

To address the problem that the system requirements didn't readily enable development, a decision was made to create two tiers of requirements. The system level requirements would capture high level functionalities the application must do and the software level requirements would decompose the system level requirements to provide a straightforward means of implementation and verification. To ensure that the software requirements adequately decomposed all of the system requirements, it was determined there was a need to establish concrete traceable links between the system and software requirements.

3.1 Requirements Capture Tool

The developer was unable to find a low cost or free tool to assist with requirements capture and linking, which drove a decision to create a tool that would assist him in this project. This tool is a simple Microsoft Access database

to capture and store the requirements, along with BIRT to produce reports that served as requirements documents.

The database consisted of two simple tables, one for capturing the requirements and the other for linking them. The requirements table consisted of an auto incrementing integer that served as the primary key, a text field for the requirement, two integers to allow setting the level to enable requirement nesting and order, and four booleans for identifying the entry as a system requirement, software requirement, header, or note. The link table consisted of an auto incrementing integer that served as the primary key, and two other integers that were used to capture the primary key of both a software and system requirement. Each entry in the link table represented a single link between system and software requirements. Multiple entries can be made to provide one-to-many and many-to-one relationships.

The use of a database to capture the requirements and also link them enabled the use of queries to combine the information in useful ways. The developer created queries to identify both the system and software requirements that were not mapped. Unmapped system requirements represented misses in the software requirements. Unmapped software requirements represented either necessary derived requirements or design and implementation choices that were not customer requirements.

BIRT easily integrated with Microsoft Access. Four reports were created: system requirements, software requirements, software mapped to system requirements, and system mapped to software requirements. The two reports that mapped the requirements together were especially useful for requirements validation and ensuring that the system requirements were adequately decomposed into software requirements.

3.2 Requirements Capture Tool Output Example

Figure 1 is an example of the system requirements for the Academic Programs committee. The ID column is the unique ID of that requirement. The level is used to view and provide a greater understanding of the decomposition of requirements. Requirement #49 is a good example of a system requirement that cannot be directly implemented but describes the high level behavior the application must support.

ID	Level	Requirement
		one-year, renewable terms. Each must have one or more years of service as a full-time member of the teaching faculty.
47	6	The Registrar shall serve the subcommittee as a member ex officio.
	5	***Academic Programs***
49	6	The chair shall be nominated by the Committee on Committees to a two-year, non-renewable term. The chair must have three or more years of service as a full-time member of the teaching faculty.
50	6	The four additional members shall be nominated by the Committee on Committees to one-year, renewable terms. Each member must have one or more years of service as a full-time member of the teaching faculty.
51	6	The Dean of the College, though not a member of the subcommittee, will frequently be invited to join Academic Programs in its regular meetings so the Dean may have ample opportunity to introduce and argue for new programs, to mediate when appropriate, and to provide relevant information. Also, the Dean may contact the subcommittee to request to meet with the subcommittee when he/she feels that discussion of issues on the subcommittee's agenda would be facilitated by the Dean's presence. The Dean will receive the minutes of the subcommittee meetings.
117	6	No more than one person per department may serve on the Academic Programs subcommittee.

Figure 1: System Requirements Example

Requirement #49 is a system requirement than can be decomposed into many lower level software requirements. Figure 2 shows the many software requirements that implement it including requirement #173.

49		**The chair shall be nominated by the Committee on Committees to a two-year, non-renewable term. The chair must have three or more years of service as a full-time member of the teaching faculty.**
	173	Minimum years of service as a full-time member of the teaching faculty shall be a means to constrain a committee position.
	206	There shall be the ability to establish the length of service of the position.
	205	A committee role can be can be filled via Nomination.
	213	There shall be the ability to establish if the position is non-renewable.
	166	'Chair' shall be a specific committee role.
	234	Full Time Faculty Member shall by a user type in the system.

Figure 2: Software to System Requirements Mapping Example

As seen in Figure 3, requirement #173 is one of many software requirements that provide a means to apply constraints to a committee position.

171	3	*There shall be a means to create constraints on each committee position.*
173	4	Minimum years of service as a full-time member of the teaching faculty shall be a means to constrain a committee position.
210	4	Minimum years of service as a full-time faculty member shall be a means to constrain a committee position.
211	4	Being an administration member shall be a means to constrain a committee position.
216	4	Being a student shall be a means to constrain a committee position.
175	4	Maximum number of people from an academic department shall be a means to constrain a committee position .
176	4	Maximum number of people from a division shall be a means to constrain a committee position.

Figure 3: Software Requirements Example

Figure 4 shows how Software Requirement #173 maps to several different system level requirements.

173	**Minimum years of service as a full-time member of the teaching faculty shall be a means to constrain a committee position.**
45	The chair shall be nominated by the Committee on Committees to a two-year, non-renewable term. The chair must have three or more years of service as a full-time member of the teaching faculty.
46	The two additional committee members shall be nominated by the Committee on Committees to serve one-year, renewable terms. Each must have one or more years of service as a full-time member of the teaching faculty.
50	The four additional members shall be nominated by the Committee on Committees to one-year, renewable terms. Each member must have one or more years of service as a full-time member of the teaching faculty.
49	The chair shall be nominated by the Committee on Committees to a two-year, non-renewable term. The chair must have three or more years of service as a full-time member of the teaching faculty.

Figure 4: System to Software Requirements Mapping Example

4 Application Architecture

The Cornell College Faculty Handbook provided all of the required information on faculty voting, faculty nominations, the committee system, and the structure and constraints of permanent committees and positions. In addition to the permanent committees, there are many ad hoc committees to address special topics. The structure and constraints of these future committees won't be known until the committees are created. This need to support flexible committee creation, structures, and constraints on both the committee and positions presented the major design challenge for the project.

The customer constraint requiring the use of JSF drove the application architecture which required the selection of an application server and a database.

Apache Tomcat was chosen as the JSF compliant application server. This decision was rather simple at the time due to being relatively unchallenged as an open source solution when this project began in 2007.

When selecting a database, the developer's research led to two choices, PostgreSql and MySql neither of which stood out as a clearly dominant choice. Ultimately the developer chose MySql because it met all of the requirements and provided a friendlier experience for a Windows environment.

5 User Interface

The use of JSF influenced the workflow because JSF does an excellent job of enforcing a rigid use of the MVC design pattern. The JSF pages did not lend themselves to adding code to them. This rigid enforcement drove all of the application logic to be in the application server behind the user interface, hence enforcing the separation of the application layer from the presentation layer.

5.1 Explicit Navigation

The navigation rules for JSF pages are controlled and configured by the faces-config.xml file. Figure 5 is a visual representation of page navigation flow between several JSP pages that are used to add and edit user data. The AddNewUser navigation case causes the AddUser.jsp page to be displayed. If the user attempts to add a new user to the system and the data validation is performed successfully, the AddUserSuccess navigation case is generated which causes the ViewUsers.jsp page to be displayed.

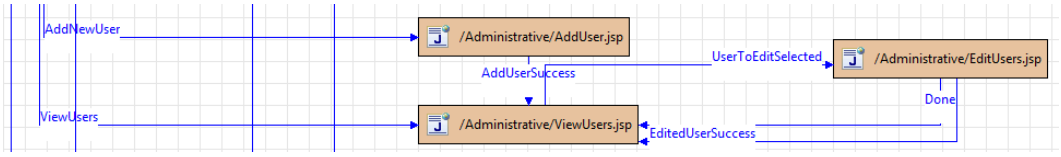

Figure 5: Visual JSF Navigation Flow

The explicit navigation of JSF does an excellent job of consolidating all of the navigation rules to a single location. This solves the problem of having spaghetti navigation rules that can plague HTML, JSP, and other web based applications.

5.2 Consistent Look and Feel

One of the common challenges for any application is that the user interface should have a consistent look and feel. This is particularly more challenging for web based applications because each page renders itself. To address these challenges the developer utilized Cascading Style Sheets (CSS) and also the ability to create and include partial JSP pages to create a single page that is displayed.

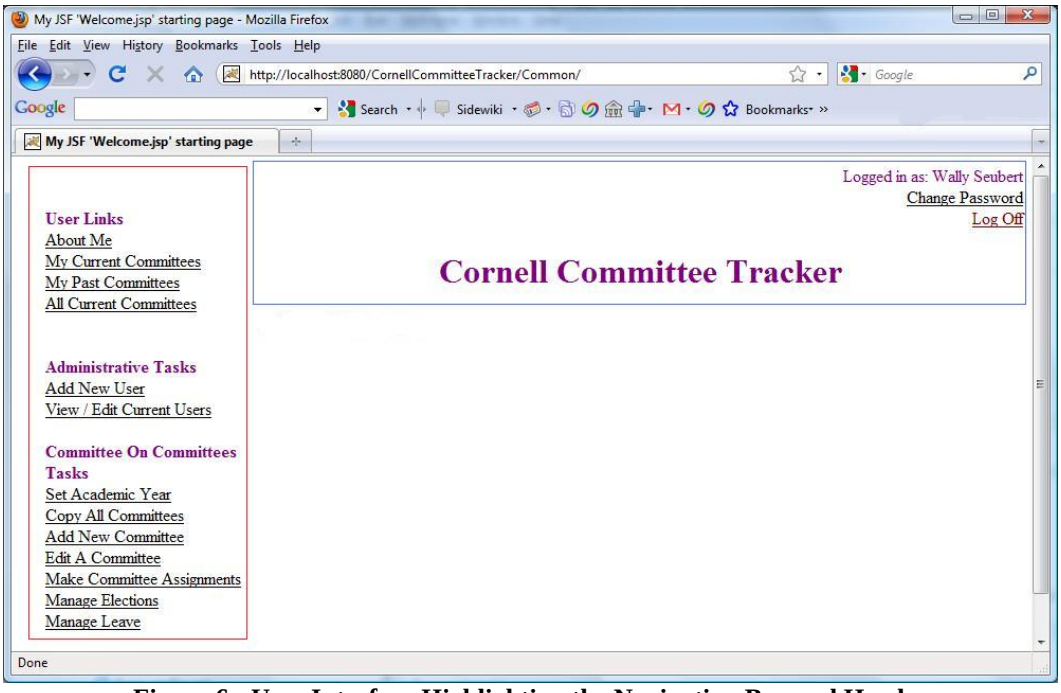

Figure 6: User Interface Highlighting the Navigation Bar and Header

Figure 6 shows an example of the applications user interface. The page consists of three major elements: the header identified by the blue box, the navigation bar identified by the red box, and usable part of the page represented by the white space below the header and to the right of the navigation bar. Both the header and navigation bar are partial JSP pages that are included by every page to enforce a common look and feel for every page in the application.

The primary purpose of the header is to display the current user who has logged in. This was done to minimize the chance that someone would accidentally use the application when logged in as a different user.

The navigation bar is the primary navigational feature of the application. The navigation bar in Figure 6 shows three main navigation groupings: user links, administrative tasks, and committee on committee tasks.

The user links are available to every user that may participate in an election. They provide the ability to view the current committees, view all the data about

the user, and if eligible, vote in an election. The user interface with just the user links can be seen in Figure 16.

The administrative tasks and the tasks performed by the committee on committees are only available to the user if they are permitted to perform those tasks. Access to them is controlled by using realm managed security. The application server will not allow the users access to the pages if they are not configured to have the role required to view those pages. Additionally, to avoid confusion and promote a secure user interface, the navigation groupings are designed to only be visible if the logged in user is given access rights to those task groups. Figure 7 shows the page that is used to add or edit information about users. There are four checkboxes that allow the users roles to be configured. These selections directly control the users access to the tasks associated with them.

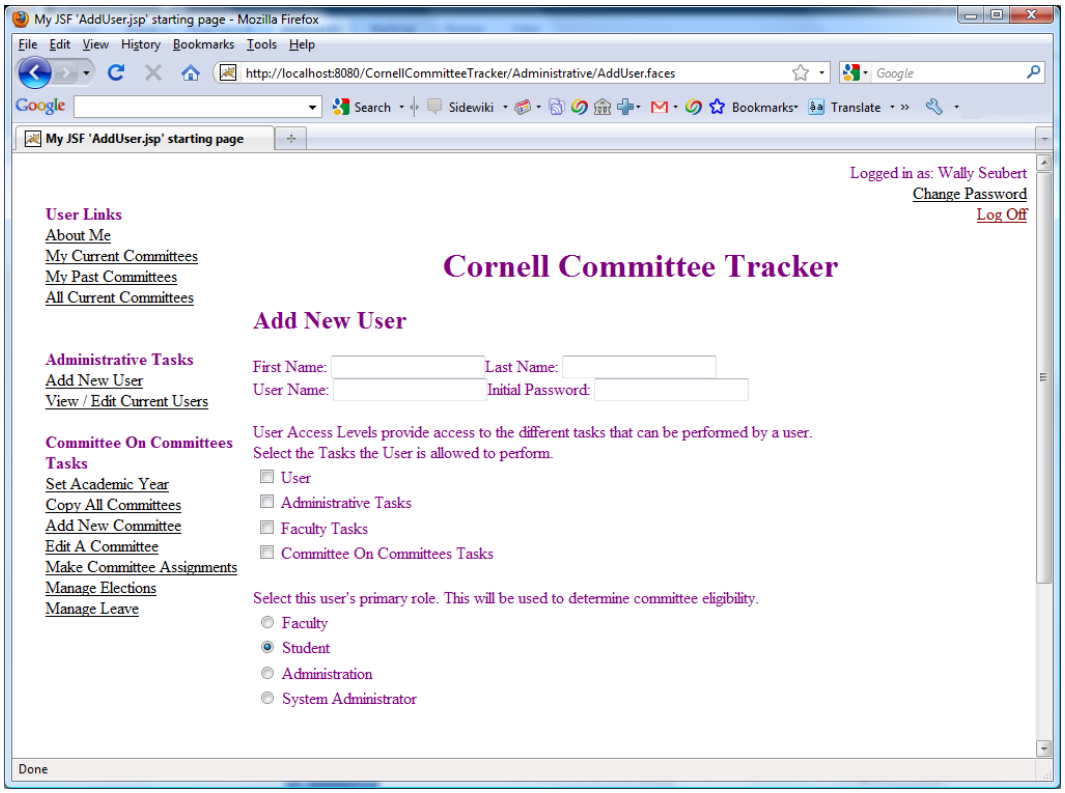

Figure 7: Adding and Editing Users Interface

5.3 Data Validation

The JSF framework provides both standard and custom validation as well as standard and custom error messages. There are three different types of data input validation performed in this application that leverage the JSF framework.

There is a need for at least one of the User Access Level checkboxes in Figure 7 to be selected. To accomplish this, the 'required' attribute was set to true for the selectManyCheckbox style input. The JSF framework takes care of the rest by displaying the error message if nothing has been selected. In this example, the JSF framework will be able to determine that the input requirements were not met because there wasn't any necessary domain knowledge required and because the data required to perform the validation came from a single field.

Another data validation requirement for the add user interface in Figure 7 is to ensure that user's username doesn't already exist in the system. The JSF framework is not capable of performing this validation check automatically because it requires domain knowledge. More specifically it requires the ability to ensure that the username does not already exist in the database. To perform this validation, a custom validation was created and registered with the user name input field. If the validation fails, an exception is thrown which includes the error message to be displayed. This type of validation was able to be performed because the entry field requiring validation was not coupled to any other entry field.

Yet another type of custom validation was required to be performed for this entry form because the JSF framework is only capable of performing validation on a single field. The final layer of validation that was performed is also custom and highly navigational. If the user is successfully added to the database, the navigation case AddUserSuccess is generated as can be seen in Figure 5. If the user is not able to be added, the navigation case does not occur, leaving the user

on the AddNewUser page and able to respond to the error message without losing the data that had been entered.

6 Data Access

This application has a backend database for storing the user and application data. The database schema is relatively simple and consists of 20 tables. Figure 8 shows the subset of the database design that is most complex. This portion is the subset that stores all information for the committee system in an academic year.

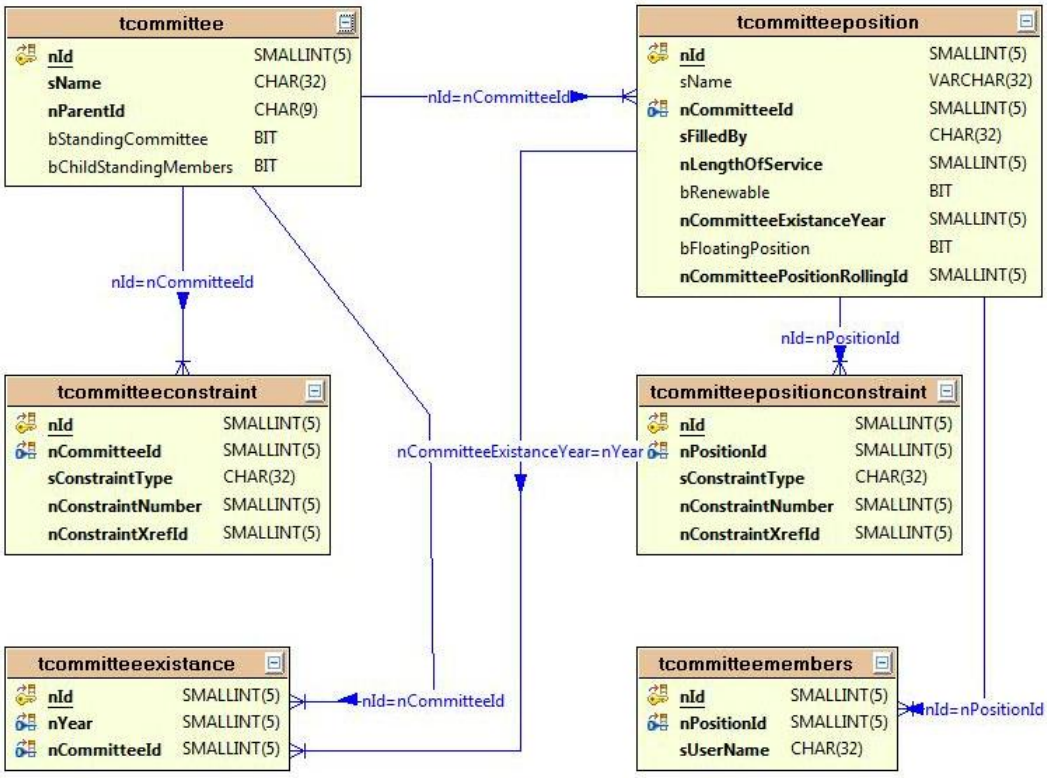

Figure 8: Committee ER Diagram

Although the database design was intentionally simple, the two substantial design decisions revolved around the optimization of computational resources for database access and a mechanism to support the use of an alternative database for both testing and development.

6.1 Database Access

This application requires frequent but sporadic access to the database by potentially many users. Creating a database connection is computationally

expensive. Maintaining and holding a database connection with ‘open’ status requires a significant amount of memory. Balancing these tradeoffs presented an interesting design choice.

Creating a new connection per database request would not be appropriate. This application requires frequent but sporadic access to the database and many pages require multiple database requests.

Creating a new database connection per user also would not be appropriate because it would require the server to have enough memory to maintain a connection for each user that is logged in. Due to the sporadic nature in which this application accesses the database, this approach cannot be justified because these connections would be idle most of the time.

The two previous solutions had significant drawbacks. The ideal solution to this problem is known as a connection pool. A connection pool creates a collection of resources that can essentially be borrowed and returned. The connection pool used by this application is used to hold and maintain connections to the database up to a configurable maximum number of connections. This solution is optimal for this application because it does not require the application to maintain the memory for a database connection for each user while providing fast access to the database because each connection does not have to be created separately.

6.2 Alternative Database

Designing for testability is always an important consideration for an application. Given that this application will be maintained by undergraduate students that will have varied degrees of experience and skill, designing for testability is a vital part of the design.

Given the decision to use a connection pool for database access, applying the singleton pattern to the connection pool would be an ideal way to ensure that there is one and only one connection pool in the system. However, this would make it

difficult to test because it would require a second means to access the test version of the database. Requiring special code to provide an alternate data access mechanism in every place the connection pool is accessed would not have been acceptable because it would make both development and maintenance difficult.

The resulting solution is a combination of the singleton and factory method patterns as shown in Figure 9. The connection pool itself is implemented in the base class ConnctionPool. It handles all aspects of implementing the connection pool. The class DataBaseConnection implements the singleton, but in a unique way. Instead of a single static access function to return the reference to the object, there are two static access functions that both return the single reference to the object that is created first. If the first access call is to getInstance() the connection pool will be created with the connection to the production database. If the first access call is to getTestInstance() the connection pool will be created with the connection to the test database. Any subsequent call to gain access by either method returns the connection created by the first call.

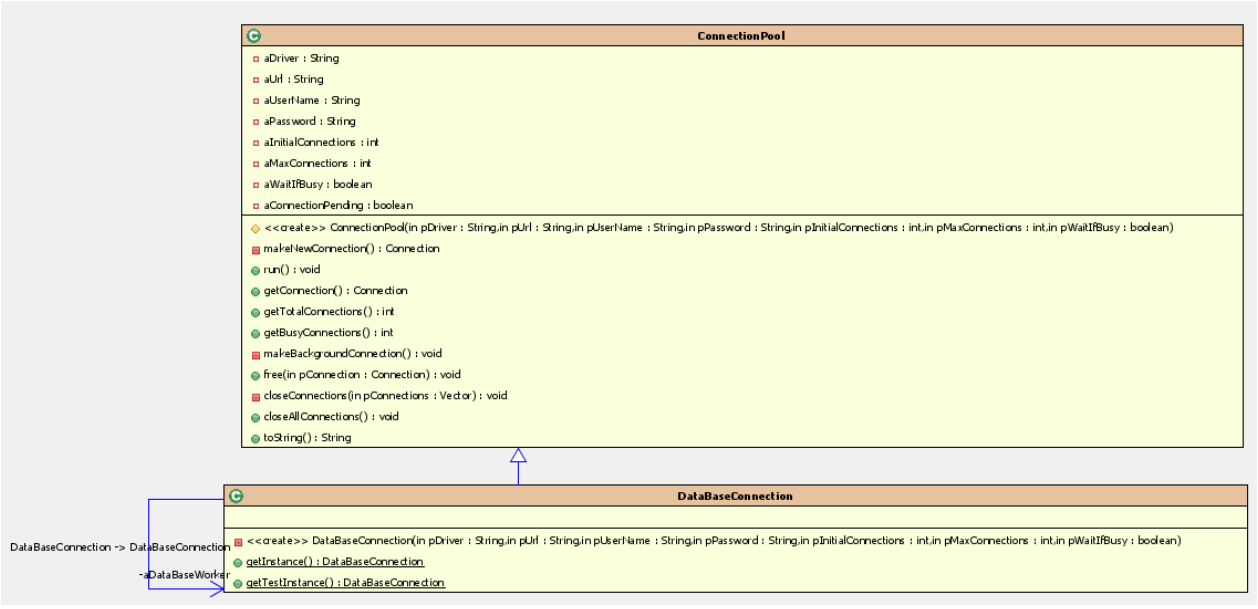

Figure 9: Database Connection Design

This dual interface to the singleton allows developers to create a test instantiation of the application. If getTestInstance() is called, the part of the application being tested will utilize the test database instead of the production database. This gives the developers the ability to unit test, prototype, and debug without fear of altering the production database. An example JUnit test demonstrating utilizing the factory method pattern to instantiate the connection pool with the test database can be seen in Figure 21.

7 Committee System Management

Every committee at Cornell College is part of a defined hierarchy. For example, the Academic Standing committee is one of the subcommittees of the Committee On Academic Affairs, and the Committee On Academic Affairs is one of the subcommittees of the Committees of the Faculty. The structure is important to maintain for display and organizational purposes.

The primary purpose of this application is to manage the committee system which is task oriented. These tasks include the ability to:

- define the committee structure including the committee hierarchy, adding committee positions to committees, and defining committee and committee position constraints,
- determine eligibility for each position on every committee without violating the defined constraints,
- assign members to committee positions in specific years,
- automatically populate the committees in the following years using time of service and service limits, and
- manage all of the nominations and elections.

7.1 Initial Design

As discussed in section 2 “Life Cycle Model”, phase 2 of the project was planned to create the committee entry system, including committee and position constraints. Since the evolutionary prototyping model was being used, the developer began by designing the section of the database needed for the committee system. Initially this design to accommodate the committee system was heavily based on queries accessing the database tables and design shown in Figure 8. This solution was well suited to define the committee structure and everything that entails. However, the remaining tasks such as determining the eligibility of every member for every specific position in a specific year started to

drive the creation of many new but slightly different queries. As the developer progressed down the path of adding and refining functionality, slight but necessary improvement to the database design kept being discovered. These changes had to be propagated to the growing list of similar queries to both read and write to the database. For short and long term maintainability reasons, it was determined that an entirely different approach was necessary.

7.2 Committee Model

The core concept to the new approach was to continue using the database for committee storage as shown in Figure 8, but create a model of the entire committee structure including positions, constraints, and assignments for the given year that can be loaded from and written to the database in its entirety. This approach to serialize the committee data allowed the developer to focus on creating an object oriented design optimized for the necessary tasks. This design can be seen with the superposition of the class diagrams in Figure 17 and Figure 20.

There is only one committee model in the application shared among all users. This is implemented via a singleton representing the committee model that has the collection of committees utilizing the Composite Pattern to maintain the hierarchy of committees and to enable iteration across all of the committees. The classes forming the composite pattern are CommitteeComponent, Committee, and CommitteeGroup and can be seen in Figure 17. These classes only contain the necessary functionalities to support the composite, with the exception that Committee uses aggregation to access the actual data for each committee.

When deployed, this application will simultaneously have multiple users. To ensure that classes dependent on the committee model are always up to date because they cache data, the committee model also implements the observer pattern and plays the role of Publisher.

7.3 Committee and Position Constraints

There are twelve requirements that specify the different means to constrain membership on a committee. The constraints are a mix of constraints on the committees themselves and the positions, as can be seen in Figure 10.

Constraint	Applied to Position	Applied to Committee
Min Years Full Time Faculty	X	
Min Years Faculty	X	
Administrative Member	X	
Student Member	X	
Max from 1 Dept		X
Min from 1 Dept		X
Max From 1 Div		X
Max from same Rank		X
Min Untenured		X
Min number in a Division		X
Mutually Exclusive		X
Tenure is Required	X	

Figure 10: Committee Constraints Table

The constraints added a significant challenge because every committee and every committee position can have a different combination of constraints, and each constraint requires different information to be evaluated. For example, to evaluate the constraint 'Max from 1 Dept', the department that each user is a member of as well as the department of every member already on the committee must be evaluated. In contrast the constraint 'Tenure is Required' only evaluates whether the member is tenured.

To avoid creating an all-encompassing class to evaluate all of the criteria for the eligibility of the committee and committee positions, the Decorator Pattern was utilized. As shown in Figure 20, both Committee and CommitteePosition have a container that utilizes the decorator pattern to contain its subset of the constraints. Each list of constraints is dynamically configured and the container is able to iterate through each constraint to determine the eligibility of each

individual user for both the committee and the specific committee positions. Each individual class implementing a constraint completely encapsulates everything required to evaluate that constraint.

7.4 Automatic Population of Committees

The initial step in the nomination and election process is to determine which positions are available to be filled for the following year. For some committees this may simply mean whether the member has served the complete duration of service, and for others it may mean that the member that has served for the longest duration is promoted to the position, such as the chair.

This effort required to perform this initial step accurately is one of the drivers for the creation of this application. Additionally, as discussed in section 7.1 "Initial Design" and section 7.2 "Committee Model", this step was a driver for the redesign of the Data Access portion of this application.

After the redesign, automatically creating the structure and populating the committee membership was relatively straightforward. The first step was to iterate over the entire committee structure and create a copy of each committee, including the positions for the desired year. The second step was to iterate over each committee and each position within the committee, using the calculated time they have served compared to the length of service to determine if they should be assigned the position or not. The verification of this effort is discussed in section 8 "Verification".

The relative ease with which this was able to be implemented validated the decision to redesign the Data Access portion of this application.

7.5 Nominations and Elections

The coordination of nominations and elections for a position begins with the determination of every eligible member for that position. Once eligibility is determined, the nomination round begins, which enables the members of Cornell

College who are eligible to participate in elections to begin the nominations. The nomination round continues until there are twice as many candidates that accept their nomination as there are positions to be filled. Elections are very similar to nominations except that there may be more than one round of an election if there aren't enough winners who have received the majority of the votes.

The use cases for the nomination and election activities are shown in Figure 11 and Figure 12. As can be seen in these diagrams, there is a complete separation of functionalities between the committee manager and voting member actors. This separation of functionalites drove the creation of two interfaces, one to allow Committee on Committee members to manage the elections and one to allow eligible faculty members to participate in the elections.

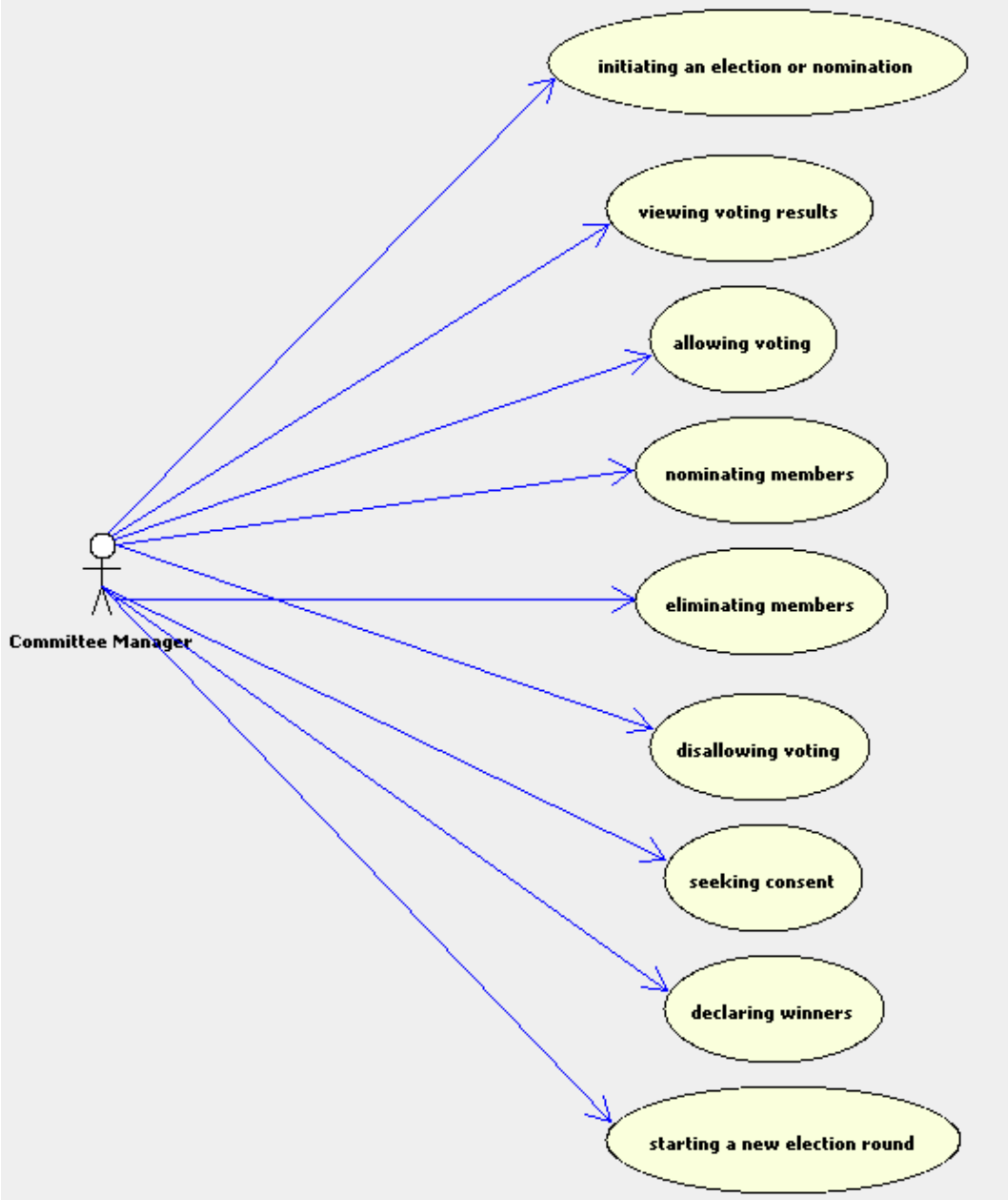

Figure 11: Committee Manager Use Cases

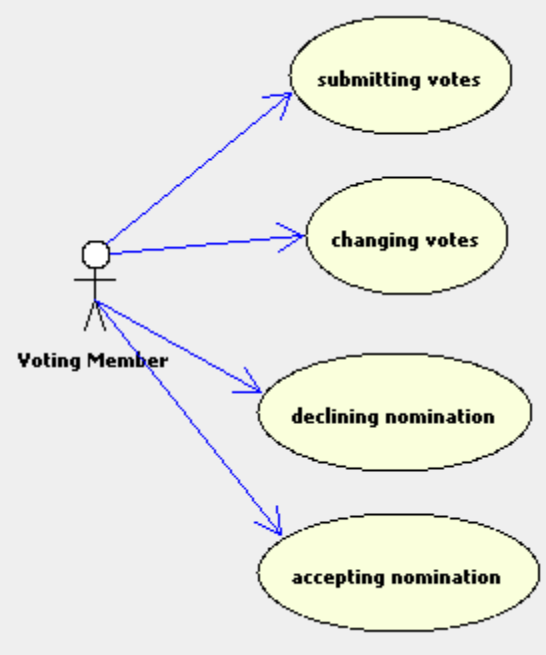

Figure 12: Voting Member Use Cases

To allow the Committee on Committees members to manage the nominations and elections, the user interface allows control over the state of the election as well as the state of the individual members that are eligible.

Figure 13 shows the design of the five action states for the election and when they are applicable. For example, when the nomination round is open, the only action available to the Committee on Committees member is to close the voting. As another example, after an election round is closed, if there aren't enough winners selected, the only choices to the Committee on Committees member are to allow more voting in the current round or to start a new round. This logic is implemented to control when the buttons are enabled and visible to Committee on Committees members to make the application simple and intuitive to use.

		Buttons for manageing a committee election				
		Allow Voting	Close Voting	Start New Round	Finalize Election	Seek Consent
Nomination Round	Open		x			
Nomination Round	Closed (Not enough Nominations Accecpted)	x				x
Nomination Round	Closed (1x Nominations Accecpted)	x				x
Nomination Round	Closed (1x Winners selected)	x			x	x
Nomination Round	Closed (> 2x Positions Accepted Nominations)	x		x		x
Election Round	Open		x			
Election Round	Closed (Not enough Winners selected)	x		x		
Election Round	Closed (1x Winners selected)	x			x	

Figure 13: Design for Election Management Buttons

Figure 14 shows the design of four of the five states an individual member can be in. Each member that is eligible for a nomination or election begins in the Eligible state. If the member declines nomination or is eliminated by the Committee on Committees, they are no longer eligible for the nomination or election and not available to be voted on by the users.

		States for an election			
		Winner	Nominate	Eligible	Eliminate
Nomination Round	Open				
Nomination Round	Closed (Not enough Nominations Accecpted)		x	x	x
Nomination Round	Closed (1x Nominations Accecpted)	x	x	x	x
Nomination Round	Closed (1x Winners selected)	x	x	x	x
Nomination Round	Closed (> 2x Positions Accepted Nominations)	x	x	x	x
Election Round	Open				
Election Round	Closed (Not enough Winners selected)	x	x	x	x
Election Round	Closed (1x Winners selected)	x	x	x	x

Figure 14: Design for Election Management Member States

Figure 15 is an example of the user interface for Committee on Committees members. In this example, the nomination round is closed, and there aren't any winners declared. Per the rules in Figure 13, the Allow Voting and Seek Consent buttons are available.

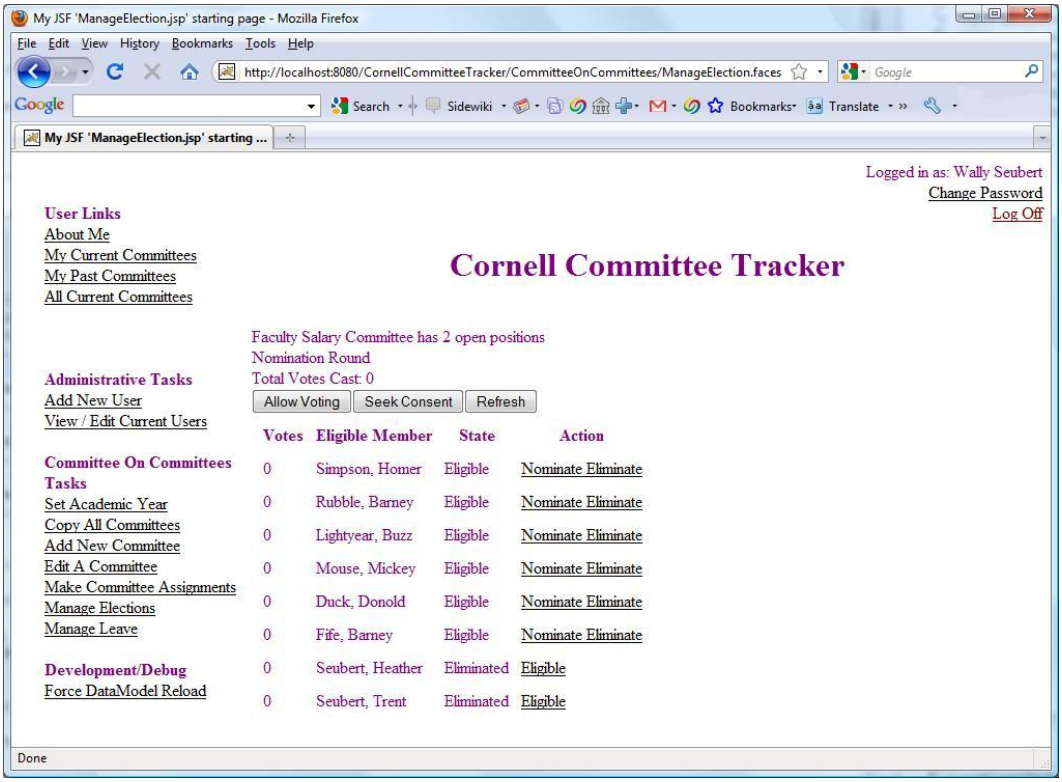

Figure 15: Election GUI for Committee on Committees Members

Figure 16 shows an example of the user interface as it would be seen by a user that is going to vote in a nomination. As can be seen by comparing the eligible user names in Figure 16 and Figure 15, the two members that have been eliminated in Figure 15 are not eligible to be voted for, as shown in Figure 16.

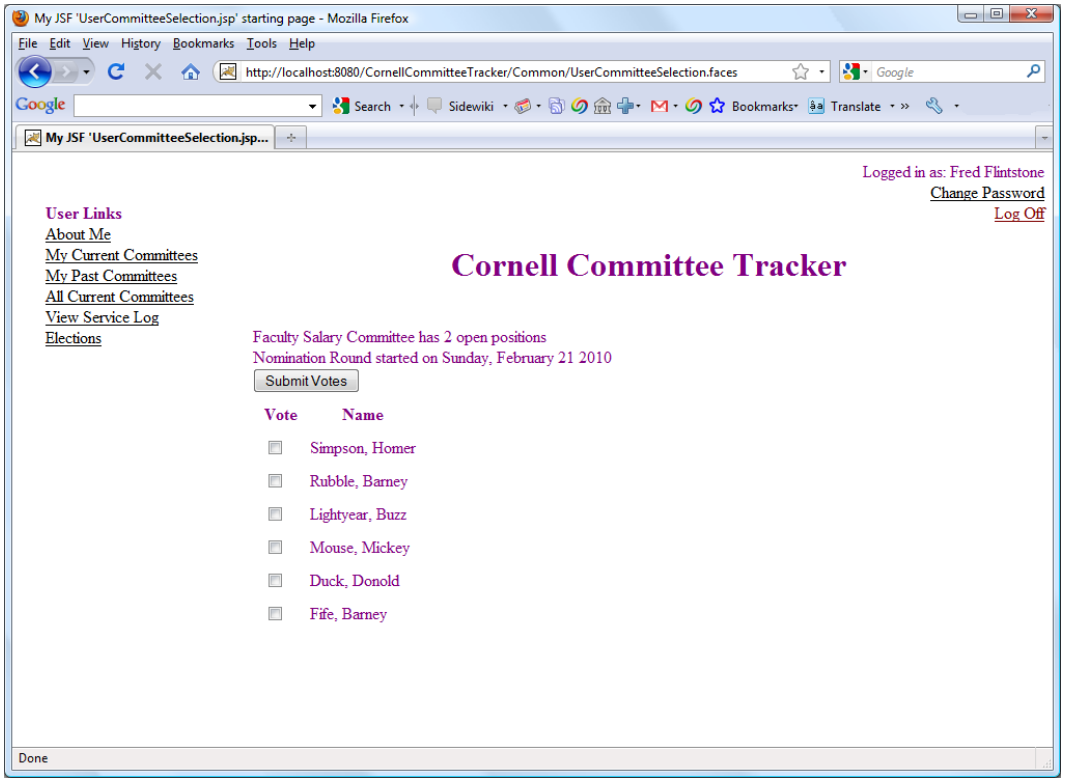

Figure 16: Election GUI for Users

The resulting user interface design for both the voter and the Committee on Committees members is very simple and intuitive to use. However, from a user interface design perspective, it was the most complex due to the need for rules to control which elements should be enabled and made visible to keep the user interface intuitive. This interface was the last one designed for the application and it demonstrates the culmination of everything that was learned throughout the application by using the evolutionary prototyping model.

8 Verification

In an attempt to automate as much testing as possible to minimize the amount of manual regression testing, JUnit was utilized for unit testing. The JUnit ANT task was configured to select classes based on pattern matching, and then to automatically execute all of those tests.

Some of the most beneficial tests created for this project were for the Committee Model. A sample of the JUnit test for the committee model can be seen in Figure 21. The aspects of the committee model that are tested in this manner are:

- Retrieving the history of a specific position for a specified depth.
- Copying the committees in the committee model and carrying forward the committee membership when the member has not exceeded the length of service.
- Iterating through the committee hierarchy that uses the composite pattern starting at different depths in the structure.
- The ability to find a specific committee.
- The ability to add and insert committees to the collection.
- The committee constraints for specific committees to determine eligibility.

In addition to automated testing via JUnit, the user interface and overall application were tested manually. To perform these tests on various browsers, Google Chrome, Firefox, and Internet Explorer were used. The primary purpose for using different browsers was to enable the simulation of multiple users simultaneously logged in and using the application. Initially it was believed that the developer could use multiple tabs in Google Chrome with each tab representing a different user because each tab has its own process. However, each

opened tab accesses the credentials from the first tab which therefore requires multiple browsers to simulate multiple users. The same behavior exists in Firefox and Internet Explorer as well.

9 Tool Selection

In addition to all of the previously mentioned tools, two other significant tool choices for this project were the IDE and the version control solution.

The version control solution chosen for this project is Subversion, which was used along with TortoiseSVN. The use of SVN allowed the developer to have a controlled and managed history of every artifact for this project at all times. As part of the evolutionary prototyping development model used for this project, not every attempt at improving the project was successful. There were times when user interface changes were not as usable as intended and other times when the application logic broke and was found via the JUnit tests. The ability to create a difference of the changes to understand broken logic, and when necessary, revert to the previous version of the file was extremely valuable. Over the duration of the project, there have been 288 commits to the repository that account for 1334 unique file changes.

The IDE chosen for this project is MyEclipseIDE. It is an Eclipse based tool that integrates many open source and also proprietary tools into a single installation. When this project began, the Eclipse J2EE project was in its infancy and MyEclipseIDE was the only tool in the cheap to free category.

The parts of MyEclipseIDE utilized the most were the Java debugger, JSP debugger, its database explorer perspective, the automatic deployment of the project to the server upon compilation, and the web based help forums. The decision to purchase a tool that provided a complete turnkey J2EE and JSF solution allowed the developer to focus on the problems at hand and quickly become productive.

10 Limitations

The biggest limitation with the current database design is that it utilizes the user's username as the primary key in the table tUserData. While this is an effective solution because there can never be more than one user with the same username, it is not conducive to changing the username because other tables link to it using the username. A solution to this problem would be to insert an integer based primary key to the table tUserData. This would require some updates in the database interface layer of the application, but the changes would not be prohibitively difficult or time consuming. Although the ability to change the username of a user isn't an explicit requirement, the limitation may make the application seem less professional to the end users if Cornell College utilizes each user's last name to construct their usernames and if the user undergoes a name change.

11 Continuing Work

11.1 Reevaluate the use of MySql

As stated in section 4 “Application Architecture”, at the time this project was started in 2007, MySql was the best choice due to limited free and open database choices. Ultimately what tipped the scales to MySql was that it had a native Windows installation and tools.

While MySql is still a valid database, the choice should be reevaluated because there have been many changes in the free and open database arena. MySql has been purchased by Oracle, PostgresSql now has a native Windows installation, and Apache Derby has been introduced in Sun’s J2EE Software Development Kit.

11.2 Reevaluate the use of Apache Tomcat

As stated in section 4 “Application Architecture”, at the time this project was started in 2007, Apache Tomcat was chosen because it was the de facto standard. Since that time, Sun has introduced Glassfish and has put a lot of effort into both developing and promoting it.

If Glassfish is determined to be a better application server, the only part of this application coupled to Apache Tomcat is the user of realm based security which is configured in Tomcat’s server.xml file. Glassfish supports realm based configuration, but the developer has not researched how to configure it.

11.3 Performance Tuning

Throughout the testing of the application, the developer was only able to simulate three simultaneous users, as discussed in section 8 “Verification”. While testing the application, the memory utilized by the Java process used by the server peaks around 260 MB and isn’t significantly impacted by multiple users. This was primarily driven by the use of a singleton for the CommitteeModel.

Additionally, the database connection pool is set up to allow for a maximum of 15 connections and the developer's testing allowed for three, due to the limitation of one user per browser type.

More work needs to be done to determine if adding many more users will push the memory usage up into an area that would be considered unusable or if the maximum number of connections for the Connection Pool needs to be increased.

11.4 Security

As stated in section 1, the "Introduction", this application is intended to be deployed on Cornell College's intranet, and security was not deemed to be important.

However, even if the application stays on the intranet forever, the users' passwords are currently stored as plaintext in the database. This would allow access to the passwords by anyone with access to the database. This on its own isn't a big security risk, but given the common practice of users reusing the same password for every online site, this could make a user's private password public.

12 Bibliography

[1] Len Bass, Paul Clements, Rick Kazman, *Software Architecture in Practice*, 2nd ed. , Addison-Wesley, 2003.

[2] Hans Bergsten, *JavaServer Faces*, O'Reilly, 2004.

[3] Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, Michael Stal, *Pattern-Oriented Software Architecture Volume 1: A System of Patterns*, Wiley, 2000.

[4] Eric Freeman, Elisabeth Freeman, Kathy Sierra, Bert Bates, *Head First Design Patterns*, O'Reily, 2004.

[5] Robert Futrell, Donald Shafer, Linda Isabell Shafer, *Quality Software Project Management*, Prentice Hall, 2002.

[6] Erich Gamma, Ralph Johnson, Richard Helm, John Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley, 1994.

[7] David Geary, Cay Hortmann, *Core JavaServer Faces*, 2nd ed. , Prentice Hall, 2007.

Appendix

The following figures are referenced in body of the thesis.

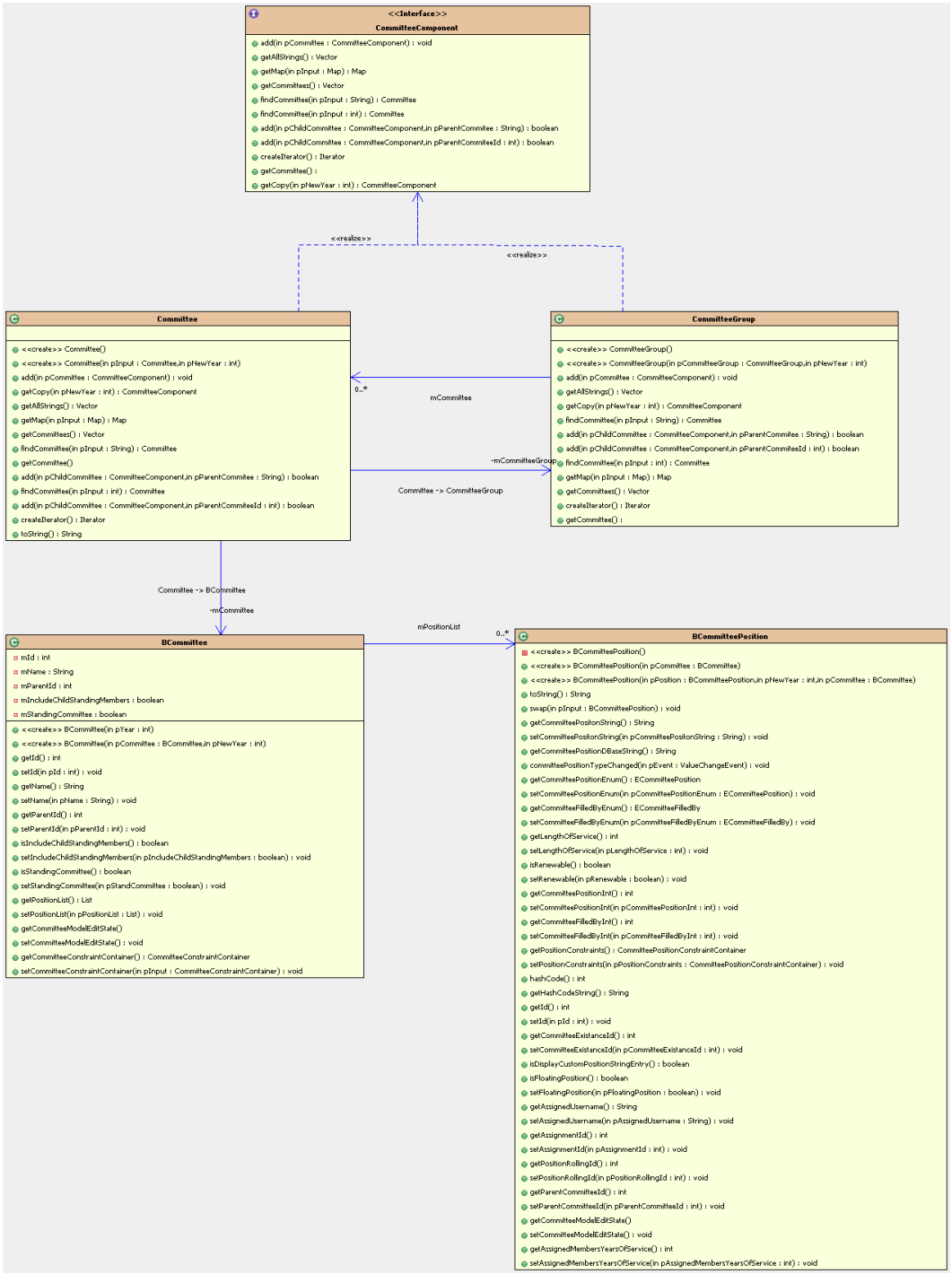

Figure 17: Committee Model

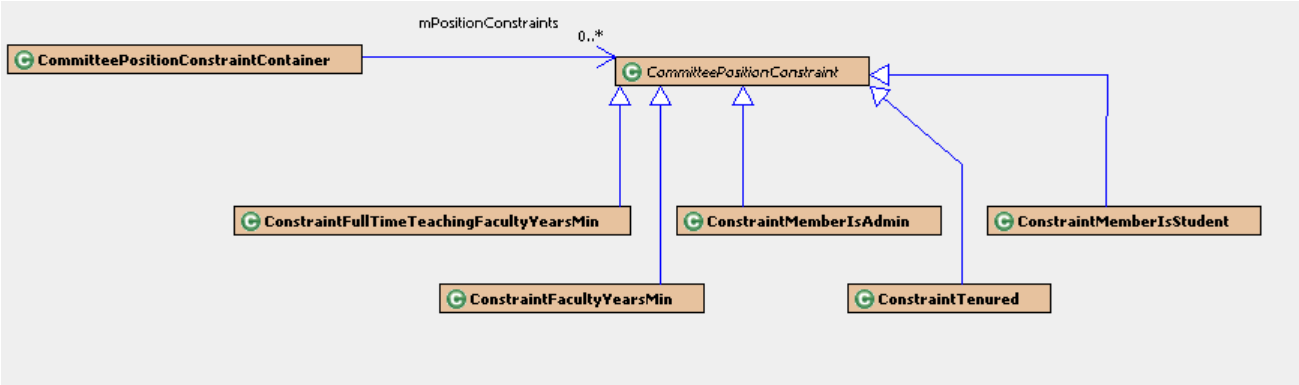

Figure 18: Committee Position Constraint Container

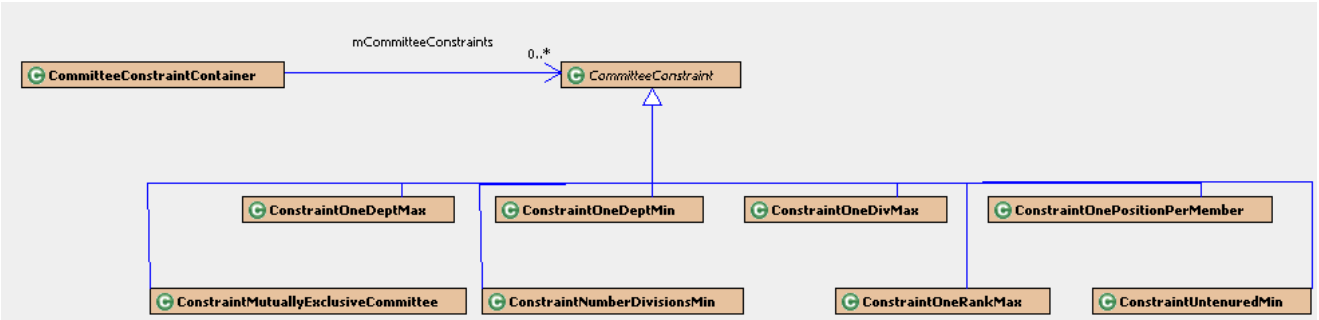

Figure 19: Committee Constraint Container

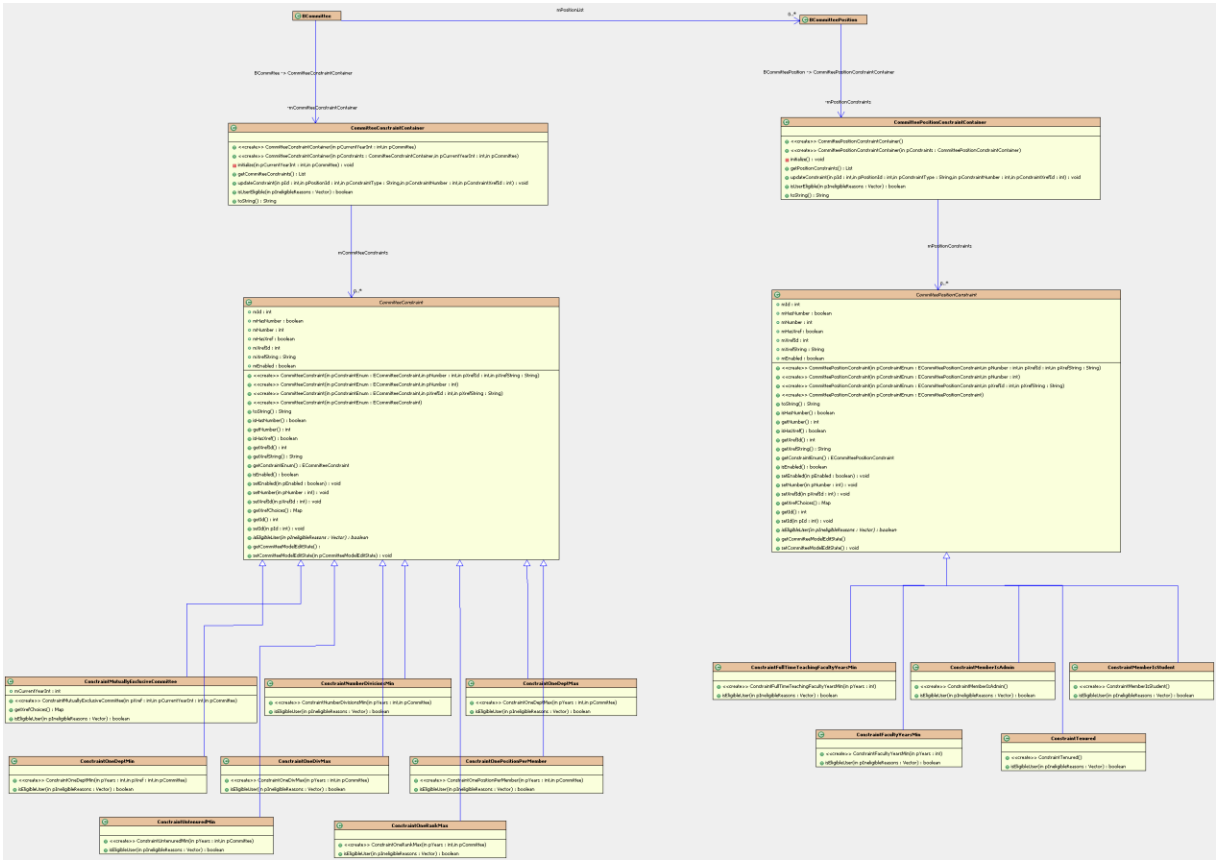

Figure 20: Entire Committee Model with Constraints

TESTCommitteeModel.java

```
package edu.cornell.committeetracker.committeeModel;

import java.util.Vector;

public class TESTCommitteeModel extends TestCase
{
    public TESTCommitteeModel(String pName)
    {
        super(pName);

        // This call uses the Factory Method Pattern to force every subsequent call to DataBaseConnection to use the
        // test dBase
        DataBaseConnection.getTestInstance();
    }

    public void testFiveYearHistory()
    {
        // BCommitteeModel utilizes the DataBaseConnection to get the Committee Structure
        Vector<BCommitteePosition> lPositionHistory = BCommitteeModel.getInstance()
                .getRecentHistoryForPosition("Committees of the Faculty", 2008, 5);
        assertEquals(2, lPositionHistory.size());

        lPositionHistory = BCommitteeModel.getInstance().getRecentHistoryForPosition("Committees of the Faculty", 2007, 5);
        assertEquals(2, lPositionHistory.size());

        lPositionHistory = BCommitteeModel.getInstance().getRecentHistoryForPosition("Committees of the Faculty", 2006, 5);
        assertEquals(1, lPositionHistory.size());

        lPositionHistory = BCommitteeModel.getInstance().getRecentHistoryForPosition("Committees of the Faculty", 2005, 5);
        assertEquals(0, lPositionHistory.size());
    }

    public void testIteratorOnDataModel()
    {
        BCommitteeModel lModel = BCommitteeModel.getInstance();
```

Figure 21: Code Sample of a JUnit Test