Leadership and Alumni Tracking System

A Manuscript

Submitted to

the Department of Computer Science

and the Faculty of the

University of Wisconsin-La Crosse

La Crosse, Wisconsin

by

Kirk Thomas Wienkes

in Partial Fulfillment of the

Requirements for the Degree of

Master of Software Engineering

May, 2010

Leadership and Alumni Tracking System

By Kirk Thomas Wienkes

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. Kenny Hunt Examination Committee Chairperson	Date
_____	_____
Dr. Kasi Periyasamy Examination Committee Member	Date
_____	_____
Dr. Mao Zheng Examination Committee Member	Date

ABSTRACT

Wienkes, Kirk, T., "Leadership and Alumni Tracking System", Master of Software Engineering, May 2010, (Dr. Kenny Hunt).

When an organization grows, it needs to evolve in order to effectively manage the increased amount of information required to effectively run. Today, this means making the leap from analog to digital. For the local chapter of Campus Crusade for Christ on the University of Wisconsin - La Crosse campus, increased growth within the ministry and with the scope of their target area has begun to cause logistical problems with managing student leaders and alumni. To solve this problem, a software engineer project named LARA (Leadership and Alumni Records Archive) has been developed to provide support with managing the internal structure of the organization including the management of applications for leadership, the management of the internal structure of student leaders, and the management of contact information for alumni. This paper describes the software process involved in creating this rich internet application, the issues faced during its creation, justifications for decisions made in the development process, and the current status of the project.

ACKNOWLEDGEMENTS

There are so many people I would like to acknowledge for being instrumental in the completion and success of my capstone. First, I would like to give my sincere thanks to my project advisor Dr. Kenny Hunt for his invaluable advice in developing the project and his innumerable contributions in writing this thesis. Special thanks go to Mark Brockberg, Erin Smith, and the rest of Campus Crusade for Christ ministry here for being both my project sponsors and for being a very influential part of my six years at UWL. I would also like to thank all the professors that I had in the computer science and math departments for all the valuable knowledge that they taught me while in La Crosse. Additional thanks goes to Dr. Thomas Gendreau, Dr. Kasi Periyasamy, Dr. David Riley, and Becky Yoshizumi for their direct contributions to my capstone. I would also like to thank the people who acted as consultants and advisors to me during this project especially David Paul Ellenwood, Tylor Ardnt, James Domagalski, and Jonathan Borre. I give my deepest thanks to my friends and family for being so supportive of me during this time and for giving me encouragement when I was in need of it. Your contributions to my capstone although not direct were not overlooked or unappreciated. Finally, I want to thank Renee for her tremendous patience, understanding, and kindness while I worked on this project. I would not have completed it on time without you. Thank you all for your support! I appreciate all of you very much.

TABLE OF CONTENTS

Page

ABSTRACT iii

ACKNOWLEDGEMENTS iv

TABLE OF CONTENTS v

LIST OF TABLES viii

LIST OF FIGURES ix

GLOSSARY xii

1. Introduction 1

    1.1 Background Information 1

    1.2 Project Scope 3

2. Software Process 5

    2.1 Life Cycle Model 5

    2.2 Software Developer Tools 6

3. Requirements 9

    3.1 Stakeholders 9

    3.2 Domain Analysis 10

    3.3 Prototype I 10

        3.3.1 Requirements Gathering and Analysis 11

        3.3.2 Non-functional Requirements 13

3.3.3 Use Cases Model 14
3.3.4 Cost Estimation 16
3.3.5 Summary 18
3.4 Prototype II 20
4. Design 22
4.1 Data Dictionary 22
4.2 General Design Goals and Constraints 23
4.3 Prototype I 24
4.3.1 Architectural Design 25
4.3.2 Database Design 29
4.3.3 Class Design 30
4.3.4 Graphical User Interface Design 34
4.3.5 Summary 35
4.4 Prototype II 35
5. Implementation 37
5.1 Prototype I 37
5.1.1 Technologies and Frameworks 38
5.1.2 Template Techniques 42
5.1.3 Code Generation 42
5.1.4 Summary 43
5.2 Prototype II 43
6. Testing 45
6.1 Prototype I 45
6.1.1 Test Data 46

6.1.2 Use Case Testing 47
6.2 Prototype II 47
7. Limitations 48
7.1 Deployment 48
7.2 Missing Functionality 48
8. Future Work 50
8.1 Prerelease and Deployment 50
8.2 Local Level 50
8.3 National Level 51
9. Conclusion 52
Bibliography 55
Appendix A: Use Case Diagrams 57
Appendix B: Use Case Point Analysis 76
Appendix C: Entity Relationship Diagrams 83
Appendix D: Class Diagrams 88

LIST OF TABLES

Table Page

Table 1. Results from Use Case Point Analysis 17

Table 2. Actors in Use Case Point Analysis 76

Table 3. Use Cases 81

Table 4. Technical Factors 82

Table 5. Environmental Factors 82

Table 6. Use Case Point Analysis Results 82

LIST OF FIGURES

Figure Page

Figure 1. CCC Member's Movement Management 15

Figure 2. Detailed Description of Functional Requirement 16

Figure 3. Software Components with Data Flow 26

Figure 4. Model-View-Controller Architecture 27

Figure 5. Basic Layers of LARA 27

Figure 6. ER diagram for Contact Information 29

Figure 7. Database constraints for the MOVEMENT table 30

Figure 8. Class diagram describing contact information entity 31

Figure 9. Campus class details from the design document 32

Figure 10. Class Constraints for the Event entity class 33

Figure 11. Screenshot of LARA's Graphical User Interface 34

Figure 12. Use Case Diagram describing the actors of the system 57

Figure 13. Use Case Diagram for User functionality 57

Figure 14. Use Case Diagram for User Contact Information Management 58

Figure 15. Use Case Diagram for System Security 58

Figure 16. Use Case Diagram for Administrator User Account Management 59

Figure 17. Use Case Diagram for Administrator Privilege Management 59

Figure 18. Use Case Diagram for Administrator Area Management 60

Figure 19. Use Case Diagram for Administrator Campus Management 60

Figure 20. Use Case Diagram for Administrator Place Management 61

Figure 21. Use Case Diagram for CCC Member Area Management 61

Figure 22. Use Case Diagram for CCC Member Campus Management 62

Figure 23. Use Case Diagram for CCC Member Place Management 62

Figure 24. Use Case Diagram for CCC Member Area Ministry Management 63
Figure 25. Use Case Diagram for CCC Member Campus Ministry Management 63
Figure 26. Use Case Diagram for CCC Member Movement Management..... 64
Figure 27. Use Case Diagram for CCC Member Team Management 64
Figure 28. Use Case Diagram for CCC Member Position Management..... 65
Figure 29. Use Case Diagram for CCC Member Training Management 65
Figure 30. Use Case Diagram for CCC Member Conference Management..... 66
Figure 31. Use Case Diagram for CCC Member Summer Venue Management 66
Figure 32. Use Case Diagram for CCC Member Vision Trip Management 67
Figure 33. Use Case Diagram for CCC Member Contract Management 67
Figure 34. Use Case Diagram for CCC Member Application Template Management 68
Figure 35. Use Case Diagram for CCC Member Discipleship Management 68
Figure 36. Use Case Diagram for CCC Member Appointment Management..... 69
Figure 37. Use Case Diagram for CCC Member School Year Management 70
Figure 38. Use Case Diagram for CCC Member Application Management 70
Figure 39. Use Case Diagram for Applicant Registration 70
Figure 40. Use Case Diagram for Applicant Application Management..... 71
Figure 41. Use Case Diagram for User Profile Management 71
Figure 42. Use Case Diagram for User Area Ministry Management 72
Figure 43. Use Case Diagram for User Campus Ministry Management 72
Figure 44. Use Case Diagram for User Movement Management..... 72
Figure 45. Use Case Diagram for User Team Management..... 73
Figure 46. Use Case Diagram for User Position Management..... 73
Figure 47. Use Case Diagram for User Training Management 73
Figure 48. Use Case Diagram for User Conference Management..... 73
Figure 49. Use Case Diagram for User Vision Trip Management 74
Figure 50. Use Case Diagram for User Summer Venue Management 74
Figure 51. Use Case Diagram for User Contract Management 74
Figure 52. Use Case Diagram for CCC Member User Profile Management..... 75

Figure 53. Use Case Diagram for CCC Member Contact Information Management 75
Figure 54. ERD for Yearly Historical Information..... 83
Figure 55. ERD for Events..... 83
Figure 56. ERD for Contact Information..... 84
Figure 57. ERD for Academic Information and Account Information..... 85
Figure 58. ERD for the Ministry Structure 86
Figure 59. ERD for Application Template 87
Figure 60. Class Diagram of Application Entities 88
Figure 61. Class Diagram of Contact Information Entities 88
Figure 62. Class Diagram for Ministry Entities..... 89
Figure 63. Class Diagram for User Entity 89
Figure 64. Class Diagram for User Profile Entity..... 90

GLOSSARY

AJAX

An acronym standing for Asynchronous JavaScript And XML used to describe the grouping of web technologies. AJAX provides a technique for client-side interaction with a web application server that will asynchronously send and receive data from that server without requiring a full page reload.

ASP.NET

The latest technology created by Microsoft for building dynamic web sites, web applications and web services. It is the next iteration on Microsoft's ASP (Active Server Pages) technology.

Application

A term within the application domain of LARA used to describe the document submitted by an applicant in response to an application template. An application is to an application template as an answer is to a question.

Application Template

A term with the application domain of LARA used to describe the document provided by the staff members to define the questions and sections that applicants are required to fill out in the application. Application template is to application as a question is to an answer.

Bean (JavaBeans)

A reusable software component in java that conforms to certain restrictions that provides an interface for which a page written using JSP or JSF can interact with java code on the server. Often beans are used to pass application domain data between JSP and JSF pages. A backing bean is a bean that acts as a container for a web page and provides mechanisms to store the page data and provides methods to initiate actions for the webpage.

CCC

An acronym used to describe the organization, Campus Crusade for Christ, for which the capstone was created for. This acronym will be used frequently in the remainder of this paper in lieu of providing the full name of the organization.

Class Diagram

A diagram described by UML that provides the static structure of classes, their attributes, their methods, and their relationships with other classes.

CRU

An acronym used to describe the most common national movement within the Campus Crusade for Christ organization. It is often used interchangeably with Campus Crusade for Christ and its acronym CCC even though it is a narrower term.

CSS

An acronym standing for Cascading Style Sheets used to describe a style sheet language that provides presentation formatting for a mark-up language most commonly HTML and XHTML.

EclipseLink

The open source JPA implementation developed by the Eclipse Foundation based on Oracle's TopLink framework.

Facelets

An open source web framework used to provide an alternate view handler for JSF. Facelets acts as a substitute for JSP which is JSF's default view handler. Facelets is XML-based and promotes the idea of reusable components in defining a view within JSF.

Hibernate

An open source JPA implementation used to provide an ORM library for Java developers to aid in the interaction between Java POJO classes and a database.

Hibernate Validator

An additional library alongside Hibernate used to validate data defined in entity classes used in Hibernate.

HTML

An acronym standing for HyperText Markup Language used to name a mark-up language that is the most popular way to publish web pages on the internet.

IceFaces

An open source framework and custom JSF tag library used to create rich internet applications. IceFaces provides additional user components that provide easily defined AJAX interaction between the client and server.

LARA

An acronym used to name the product being created in this capstone project. LARA stands for the Leadership and Alumni Records Archive.

JavaScript

A client-side scripting language used for basic computations, Ajax, and dynamic HTML pages.

JPA

An acronym standing for Java Persistence API used to facilitate interactions between Java classes, or entities, and a relational database. JPA is specification of this standard not an implementation. EclipseLink, TopLink, and Hibernate are popular implementations of the JPA standard.

jQuery

A lightweight cross-browser JavaScript library used to interact with an HTML document dynamically. It is the most popular JavaScript library used today.

JSF

An acronym standing for JavaServer Faces used to describe a web application framework intended to simplify the creation of web-based user interfaces. JSF uses a model-view-controller architecture and supports the use of events within a web application.

JSP

An acronym standing for JavaServer Pages used to describe a Java technology used to create dynamically generated web pages. JSP is the Java equivalent to ASP and PHP.

JTA

An acronym standing for Java Transaction API used to facilitate the use of transactions with in a Java application.

MySQL

An open source relational database management system used to provide data storage. MySQL is a very popular database used by many open source projects.

ORM

An acronym standing for Object-Relational Mapping used to describe a programming technique for mapping the conversion between objects within a programming language and the data stored in a relational database.

PHP

An acronym originally standing for Personal Home Page that current recursively stands for PHP Hypertext Preprocessor used to create dynamically generated web pages.

POJO

An acronym standing for Plan Old Java Object used to describe a regular Java object. This term is frequently used in discussions of persistence between Java objects and a relational database.

RichFaces

An open source framework and custom JSF tag library used to create rich internet applications. RichFaces provides additional user components that provide easily defined AJAX interaction between the client and server.

Ruby

A programming language generally used with Ruby on Rails to develop web applications. Ruby was intended to support an agile development lifecycle.

Seam

A web application framework developed by JBoss. This web application framework is meant to be used with JSF and provides additional features to manage a JSF web application emphasizing the interaction of JSF with JPA. Seam's main features include the introduction of a conversation context residing between an http session and http

request and uses a conversation as the base unit of length for maintaining a persistence unit.

Spring

A web application framework for Java used to provide additional features to Java web applications. Some features of Spring include inversion of control, aspect-oriented programming, transaction management, authentication and authorization.

SQL

An acronym standing for Structured Query Language used to manage data within a relational database.

SSL

An acronym standing for Secure Sockets Layer used to provide security and encryption between a client and a server. It is evident SSL is being used on a web site by the use of HTTPS.

Use Case Diagram

A diagram described by UML used to graphically represent functional requirements within a software system.

UML

An acronym standing for Unified Modeling Language used to describe a general-purpose modeling language used to specify and visualize various aspects of the software development process.

XHTML

An acronym standing for eXtensible HyperText Markup Language used to name a mark-up language that uses XML to define a web page. XHTML is a more restrictive way to define an HTML web page.

1. Introduction

Within any organization, growth is generally considered a good thing. Growth reflects a greater awareness of an organization's cause or purpose. Growth aids in an organizations ability to obtain resources and to achieve its goals. Growth allows an organization to broaden its scope and to tackle issues related to its original cause. Unfortunately, growth also brings challenges and requires flexibility. Like a teenager acclimating to a growth spurt, an organization must acclimate to its growth so it does not trip over its own feet, metaphorically speaking. As an organization grows, it must adapt to its new size in order to succeed and continue growing. It must provide new structures to guarantee efficiency in managing its workload. It must prevent unnecessarily duplicating work while ensuring that all necessary work is completed. One such growing pain for an organization is information and human resources management. As more information is generated and more personnel employed, the paper record keeping becomes increasingly cumbersome and unmanageable. Developing a digital records management system for these records becomes an important step in the maturing of an organization in the digital age.

1.1 Background Information

For the University of Wisconsin - La Crosse chapter of Campus Crusade for Christ growth in the ministry within the last decade has been a huge blessing, but this blessing has also brought challenges. As the chapter has increased in both student involvement and the full-time staff employed on campus, additional logistical problems have arisen in

managing student leaders within the organization and with keeping in contact with alumni who have moved away from La Crosse.

Over the last decade as the number of students involved in leadership has increased, it has become increasingly difficult to prevent student leaders from "falling through the cracks" in the current system. Despite the best efforts of the staff and senior student leaders, students are not receiving the support that they need to be effective within the organization. It has become apparent that it is important to ensure that the student leaders have had the proper training and have had the proper spiritual guidance to ensure their effectiveness in this CCC ministry as well as for their involvement in ministries later in life. Currently, the CCC staff has no means to track this type of important information for their ministry. Additionally the task of figuring out the structure of the next semester's leaders has become a huge logistical challenge. At present the whole process is done with pencil and paper from application to appointment.

As difficult as it is to track vital information for current student leaders, it is even more difficult to retain reliable information about alumni once they leave the ministry at graduation. At present, almost nothing is being done to ensure the ministry stays connected with these graduates. Since alumni have been a big part of the ministry, they have an interest in the current state of this organization on this campus. As an outlet for financial and prayer support, alumni at UWL have been left relatively untapped. As a result, the community of the greater La Crosse area has supplied the majority of the financial and spiritual support for their ministry. If the CCC movement could tap into the alumni resource, it could lead to an increase in available funds with which to support full-time staff, to pay for additional outreach events, and to provide better training for the student leaders.

Additionally within the last few years, the regional office has passed down a paradigm shift for the goals for this CCC chapter. In the past, this chapter was only responsible for ministering on the three college campuses in city of La Crosse, but now the local chapter is responsible for ministering to over 15 campuses in the greater La Crosse area from Winona to Mauston. They have been transformed from a campus ministry to an area

ministry. Consequently, it became evident that current process of managing the chapter's records was quickly becoming obsolete in the technology age.

As a result of these needs, the CCC staff desire to harness the power of technology to help them with their administrative duties so that they can better help support the student leaders and to free them up for additional time on campus ministering to college students. Due to the limited time CCC staff has for data entry, it is important that this system reduces the data entry as much as possible so that maintaining this system is not a logistic burden in itself. Additionally due to some of the CCC staff's limited knowledge of technology, it is important that the system be as simple and user-friendly as possible. The cost of learning and using the system should not outweigh its usefulness.

1.2 Project Scope

When examining the needs of the CCC chapter in La Crosse, three main aspects were identified as key components for this capstone project: a student leader application component, a student leader and ministry management component, and an alumni-tracking component. These systems would aid in retaining yearly records of the status of the ministry that would provide a historical record of the area ministry. It would also provide mechanisms to flow information between the components as a college student progresses from applicant to student leader to alumni.

The student leadership application component of the system will aid CCC staff in creating application templates for leadership and publishing them so they are available for students to apply for positions within leadership. Additionally, the system will help staff manage applications received, place applicants into available positions, and organize a hierarchical structure for discipleship of student leaders. Once this information is finalized, it will flow information into the next component of the system.

The student leader and ministry management subsystem will aid CCC staff in seeing the current status of their ministry. It will provide mechanisms for defining the structure of an area ministry into campus ministry, movements, and teams. Additionally, this

component will aid in the creation of meaningful reports about the current composition of their leadership, producing visualizations to help communicate the status of their leadership, and for maintaining records on their current student leaders. As student leaders graduate, their information would flow into the next component of the system.

The alumni-tracking component will aid CCC staff in maintaining contact information for alumni of their ministry. This system will be able to search through the historical records of previous student leaders and maintain a set of contact information for the alumni. This system will provide methods for alumni to sign up for newsletters and send updated contact information to the ministry. This system will be the basis for an increased effort to remain connected to alumni and to seek their support for the current ministry on their Alma Mater.

2. Software Process

Although the end result of a software development project is important, the process of how the software was developed is often more important. Software process is the essence of software engineering; it is what helps insure the success of the project and provides a framework for how software should be developed. Software process is difference between writing software and engineering software. Being defined as the "coherent set of activities of specifying, designing, implementing and testing software system", the main components of software process are requirements, design, implementation, and testing which are described in more detail in chapters three, four, five, and six respectively.

2.1 Life Cycle Model

Although the components of good software processes are the same regardless of your methodology, there are many approaches or perspectives for how and when these components interact with each other. While each project will eventually follow its own path in development, there are many generalized approaches for structuring that path through the software process; these generalized approaches are called life cycle models.

Choosing the life cycle model for a project is not a simple choice or a choice to be made lightly when beginning a software engineering project. Every life cycle model from the primitive build and fix to the classic waterfall model to the more contemporary agile model has advantages and disadvantages that greatly influence the success of a software development project. Many factors go into deciding what type of life cycle model to use in a project including but not limited to the project type, the project size,

time for development, budget, resources available, team size, team experience, organizational constraints, maintenance requirements, stability of requirements, and application domain.

After examining many of the factors for LARA, an incremental prototyping model was determined to be the best fit. The first major factor involved in this decision was the project type in conjunction with the developer's experience. Being a largely data-oriented, web-based project, the development of LARA required a broad knowledge of numerous topics including web standards such as HTML, XHTML, CSS, and JavaScript; relational databases topics such as SQL; and of a middle-tier language used to dynamically generate web pages such as ASP.NET, PHP, Ruby or Java. Having no experience in web development and web standards, a prototyping model was attractive for giving the developer time in the implementation phase to experiment with and learn the web standards, dynamically generated web pages, and how they connect with a middle-tier language. Since the developer has lots of experience using relational databases and interacting with them using multiple programming languages, a significant portion of the prototype would be of sufficient quality to prefer an incremental approach to prototyping rather than a rapid (or throw-away) approach. Additionally at the time of project proposal, the requirements for the project were very unstable and a prototyping method would allow development to begin on the parts of the requirements that were more stable while the less stable parts of the requirements were being defined and refined. Since the application domain was already split into components, it provided a clue that a prototyping method could be useful in incrementally developing the project. These components would also provide hints for defining the scope of prototypes so long as a global picture of the entire system was maintained in the design of these components for their eventual integration.

2.2 Software Developer Tools

Software engineering like any profession or trade is aided by the use of tools, and although "it's the poor craftsman who blames his tools" the right tool can mean the difference in a successful end product and an unsuccessful end product. The choice of software development tools is a decision that can greatly impact the speed, quality, efficiency, and proficiency to which a software engineer can craft his or her trade. The most fundamental software developer tool choice is selecting an IDE, which for Java means a very broad selection of nearly 20 IDEs of which the main IDEs are NetBeans, Eclipse, JCreator, IntelliJ IDEA and JDeveloper. Initially development began in NetBeans IDE due to previous experience and proficiency of the developer using that IDE, but as development continued NetBeans displayed a lack of plug-in support for many of the technologies and frameworks being utilized in the project. As a result, midway through the implementation phase of the first prototype, development was moved into the Eclipse IDE where it stayed for the remainder of the project. As described later, plug-ins such as Subclipse and JBoss tools were invaluable for the developer when implementing LARA.

Since the project utilized UML as a method for expressing the requirements and design, many UML software tools were investigated for the development of use case and class diagrams. The tools investigated were the NetBeans UML plug-in, Enterprise Architect, SmartDraw, MagicDraw, Visual Paradigm, Eclipse UML features, StarUML, VisualModeler, and Rational Rose. Since there was no available funding for commercial UML tools, the NetBeans UML plug-in was chosen since it was free, provided some code generation, was available on both PC and Mac, and provided a means to export images of UML diagrams without adding a watermark. Additionally since development was going to be in NetBeans initially, it was a logical chose for providing ease of integration between the UML diagrams in design and the code written in implementation.

In addition to the IDE and UML tools, many other tools were required and used in the development of this project. For source control, Subversion was chosen for its cross-platform support, for its IDE integration, for its availability for development, and for the experience the developer had in using this software. For client-side web debugging, the

Firefox plug-in Firebug was used almost exclusively with exception to some minor usages of the web developer tools in Chrome.

Although many development tools are useful and available in a commercial development environment, many useful development tools were unavailable in this project because the sponsor is a local chapter of a non-profit organization and lacked the IT infrastructure to support the use of these tools. Some of these tools included time tracking software, project management software, and bug tracking software. These tools were determined to have too high of a start-up cost to be used in this project since they would require a high level of set up that the developer could not support using the developer's own resources.

3. Requirements

When beginning a software engineering project, requirements engineering is the process by which requirements are gathered, analyzed, defined, and evaluated. Requirements define exactly what the software should do and should not do. This phase of development defines the scope of what the end product should be and should do. Defining requirements in a clear, concise, and accurate manner is fundamental to the success of a product. Since no matter how good the design, implementation, and testing of the product, it cannot and will not be used by the end user if it does not perform the tasks it was required to do

3.1 Stakeholders

One the first tasks in requirements engineering was define who the stakeholders are in the project. Stakeholders are the people who are vested in the resulting product being created including end-users, end-user managers, financiers, regulators, software developers, and IT professionals in charge of managing the system after development. Being the main end-users of the system and financier of the project, the CCC staff members are the primary stakeholder within the system followed by the applicants and student leaders who will be applying through this system. Additional stakeholders are the developer Kirk Wienkes and regional / national CCC offices. All of these stakeholders were identified during the initial requirements phase although the majority of the developer's interaction was with the CCC staff members, mainly Mark Brockberg. The developer made contact with the regional and national CCC offices to get them involved in the project with the hopes that it might be adapted to become a regionally or nationally

used system and to see if they were willing to provide some server support for the system. Little support was given to the project from the regional or national office even though they expressed some interested in the project. As a result, the IT support for this project was required to be at the local level. This added an unknown stakeholder into the mix since it was unclear from the beginning how and where the project was to be hosted. This oversight caused many problems later in development when trying to find affordable IT solutions for the client and may have changed some design decisions had this stakeholder been fully considered at requirements and design phases.

3.2 Domain Analysis

Domain analysis is the process by which a software engineer learns about the application domain to better understand the problem. This can be a very important step when the application domain of a project is unfamiliar to the requirements engineer. Fortunately for this project, the software engineer working on the project had previously been in CCC student leadership for two years; consequently, the software engineer was already a domain expert and did not require much additional domain analysis. On additional domain analysis that was required early in development was to identify the internal structure of a local CCC ministry. Although the developer had an understanding of the organization, a domain analysis was required to generalize an area ministry so that it could support many campus ministries over varying size and scope.

3.3 Prototype I

Since the initial requirements from the project proposal were unclear and not well defined, rigorously defining requirements for the first prototype was a high priority at the beginning of the project. Since the project used a prototyping method, not all requirements were defined at the start but it was necessary to rigorously define the first

prototype and to loosely define the other aspects of the system used in later prototypes so that the initial design could be easily extendable into the next prototypes.

3.3.1 Requirements Gathering and Analysis

Requirements gathering for the first prototype of this project began the week before the start of the fall 2009 semester, ran for nearly two months, and consisted of seven meetings with CCC staff lasting between one and two hours apiece. Since the sponsor did not have a clear picture of what they wanted to do but instead had a broad vision about what the software could solve, requirements gathering proved to be a lengthy and difficult task. After brainstorming a list of initial possible requirements, the software engineering refined the requirements that were thought to be the most useful for the CCC chapter based on previous knowledge of the application domain. As a result after the first few meetings, most of the next sessions began with the software engineer asking a list of questions derived from the analysis preformed on the requirements gathered in the previous session mainly asking for clarifications about ambiguous requirements and asking about possible extensions or restrictions on some features of the system. Through the first seven meetings with the sponsor, the software engineer weeded through all the ideas about requirements and discovered what the sponsor really wanted and more importantly what the sponsor really needed. This was the focus of the first half of the requirements phase.

As ideas were brainstormed and analyzed, one of the major requirements was investigating possible avenues for integration with existing systems with the CCC organization and with information available to the local CCC chapter using their status as a campus organization. After making contact with a few individuals within the regional and national CCC offices, the developer was informed that access to records for their system would not be possible unless the project was taken on from the regional or national level. As a result, the system needed to be developed without access to these information sources so that it could run independently if it was not picked up by the

regional or national office. Additionally, the developer contacted the Alumni Association and the registrar's office to see what kind of the information would be possible to get and how that information could be obtained. Both organizations were willing to share some information about students including some non-personal information about all the students enrolled at UWL from the registrar's office and the contact information of Alumni who indicated that they were previously involved with CCC while at UWL. An important note is that this information is not readily available to just anyone asking for it but was only available to the CCC chapter since they were a registered campus organization and only could be request and obtained by their faculty advisor. As a student, they were only able to inform the developer about the type of information that they could provide the faculty advisor and the format for which they could provide it in but could not provide that information directly to a student. Resulting from this investigation, the developer determined that this information could be a secondary source for information but would not be a primary source since it was unable to be easily accessed programmatically meaning it was not be possible to have access to views within their database systems. The final investigation about integration was about some software that the local chapter uses to track donor information, a system called TNT-MPD. After being given a demo of the software, the developer determined that it could be possible to share some information between the systems if LARA outputted information using the proper format for import into TNT-MPD. Having a personal relationship with the developer of TNT-MPD, Mark Brockberg seemed confident that he could provide the developer with access to any materials regarding the format of data within the system which turned out to be reading to and from an encrypted access database on the machine itself. Although it was determined that there would not be much useful interaction between the systems supporting some type of import and export feature between the two system could prove useful so it was added as the only practical integration with another software system.

As the main picture of the system began to emerge, it became clear that building a robust and adaptive system would require defining the generic structure needed to define

the structure of a campus ministry. Such a generic structure would require the development of a model that would support large campus ministries, small campus ministries, the transition of between the two, and historical snapshots of that structure for maintaining records of the changing structure of the ministry over the years. This provided a challenge for the developer. In the requirements phase, the developer needed to define what information would be required to accomplish this generic approach to an area ministry. Although the definition of this structure would be difficult, this problem would provide an even greater challenge in the design phase on how to effectively represent this information.

3.3.2 Non-functional Requirements

Concurrently when gathering and analyzing functional requirements, the developer gathered non-functional requirement that would define the parameters under which the system would need to be developed. Unlike functional requirements, which constructively define the services the system should provide and how the system should behave, non-functional requirements define constraints on how the system can be developed or implemented. Common non-functional requirements include constraints on timing, standards used, organizational policies, security, portability, ease of use, and reliability. Through the meetings with the sponsor, the following non-functional requirements were identified.

- The system must be web based
- The following operating systems must be supported: Mac OS-X and Windows
- The following browsers must be supported: Firefox 3.5, Internet Explorer 8, and Safari 4
- Many users must be able to use the system at the same time.
- The database must block all unauthorized access
- A login is required to access the main part of the system
- Passwords must be encrypted

- The system must support SSL in its web implementation for protecting personal information

3.3.3 Use Cases Model

Once the requirements were gathered and analyzed for the first prototype, a use case model was developed to define the functional requirements of the system. The first step in developing this use case model was identifying all the actors in the system. In a use case model, an actor defines a role played by an external entity that interacts with the system. This includes types of users who will be interacting with the system and other computer systems that will interact with the system. In LARA, the primary actors of the software system include administrators, CCC members, and students. Some of these actors were decomposed even further since there are multiple types of CCC members including CCC staff, CCC intern, and CCC volunteer. Additionally, the role of student leader was broken down by their current place within the CCC student leadership process. Using that logic, a student could be an applicant, a student leader, or an alumnus. In addition to the primary actors in LARA, the first prototype also would interact with a database which would also be considered a secondary actor since it is an external entity that interacts with the system but does not initiate use cases as a primary actor does.

Once the actors were defined, the use cases or important functionalities were defined and their interaction with actors and other use cases were noted. To illustrate an example, Figure 1 describes the use cases and actors involved with a CCC member managing movements within a campus ministry.

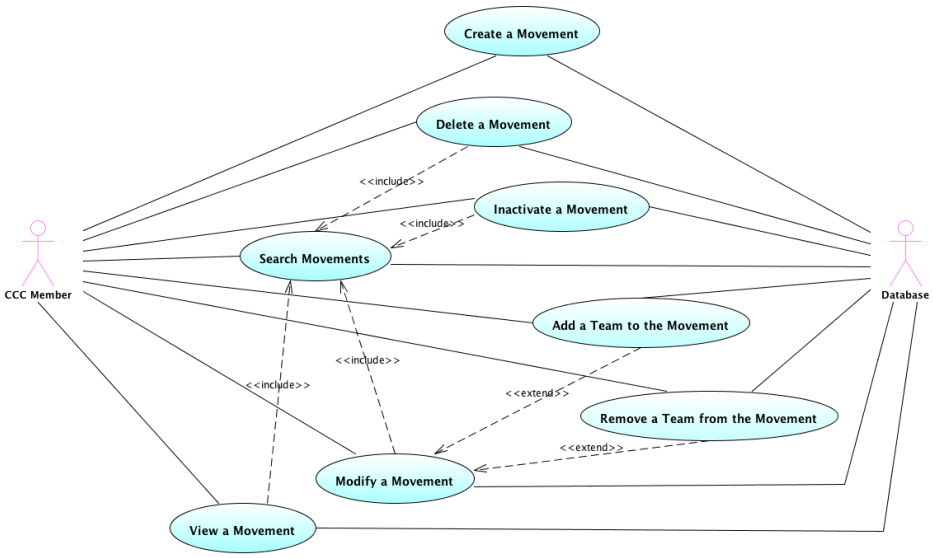


Figure 1. CCC Member's Movement Management

Even though the use cases defined in Figure 1 only describe some basic operations on a movement, the use case diagram uses two actors and has numerous dependencies between use cases. Being only a small aspect of LARA, this use case diagram hints at how many functional requirements would be required to define then entire system. In total, the use case diagrams specifying the first prototype included 9 actors and 206 use cases

Along with the use case model, a textual requirements document was written to explain the functionalities in the use case model in further detail. For each use case, a functional requirement was written describing the use case which entailed enumerating the following information: a unique index, name, priority, purpose, input parameters, action, output parameters, exceptions, remarks, and cross-references to any other functional requirements. An example of one such functional requirement is given in Figure 2 and describes one the use cases given in Figure 1.

Index:	CCC.Movement.7
Name:	Add a Team to the Movement
Priority:	Medium
Purpose:	To add a Team to an Movement
Input parameters:	Movement Id Team Id

Action:	• Ensure that the user invoking this functionality has the proper privileges to invoke this functionality
	• Ensure that the user invoking this functionality is currently logged into the system
	• Ensure the input parameters are valid
	  o Ensure that the Movement Id exists in the database
	  o Ensure that the Team Id exists in the database
	• Ensure Privilege / Database consistency
	  o Ensure the Movement does not already have the Team
	• Add the Team to the Movement in the database

Output parameters:	None

Exceptions:	• User invoking this functionality does not have the proper privileges
	• User invoking this functionality is not logged into the system
	• Invalid Input Parameter
	  o Movement Id does not exist in the database
	  o Team Id does not exist in the database
	• Privilege / Database Inconsistency
	  o Movement already has this Team

Remarks:	None
Cross-references:	CCC.Movement.5, CCC.Movement.2, CCC.Team.5

Figure 2. Detailed Description of Functional Requirement

Refer to Appendix A for a listing of all the use cases diagrams and refer to the requirements document for a detailed explanation of the use cases and functional requirements.

3.3.4 Cost Estimation

After the development of the use case diagrams, a use case point analysis was performed to estimate the cost of the project. A use case point analysis is a method to derive cost estimation for an object-oriented system defined using a use case model. It is a cost estimation model similar to a function point analysis. With the use case model, the developer would be able to get an estimate for the time required to develop the system. The basics of use case analysis are that the cost of project can be estimated by a weighted sum of actors and use cases based on a complexity factor that is then scaled by environmental and technical factors. This will provide the developer with an adjusted use case point measurement which can be transformed into an estimate of effort in person

hours given a constant to define the number of person hours per use case point. This constant is a subjective estimate made by the use case analyst depending on the size of each use case and how finely the use cases where defined within the system. For LARA, that constant was estimated to be two person hours per use case since the use cases were defined on a small scale.

One of the most difficult parts of a use case analysis is correctly identifying the relevance of the technical and environmental complexity factors. With 21 different factors contributing to the scaling factor, the scaling factor has a tremendous impact on the resulting estimation and thus providing inaccurate relevance for each factor can cause the estimate to be extremely inaccurate. For LARA, the most important factors that increased the cost estimation were because LARA was a distributed system, required a high usability, needed a high portability, had unstable requirements, and required special security for privacy concerns. The factors that decreased the cost estimation the most were because the developer had a strong familiarity with UML, experience with object-oriented programming, and was highly motivated. Additional factors that affected the cost estimation were because LARA did not require direct access for 3rd parties, had low performance constraints and did not require special training to use. The main factor that this use case analysis did not reflect was the inexperience of the developer with web based technologies and web application frameworks. That inexperience proved to increase the time for development significantly.

Unadjusted Actor Weights (UAW)	25
Unadjusted Use Case Weights (UUCW)	1045
Unadjusted Use Case Points (UUCP)	1070
Technical Complexity Factor (TCF)	1.11
Environmental Factor (EF)	0.536
Use Case Points (UCP)	636.6072
Effort (Person Hours)	1273.2144
Effort (Man Weeks)	31.83036
Effort (Man Months)	7.275510857

Table 1. Results from Use Case Point Analysis

Based on the use case point analysis, the developer determined that it would take about seven months to complete the first prototype. This was the first clue that the original scope of the project would be too large to complete within the normal capstone time frame for a full-time graduate student. The results of the use case analysis can be found in Table 1 and the entirely of the use case analysis can be found in Appendix B.

3.3.5 Summary

In summary, the requirements document for this first prototype included 8 actors, 206 use cases, 268 functional requirements, 17 non-functional requires, 186 pages and was developed over a two month span. The following provides an overview of the functionalities in the first prototype.

- Users
    - Users can login, logout, change password.
    - Users can view and modify contact information
    - Users can view and modify user profile including academic information, future plans, training received, conferences attended, summer venues attended, vision trips attended, current contact information, and permanent contact information.
- Administrators
    - Administrators can create, modify, delete, view and search user accounts.
    - Administrators can grant, revoke, view and search privileges.
    - Administrators can create, modify, delete, view and search areas.
    - Administrators can create, modify, delete, view and search campuses.
    - Administrators can create, modify, delete, view and search places.
- CCC members
    - CCC members can view and search areas.
    - CCC members can modify, view, and search campuses.

    - CCC members can create, modify, delete, view, and search places.
    - CCC members can modify and view their area ministry.
    - CCC members can create, modify, delete, view and search campus ministries, movements, teams, and positions in their area ministry.
    - CCC members can create, modify, delete, view and search training and contracts in their area ministry.
    - CCC members can create, modify, delete, view and search conferences, summer venues and vision trips in their area ministry.
    - CCC members can create, publish, copy, modify, delete, view and search application templates for their area ministry.
    - CCC members can create, copy, modify, view, and search school years and define discipleships and appointments in those school years for their area ministry.
    - CCC members can modify other user's user profiles. Additionally, they can keep notes on them, which they can only see.
- Applicants
    - Applicants can register for a user account.
    - Applicants can start, submit, modify, cancel, view, and search applications for application templates within their area ministry. This includes answering questions on applications, defining desired positions, and agreeing to contracts.

Over the course of the first requirements phase, certain techniques were developed and identified as important. First, the developer found it crucial to take good notes during meetings with users since failing to do so during one meeting resulted in the developer having the ask some of the same questions again since the answers were not recorded and could not be remembered. Following from this lesson, the developer found it handy to create an audio recording of the meetings so that he would have a record of the conversation that proved very useful in filling the gaps of the meeting notes and that was

indispensible when analyzing the requirements gathered in the meeting. Overall, this requirements phase took a substantial amount of time but was needed in developing a clear picture of what the system needed to do and provided the basis for the success of the remainder of the project.

3.4 Prototype II

The requirements phase for the second prototype began in late January 2010 and is currently on going. So far requirements have been gathered during part of two meetings in February and from requirements leftover from brainstorming in the first prototype but did not make it into the final first prototype requirements document. This prototype mainly consisted to the alumni component of the system and some feature additions to the student leadership application component and the student leader and ministry management component. This prototype will focus on adding additional reporting features and visualizations to data within the system. The exact scope of this phase is not yet complete and thus the following are a list of possible requirements for the second prototype.

- Administrators
    - Administrators will be adding users into the system as a way to bypass the traditional apply for leadership.
- Student Leadership Application Component
    - Application templates should have restrictions defined by CCC members, which limit the students who can see these applications for applying. Some restrictions to implement are based on years involved and gender.
    - Applicants should be able to add pictures to their user profile.
- Student Leaders and Ministry Management Component

    - CCC members should be able to make reports with corresponding charts and graphs drawing from current and historical information about their area ministry.
    - CCC members should be able to generate visualizations about the structure of their ministry including area of responsibilities, discipleship graphs, name-learning visual aids, and yearly pictures of the composition of their student leadership.
    - The system should store information about the staff team and the staff coaches for each team within an area ministry.
- Alumni-tracking Component
    - CCC members will be able to update contact information for an alumnus and export alumni newsletter information to TNT-NPD.
    - CCC members will be able to graduate current student leaders into the alumni portion of the system.
    - CCC members should be able to hold, organize, and query historical leader information.
- Other
    - The developer should provide additional GUI features to leverage the pictures added to the system.
    - The developer should change the estimated graduation from a date to a semester within a school year.

The extent and scope of the second prototype's requirements will be dependent on the current developer's free time and on whether another developer continues the work begun by the current developer.

4. Design

Following the requirements phase, development proceeded into the design phase. Software design is an iterative, problem-solving process by which implementation details are added to the requirements; constraints are imposed by quality, cost, time, platform, and process requirements; and general principles are followed to insure the quality of the resulting product. While the requirements provide the "WHAT" aspect of the system, the design phase provides the "HOW" aspect of the system. There are many important decision that are made during the design phase that direct the type of system that is developed, and making poor decisions early on in the design phase can set the project up for large delays if they need to be changed later in the software process. The design should provide a strong yet flexible foundation for the implementation. For LARA, the design uses UML as a design notation and is described in the remainder of this chapter. Due to the experience of the developer, UML class diagrams were used for describing features of the system and Entity Relationship Diagrams (ERD) were used for describing the database aspects of the system design.

4.1 Data Dictionary

With complex systems, it is difficult to maintain the consistency of data across the entire system especially with respect to naming conventions. In order to help with consistency, a data dictionary can be used to help maintain a record of the data managed by a project. A data dictionary lists all the names used in the system models and describes all the entities, relationships, and attributes within the system. It would provide a reference guide useful for maintaining consistency about the data-related constraints

across all aspects of development. Due to the usefulness of a data dictionary and the amount of data LARA needed to manage, the developer decided that a data dictionary's usefulness justified its creation in development. The developer wrote a data dictionary starting in the first prototype's requirements phase and kept it up-to-date throughout the first prototype's design phase. The resulting data dictionary consisted of 42 entities each with attributes, some of which defined relationships between other entities. A brief description was written for each entity describing what the entity was and its purpose; additionally, for each attribute describing what kind of information stored. The data dictionary was invaluable when developing consistency between the database, the class entities, and the constraints that validate the data within them. These details and many more are described in length in the design document developed for the first prototype.

4.2 General Design Goals and Constraints

Since aspects of the design would be shared throughout all the prototypes of the project, the developer though it necessary to provide some general design goals that should be followed throughout the designs of all the prototypes. Overall, the system should be developed using an object-oriented approach using the diagrammatic notations of UML with associated textual descriptions. At the architectural level, the developer thought that an MVC design pattern should be implemented as a general design principle for the system. This design pattern would provide a great deal of flexibility when upgrading the system since it would be developed in layers that could be switched out if a technology switch was required later in development. Initially, this design goal was made in case the project was taken over by another development team within the CCC organization after this developer completed the first few prototypes. Although no CCC development team expressed interest in working on the project, this design goal proved quite valuable as the developer added additional frameworks in the first prototype. These frameworks and the choices surrounding them will be discussed at greater length later in the fifth chapter.

Since with the technology of today it would be considered unreasonable to require applicants to download software in order to apply for leadership within Campus Crusade, the project would only make sense as a web application granting access to anyone with an internet connection. Thus, concurrency and data consistency would need to be considered when using the web technologies. Although these would have been an issue with a desktop application implementation as well, it was important recognize their importance and seek solutions through web technologies since these are issues in any present-day, data-centric web application. Finally in order to provide broad support for various applicants, the system must be accessible from either a Windows or Mac system and must be supported by the major browsers including Internet Explorer 8, Safari 4, and Firefox 3.5, a requirement that most web application should follow if accessed beyond the shelter of a company IT policy.

Keeping in mind the possible future interest of national or regional CCC, the developer imposed some additional constraints for the design and implementation of based on the capabilities of these offices. As a result, only the following development languages were considered for the development of this project: Java, PHP, and Ruby. Furthermore, the use of free software was preferred over the use of costly proprietary software whenever possible since initially the local CCC office would host this project. Consequently, MySQL was chosen to be used and thus all database design work needed to be aware of the limited features of MySQL versus a proprietary relational database management system such as Oracle. Without triggers, sequences, and a powerful set of constraints, a MySQL implementation of the database could not rely on the aforementioned.

4.3 Prototype I

Given that an incremental prototyping life cycle was chosen for this project, the initial design for the project was very important to success of later prototypes. Therefore, the initial design needed to provide a generalized design that would be generic enough to allow for expansion in later prototypes since both design and implementation would be

reused between prototypes. The goal of the initial design was to find the balance between an extremely generic design that would be difficult to implement in a timely manner and a very specific design that might require major design changes in later prototypes. The initial design needed a Goldilocks' type approach not too generic, not too specific, but just right. Accordingly, the initial design for the first prototype needed to meet the requirements for first prototype, but also plan for the additional features that might be required in a second or third prototype. Since the developer had the most experience with the databases, the focus of the first prototype's design would data-centric and would focus on the components of the system surrounding the database. This would provide a solid base for the first prototype and would allow the controller and view layers of the system the flexibility required for the developer to learn web design.

4.3.1 Architectural Design

An architectural design describes how the software system will be divided into subsystems and components and defines the connections and interactions between them. Within LARA, there were two main methodologies used in dividing up the system. The first divisions were created by the natural groupings of functional requirements into the main components of the system: a student leadership application component, a student leader and ministry management component, and an alumni-tracking component. The other methodology used to define division was the Model-View-Controller (MVC) design pattern which describes a way to divide software by splitting up the logic required to represent data, view the data, and the manipulate the data in the system. The MVC approach provides a low-coupled, high-cohesive framework for writing robust, maintainable software.

Architecturally, breaking the system into a student leadership application component, a student leader and ministry management component, and an alumni-tracking component provided a logical order to develop the software and provide hints into the data flow of the system. A visual approximation of the flow of data into and through the system is

provided in Figure 3. In the diagram, the arrows represent the flow of data while the ovals represent users and the rectangles represent the three main components of the system. Although the staff members have that ability to modify and add data to the system, the majority of the data entry is performed by the users on the left-hand side of the diagram: applicants, students and alumni. This data once entered trickles down into the other main components as a user goes from applicant to student leader to alumni. As a result, it was reasonable to develop the system starting with the student leadership application system and moving down into the other components as the more requirements are added to each new prototype.

Figure 3. Software Components with Data Flow

In order to create a system that would be maintainable and consistent throughout all prototypes, discipline was required to develop the product using a proven architectural design pattern, the MVC design pattern. For a graphical representation of MVC, refer to Figure 4 which shows how the components of an MVC architecture interaction with each other. Although it would restrict the ways the system could be designed, the design pattern would force code to be written in small, maintainable chucks by forcing a separation of presentation information, business logic, and data persistence logic. This separation provides for easier maintenance, creates reusable code, and makes it much easier to switch out an entire layer if necessary. For an in-depth explanation of the Model-View-Controller design pattern, please refer to one of the many resources and guides available on the internet or in your local library.

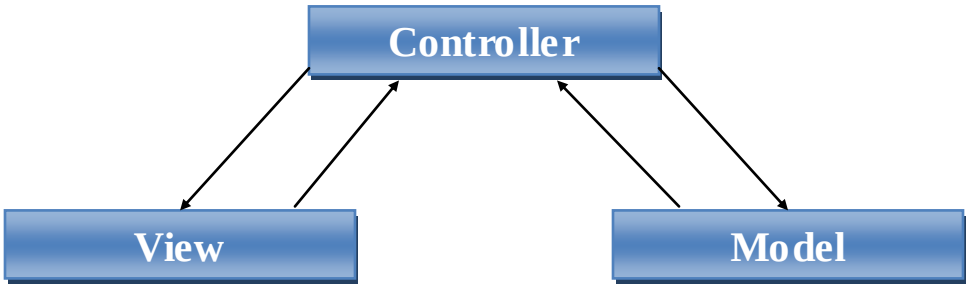


Figure 4. Model-View-Controller Architecture

With a basic understanding MVC, discussion of MVC design used in LARA can continue with an examination of the layers that comprise LARA's implementation of the MVC design pattern. The layers defined within LARA are the entity layer, the data access layer, the action layer, and the presentation layer. Figure 5 shows the layers defined in LARA.

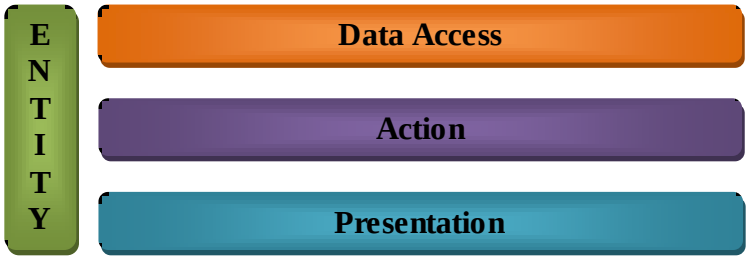


Figure 5. Basic Layers of LARA

The entity layer describes the application domain data passed between the data access, action, and presentation layers of LARA. This layer provides the classes that encapsulate the data stored in the database that are manipulated and passed between the other layers. For example, this layer would provide a class representation for an application domain object such as an application, a conference, and a user profile among other data defined within the data dictionary. This layer is comprised only of POJO style classes and only

stores data without regard to the processes by which the data stored is governed. These classes provide the vessels for the internal representation of the application domain within LARA. The structure of each class is simplistic providing instance variables with getter and setter operations along with a few constructors. The complexity of this layer is not found in the classes themselves, but with their interaction with each other and how this affects the database through the data access layer.

Next, the data access layer provides all the functionality associated with passing information between local entity objects and the database that permanently stores all the records associated with LARA. This layer would house all the SQL statements for the project providing relative ease in changing database providers. For example, this layer would provide the functionality to create, read, update and delete entities to and from the database. All searching functionalities also have their implementation found within this layer. With respect to the MVC architecture this layer provides the majority of the model layer.

The action layer provides the business logic associated with the system. All of the validations required to process data and all the steps required in a business use case fall under this layer. For example, this layer provides the logic required to start an application from an application template provided by the area ministry CCC staff. It would check to see if the user can apply to this application template, creates the necessary entity objects, and uses an object in the data access layer to permanently record that an application was started and store all related information in the database. Most of the application domain logic outside of the data model can be found in this layer of LARA. With respect to the MVC architecture this layer provides the majority of the controller layer.

The presentation layer provides the logic required to show the user the system information and capture user interactions with the system. Being the only layer that a user actually sees and interacts with, the presentation layer is important for providing a pleasant user experience with LARA. For LARA, this was the layer that dynamically generates HTML pages based on entities in the entity layer and actions performed in the

action layer. Very little was designed in this layer during the first prototype due to the developer's lack of experience in web presentation technologies. With respect to the MVC architecture this layer provides the majority of the view layer.

4.3.2 Database Design

The database design for a data-centric project is a very important since it ultimately provides the permanent storage for the data within the system. As mentioned earlier, the developer was required to use a MySQL database. As a result, the developer was limited to those features available in MySQL which does not include triggers, sequences, and a complete set of constraints. Although there is less expressive power with MySQL when compared to Oracle, MySQL is free, readily available, and widely used in the open source community consequently making MySQL an attractive database provider for a non-profit organization.

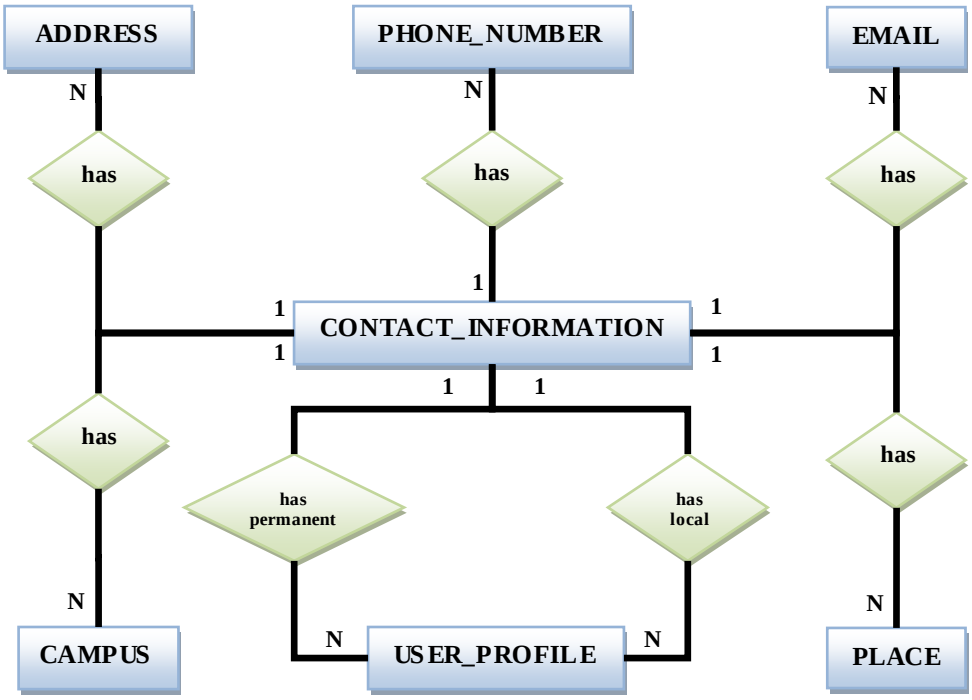


Figure 6. ER diagram for Contact Information

To begin the database design of LARA, multiple Entity-Relationship Diagrams were developed describing the entities from the requirements document and data dictionary. The process of database design took multiple iterations before a generic database design was developed that followed the proper database design principles and provided the expressive power defined in the requirements. Figure 6 shows one of the entity relationship diagrams developed in this phase and defines the entities dealing with contact information. For the other ER diagrams developed in LARA, refer to Appendix C.

Once the ER diagrams were analyzed and finalized, a database was generated. The final database design for the first prototype consisted of 58 database tables. In addition to the database design, the developer developed constraints on the data within the database to help insure data integrity. Within the design document, the developer provided many tables like Figure 7 that described information relating to a single database table and the constraints on the information stored in that table. Although some of the constraints were not expressible using MySQL, the developer thought it wise to define them anyways since they would need to validate data somewhere in the project and those constraints would be very useful if the project was moved to a different type of database that provided better constraint support than MySQL.

MOVEMENT

MOVEMENT_ID	ID_TYPE	----	PK	NO
NAME	TINYTEXT	----	----	NO
DESCRIPTION	TEXT	----	----	NO
CAMPUS_MINISTRY_ID	ID_TYPE	FK [CAMPUS_MINISTRY]	----	NO
NATIONAL_MOVEMENT_ID	ID_TYPE	FK [NATIONAL_MOVEMENT_ID]	----	NO

1) 1 <= NAME length
2) 1 <= DESCRIPTION length
3) UNIQUE(CAMPUS_MINISTRY_ID, NATIONAL_MOVEMENT_ID)
4) UNIQUE(CAMPUS_MINISTRY_ID, NAME)

Figure 7. Database constraints for the MOVEMENT table

4.3.3 Class Design

After the architectural design was completed, the development began on the class design concurrently with the database design. The class design takes the architectural design a step further in providing implementation details to the requirements of a project. While an architectural design gives a general idea of how components are structured and interacts with each other, the class design takes the design and provides the internal structure and method signatures of the classes used within the components of the system. The majority of the class level design work for the first prototype was done in the entity and data access layers. The action and presentation layers were left fairly empty because the developer lacked of experience in web development to know how these layers would be structured in a web environment as contrasted to a desktop environment where the developer had experience.

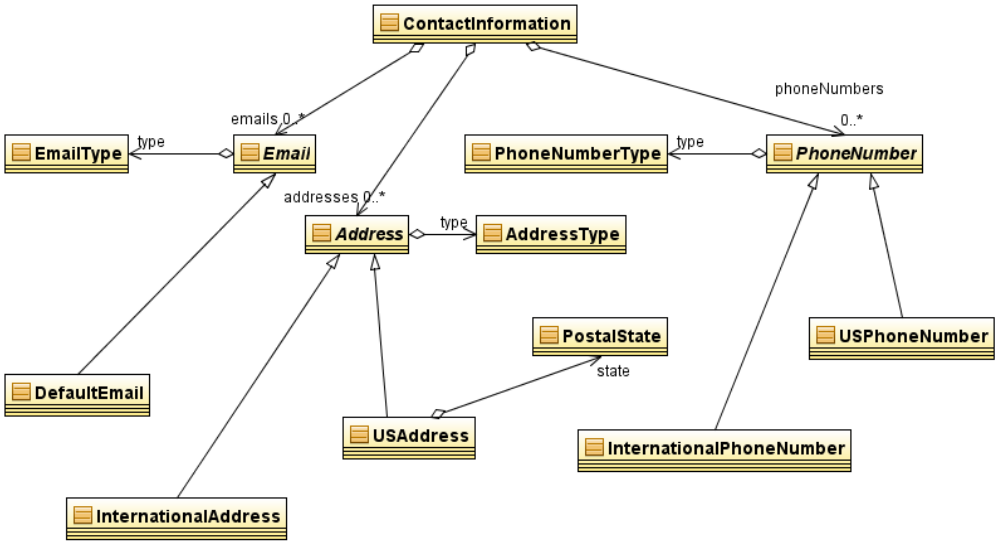
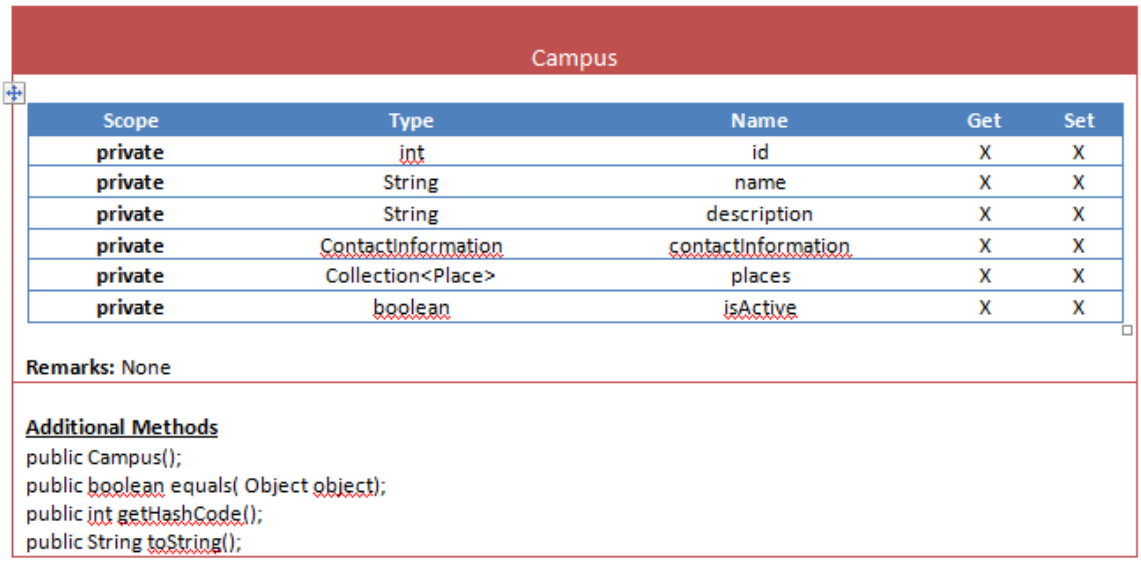


Figure 8. Class diagram describing contact information entity

The entity classes in the entity layer were developed iteratively with the database design. These entities map closely to the database design and only vary to provide some convenience for the programmer and their interactions with other layers within the system. Figure 8 describes the entities related to contact information that parallel the

database design described in Figure 6. In addition to high-level class designs, the design document also provides a class level look at the methods and instance variables associated with each entity. Figure 9 shows an example of one of the class diagram defining the internal components of an entity class. For all the class diagrams for entities in LARA, refer to Appendix D of this document and for all the detailed class designs please refer to the design document.



Campus

Scope	Type	Name	Get	Set
private	int	id	X	X
private	String	name	X	X
private	String	description	X	X
private	ContactInformation	contactInformation	X	X
private	Collection<Place>	places	X	X
private	boolean	isActive	X	X

Remarks: None

Additional Methods

public Campus();
public boolean equals(Object object);
public int getHashCode();
public String toString();

Figure 9. Campus class details from the design document

For entity classes, an additional section was developed in the design document to provided constraints on the data held inside the entity classes. While this covered similar information to the database table constraints, these constraints were gear toward the object oriented design of the entity classes. This additional section provided clues to the implementer for the constraints that needed to be checked on the entity classes before being persisted to the database. Figure 10 provides an example of one of the tables that described the additional constraints on the entity classes found in the design document.

Event

EventId	Id Type	Unique in the system Not Null
AnnualEvent	AnnualEvent	Not Null
SchoolYear	SchoolYear	Not Null
Name	String	1 <= Length <= 255 Not Null
Description	String	1 <= Length <= 65535 Not Null
ActualAttendance	Integer	0 <= Value
StartDate	Date / Time	Not Null Value < EndDate
EndDate	Date / Time	Not Null StartDate < Value
DateAdded	Date / Time	= CURRENT_DATE @ creation Not Null
DateModified	Date / Time	= CURRENT_DATE @ last modification Not Null

Figure 10. Class Constraints for the Event entity class

Moving away from the entity layer, the design document provided a template for how to structure the most common type of class in the data access layer of LARA, the data access object. The data access object defines all database operations on an entity class including adding, deleting, updating, searching, and getting. For example, the ApplicationDao interface defines all of the addition, update, delete, get, and search operations on the Application entity class. The data access object was designed as an interface with an implementation for easy in switching database providers. As a result of using this interface, a mechanism need to be devised to gain access to a data access object implementation without knowledge of the exact implementation classes being used with respect to the action layer. In order to solve this problem, an additional class was developed using the factory design pattern and was named the DaoFactory. This class provides methods to get access to the data access objects with direct knowledge of the implementation of the data access objects. Thus, only the DaoFactory class would need to be changed if the developer desired to switch implementation of the data access objects.

4.3.4 Graphical User Interface Design

One of the most important aspects of software design is the graphical user interface design. Since the graphical user interface is the part of the system that is seen and used by people, it is one of the most difficult to do properly. A software product with a great user interface and limited features will get more use than a software product with a poor user interface and amazing features. A poor user interface is the biggest reason that great software is never used.

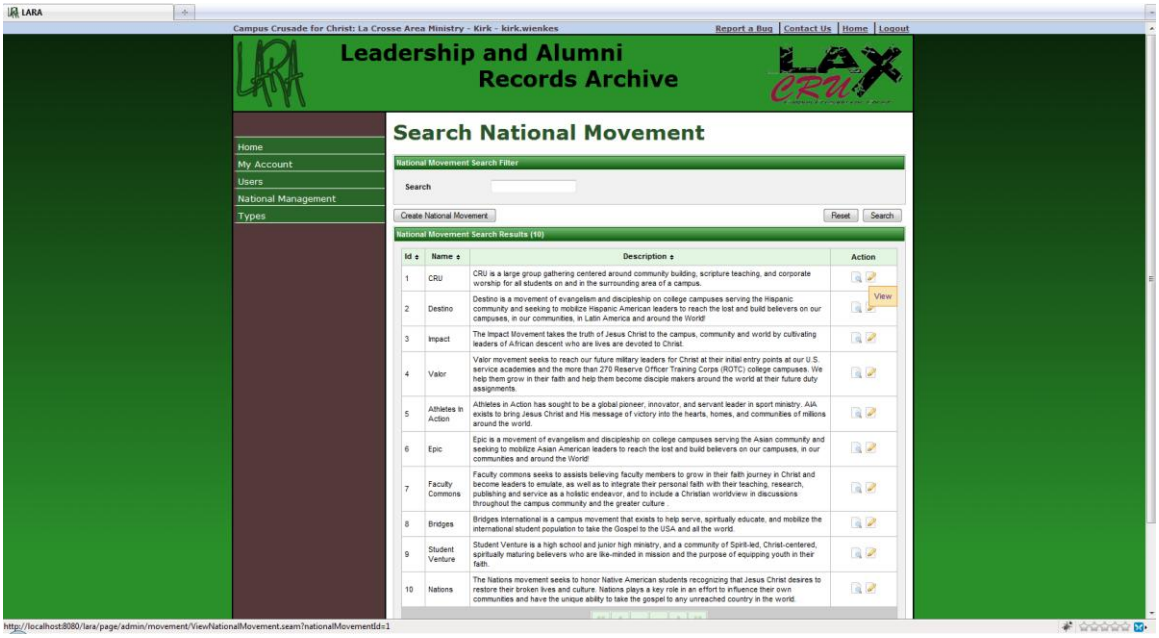


Figure 11. Screenshot of LARA's Graphical User Interface

For LARA, the user interface was a topic for discussion during many of the meetings with the client. Having limited experience with a web frontend, the developer had few ideas or preconceptions about how the web frontend show look or behave. As a result, the developer spent multiple hours with the client discussing the types of websites that the client liked and disliked. The developer walked through some of the websites the client frequently used in order to discuss features in a UI that the client found most useful

and designed the user interface based on these types of features. The following is a list of websites that the client and developer analyzed for user interface features: allcallings.org, missionalteamleader.com, firstfree.org, laxcru.org, uppermidwest.org, newegg.com and amazon.com. From these interactions, the type of website that the client wanted was a clean, user-friendly website without a lot of bells and whistles where the functionality is clearly seen and accessible. During another meeting that discussed UI choices, the developer learned that the client would like a vertical, menu-based, main navigation and that the website should have a green color scheme to match the CRU website at laxcru.org. These meetings were very helpful in developing the basic UI layout for the first prototype and served as the basis for design on the UI. Figure 11 shows an example screenshot of the graphical user interface of LARA.

4.3.5 Summary

In summary, the design phase began in late October 2009 and was completed in December 2009. The resulting design document for the first prototypes included 62 entity classes, designs for the data access object layer, 58 database tables, 16 pages of constraints on entity classes, 13 pages of constraints on database tables, and 70 pages in total. The constraints provided in the design document were a tremendous help during the implementation phase ensuring the data integrity of LARA. Due to the inexperience of the developer, this design provided little guidance in the development of the action and presentation layers described in the design. As a result, the implementation phase would be required to experiment and fill in the gap not provided by the design document. Even though these gaps delayed the implementation phase, the design for the data access and entity layer proved to be robust which was important in the development of the other layers.

4.4 Prototype II

Since the design phase of the second prototype was not reached at the time this paper was written, this paper will not discuss any design issues associated with the second prototype of LARA.

5. Implementation

Following the design phase, the development proceeded into the implementation phase. This is the phase when the software is written according to the specifications laid out in the design phase in order to accomplish the requirements laid out in the requirements phase. The implementation phase of a good design often is little more than a mechanical transformation from design to code. Even though this transformation can be quite rigid, there is room for innovation for the developer to create efficient, reusable software. Completing the process started in the design phase for providing implementation details to the requirements, this phase finishes the all the decisions left unspecified in the design. Depending on how detailed or generic the design is, the range of the decision required in the implementation phase can vary substantially.

With the case of LARA, the design was geared toward a generic web application and tried not to assume much about how the web application would be implemented. This meant that the design did not make assumption about the web technologies that would be used or what web application frameworks would be leveraged. As a result, a lot of room was left for the developer to experiment and find what worked during the implementation phase.

5.1 Prototype I

The first prototype's implementation phase was the most uncertain phase in the project. This uncertainty in the phase provided many problems for the developer. Due to the developer's inexperience with web development, this phase was both an implementation phase and a crash-course in web development.

Although the developer attempted to learn about web development before the implementation phase began, the volume of knowledge required to correctly develop a web application exceed the time available before the implementation needed to begin. To make the matter even more difficult, the developer still only had book knowledge of the web technologies and had yet to really use them in a non-trivial manner. The following is a list of technologies that the developer learned or partially learned in the implementation phase of the first prototype: XHTML, CSS, JavaScript, jQuery, JSP, JSF, Facelets, RichFaces, JPA, and Seam.

5.1.1 Technologies and Frameworks

When developing any software, the proper selection of technologies and frameworks can accelerate the development of a project significantly if chosen wisely. Most of the time, the developer does not have a choice and the use of a specific framework is mandatory and the use of additional framework is prohibited without proper approval. With LARA, the opposite was true. Since the local CCC had no previous web application development, the choice was given to the developer to choose any framework that the developer thought would provide the best approach. This is an unusual type of development where the developer was given such as wide choice in the technologies to be used even the power to chose any development language.

In order to increase the chances that LARA would be picked up by the national or regional CCC office, only Java, PHP, and Ruby were deemed acceptable development languages since these were the web languages supported at the national office level. Choosing between these three languages was a difficult decision that was carefully weighted by the developer. There were many factors that were considered when choosing the development language including the age of the development language, the portability of the resulting web application, the cost of hosting the resulting web application, the image processing support available, the frameworks available, the

developer's experience in the development language, the developer's preference and the national office's preference.

Ultimately, Java was chosen as the development language over PHP and Ruby because of the developer's experience in using Java especially with ORM technologies like Hibernate and the availability of individuals willing to act as consultants for a Java technology that would be used in development. The developer thought that this would be the best way to reduce the time required for development since the database backend could be developed concurrently while learning the web development aspects of Java. The downsides of Java were the cost for deployment and the lack of portability to move the web application since many of the cheap web hosts do not support Java web technologies. PHP was the second choice and would have been used if the client requested the developer to use after being presented with the cost implications of Java which were presented in a 7-page, comparative cost-analysis between Java, PHP and Ruby. PHP's main advantages were its portability and availability with nearly all of the cheap web hosts. Although the price was right for PHP, the developer felt that the increased learning curve with learning PHP would be too great and would delay the project's completion. Although Ruby was the top choice from the national office, it was not chosen because of how new the language was and the developer worried that there would not be the proper online resources for the developer to get the help needed to complete the project since the national office was not able to give that type of support to the developer. For these reasons, Java was chosen as the development language.

Once Java was chosen, the developer researched the ways Java provided support for dynamically generated web pages and found that the choices were servlets, JavaServer Pages (JSP) and JavaServer Faces (JSF). Having no restrictions on using newer technologies and since these technologies were built on top of each other, the developer chose to use the most recent but stable technology which was JSF 1.2. During development, JSF 2.0 was released but the developer thought that it was too risky to use since it was still so young still. JSF provided a model-view-controller architecture with

event handling which seemed to be a significant improvement over the standard JSP which is comparable in sophistication to PHP.

After the developer wrote a project unrelated to LARA using JSF implemented using JSP as a view handler, it became clear that without additional training in the uses of JSF or JSP the limited knowledge of the developer in these technologies would not be sufficient to build a sophisticated web application that handles as much data as LARA was required to handle. Thus, the developer began searching for additional web frameworks that would enhance JSF and increase the developer's ability to rapidly develop the web application without being bogged down with the low-level implementation details of a web application. From this realization, many web frameworks were investigated including Spring, Struts, Tapestry, IceFaces and MyFaces before the following web frameworks were decided upon: Seam, RichFaces, and Facelets.

Seam was chosen as a the main web application framework over Spring or Struts because it provided excellent support for persistence management with Hibernate, for conversations as an intermediate context between a request and a session, and had a rapid development tool called seam-gen that would set up a base CRUD application reverse-engineered from entity classes or a database schema. The developer believed that this would provide a good start for the web application and would reduce the development time required to write LARA. In hindsight, upgrading JSF with a web application framework like Seam, Spring, or Struts was the right choice but it is still unclear if Seam was a better choice than Spring since Spring seems to be better supported and is a lighter-weight framework than Seam. The difficulty in setting up Seam on various servers and the resources required to run a Seam application are much higher than the web developer originally estimated and may cause maintenance and deployment issues for the client.

RichFaces is a component library for JSF that provides AJAX enabled components for UI development. This allows for a Rich Internet Application experience without requiring an extensive knowledge of AJAX for the developer. RichFaces hides much of the low-level AJAX JavaScript allowing the developer to just use the components

provided and benefit from the capabilities of AJAX. IceFaces and MyFaces are similar component libraries for JSF that were considered but were not chosen because RichFaces seemed to have better integration with Seam.

Although JSF by default uses JSP as its view handler, Facelets is another technology that can feed into JSF and also adds features to JSF that JSP does not. Some features that Facelets embraces are template and composition features that allows for the easy reuse of components throughout a web application. Being implemented using XML, Facelets is also much easier to validate over JSP. Facelets also was developed with a more advanced naming container feature as a result of many of the common issues naming issues that JSP is plagued with especially within tables. To illustrate that Facelets is the future of JSF and will eventually become the JSF standard, the JSF 2.0 specification was heavily influenced by the features provided by Facelets.

Beyond the web development frameworks used, LARA also used some frameworks to manage database interactions. Bootstrapped through Seam, LARA used the Java Persistence API (JPA) and the Java Transaction API (JTA) to aid in persisting of objects in the entity layer to the database. Hibernate was used as the implementation of JPA, which is only a specification provided by Java. Hibernate was an easy chose for a JPA implementation based on the extensive experience the developer had using Hibernate and NHibernate, the .NET version of Hibernate. In addition, Hibernate is the default choice and most supported JPA implementation used by Seam. Seam adds additional features to JPA to provide easy access to the Hibernate features that supersede the JPA specification. The most notable of these features are manual flushing and Hibernate Validator. Manual flushing provides the developer with better control over when database calls are persisted to the database which it a major feature absent from the current JPA specification. Hibernate Validator is an extension of Hibernate that provides annotation based validators for POJO entities that validate the data in an entity before it is persisted to the database. In conjunction with the bean validation features within Seam and RichFaces, basic data validation at the view level on the server becomes almost trivial.

5.1.2 Template Techniques

Due to size of the database and the code similarity between components, the developer utilized some template techniques in order to write code efficiently and consistently. Since all of the data access objects were the same except for the searching methods, the developer wrote a small java program that read a base template for the data access object code and fill in the parameters that changed from entity to entity. This project was rapidly developed and was used to create the base classes for the data access objects, the data access object interfaces, and the data access object factory. The benefit of this method of using templates was that from a comma separated value file the program was able to produce multiple files based on the parameters provided; unfortunately, the program had limited expressive power as compared to a well developed template library such as FreeMaker, a Java template engine library. Another template technique used was the Snippet window in Eclipse that provides a very basic template engine for small pieces of code. The downside of this template engine was that all the parameters had to be entered in manually and only a single file could be manipulated at once. This feature was important in the development of the basic pages structure that certain types of pages followed throughout the project. It promoted consistency in development and allowed for all new pages to use the best-tested and most up-to-date version of that type of page. Unfortunately, this technique did not help with upgrading existing pages that were already customized, but since all of the pages were all created similarly most of the changes could be propagated to all the other pages using Eclipse's find and replace functionalities using regular expressions.

5.1.3 Code Generation

One of Seam's main appeals in deciding a web application framework was the code generation functionality of the tool seam-gen. The purpose of seam-gen is to provide a rapid prototype for a seam-based web application. From the start, seam-gen could

generate an initial seam application that would hook up to the proper database, link in the proper libraries, and generate a base template for the application provided that the developer answered a series of command line questions to fill in some details. Another useful feature was the auto generation of pages and backing beans used to perform CRUD operations on a database table given an entity class or a database schema. This provides a tremendous amount of auto-generated code that could quickly get a project up and running. LARA was developed using this initial setup and CRUD page creation but quickly diverged from the auto-generated code. Although the initial CRUD pages mostly worked, they did not provide the flexibility in the back end to substitute in the data access layer that was already developed. As a result, these CRUD pages and corresponding backing beans were mostly abandoned for the production level code but were studied for developing some best practices in using Seam. While not much of the auto-generated code survived into the final implementation, the auto-generated code was instrumental in the developers learning and understanding of Seam. Most of the setup files provided by Seam in the initial setup survived to the final implementation with modification from changes later in development.

5.1.4 Summary

In summary, the implementation phase of the first prototype began in December 2009 and was completed in May 2010. The resulting code for the first prototype included 251 classes, 106 web pages, 58 database tables, 149 reusable Facelets components, 63 entity classes, 68 action classes, 44 data access object classes, 20 custom JSF validators, and 3 custom JSF converters. While the implementation of the first prototype took a long time and required a great deal of experimentation, the final implementation of the first prototype proved to be good code that will be heavily reused in the next incremental prototype implementation.

5.2 Prototype II

Since the implementation phase of the second prototype was not reached at the time this paper was written, this paper will not discuss any implementation issues associated with the second prototype of LARA.

6. Testing

Nearing the end of the implementation phase, the testing phase began. The testing phase is an important phase in development where the software bugs are discovered and worked out. There are many types of testing that can and should be performed during the testing phase including white box testing, black box testing, grey box testing, unit testing, integration testing, system testing, regression testing, acceptance testing, alpha testing, and beta testing. All these types of testing are invaluable with developing in a professional environment, but can prove difficult when developing a software application without professional tool support. Testing is a very time consuming phase that should be run by a different team than the development team in order to provide a non-bias assessment of the software. This best practice is impossible when both the development team and testing team consist of a single developer, which happened to be the case with LARA. Frequently when a project is running over-budget and over-time in real world software development project, the first phase to get cuts is the testing phase and LARA was no exception.

6.1 Prototype I

As a result of the limitation in testing described above, the testing in LARA focused on unit testing and using close-to-real-world test data in an alpha testing environment. Although ideally the testing phase would have lasted months, the developer ran out of time on the deadline for the project completion and thus the testing phase was cut during the first prototypes with the hopes that testing could be better completed in the second

prototype once the project was able to be developed on the final deployment server and would use some beta testers to aid in the testing effort.

6.1.1 Test Data

To aid in the testing during development in the implementation phase and to ease the transition from test data to the production data, realistic test data was important to both the developer and the client. It was the goal of the developer to use test data that accurately represented the La Crosse Area Ministry in as many ways as possible without exposing private or personal information in doing so. For this reason, all the test data involving personal information was faked with dummy information and only used real data if that information was already readily available to the general public via the laxcru.org website. Thus, the real staff member information was used in the test data but fake student data was created for the protection of those students' information before proper security measures were put in place. All of the other information including the internal structure of the area ministry, the actual application templates used in current CCC application and the enumerated values for type information were created over the span of four meetings and numerous emails between the developer and various CCC staff members. At present time, the test data consists of 2970 database records.

In order to develop test data, many customized Excel spreadsheets were used to create fake data for personal information and record data from the CCC organization about the data they desired to be in the production database. This test data was constructed based on the developer's previous knowledge of the organization and with the help of two of the CCC staff members, Mark Brockberg and David Hafner. Since no digital records had previously been stored for this information, nearly all the test data was manually entered by either the developer or by the CCC staff using forms provided to them by the developer. As the test data was being gathered, the developer created specialized Excel spreadsheet to transform the test data found in the spreadsheets into a .sql file with executable insert statements for a MySQL database. The test data was developed this

way to help the developer manage the test data and provide easy methods for the database to be reset back to the default test data.

6.1.2 Use Case Testing

Although no formal testing documents were created, the developer performed tests of many use cases developed in the requirements phase to develop a confidence in the quality of the code implemented. A checklist was created from the use case diagrams in the requirements document to aid in the testing of the first prototype. With this list, the developer tracked the progress of each use case from implementation to test completion. The following is a list of all the stages that were tracked with this list: completions of implementation, addition of all validation checks, test with valid data, and ad hoc testing with invalid data. From the list of 268 requirements, 243 were implemented, 238 were tested and passed with valid data, and 229 were tested and passed with invalid test data and an ad hoc approach.

6.2 Prototype II

Since the testing phase of the second prototype was not reached at the time this paper was written, this paper will not discuss any testing issues associated with the second prototype of LARA.

7. Limitations

Despite the developer's best efforts there are still limitations to LARA at the completion of this thesis. The following section explains what the current limitations of LARA are with respect to its deployment and the functionality from the requirements of the first prototype that have not been implemented.

7.1 Deployment

At present time, the sponsor does not have any location that is suitable to deploy the web application. As a result, LARA currently is not accessible outside the development environment and thus is quite limited in its usages.

7.2 Missing Functionality

The following are a list of requirements for the first prototype that were incomplete at the completion of this thesis and thus must be considered a limitation of the system.

- Support for international addresses and phone numbers in the presentation and business logic layers of the system (The data access layer and database do support this feature)
- Support advanced user account management features for the administrator such as creating, deleting, and invalidating a user account
- Support complex validation before flushing to the database so that the Hibernate session is not unrecoverable.
- Support adding and removing a campus to and from an area ministry

- Support multiple-choice questions in application templates
- Support copy features for application templates and school years
- Support CCC member's ability to manually edit another user's profile.

8. Future Work

From the broad scope of the initial requirements of the project, it is clear that there are many avenues for future work with this system. There are an infinite number of useful features that could be added to the system to aid the CCC staff in managing their ministry.

8.1 Prerelease and Deployment

Although the system currently works on a development server, there is some work that will be required for this software to be released for use. The first and most important is for the local CCC chapter to provide a suitable location for the application to be deployed. At present time, the local CCC chapter does not have the capabilities to deploy such a web application on their current web host. In addition to providing a server for deployment, the server must be configured with the proper authorization and authentication features to support SSL using HTTPS. Additionally if the server is not a JBoss server, they web application will need to be modification to comply with the standards of a different server such as Tomcat or GlassFish. Once the sponsor makes this determination, additional testing will be required to ensure that no functionality was lost with the transition between servers.

8.2 Local Level

For the usage of this system at the local level, there are many features that could be added. Many of these were iterated in the brainstormed requirements for the second prototype found in chapter three.

8.3 National Level

Even thought there is a lot of room for expansion at the local level, the real place for expansion of LARA is at the national level. Currently, LARA is designed to work with multiple area ministries and thus theoretically could be considered a national level application but in its current form it is quite limited in its use at that level. With the proper integration into the national CCC systems and database and an upgrade in the deployment server, LARA could be taken nationally and be able to impact every CCC ministry across the nation. With this level of system support, nationals would have access to an incredible amount of information and possibilities. Even if LARA is not taken national it could serve as a prototype or as an example of what a local level ministry tracking software could be. Even that amount of attention would provide LARA with plenty of future work.

9. Conclusion

Having completed the first prototype of LARA, it is clear that the software engineering process was essential for the success of this project. The incremental prototyping model was the proper choice for the life cycle as it allowed the developer the flexibility required to develop an application while learning necessary principles in web development. One aspect that should have changed with regard to the life cycle was the size of the prototypes; they should have been much smaller. The development of LARA was slowed due to the vast amount of knowledge that was required to develop even the first working prototype, a problem solvable with smaller prototypes. The one advantage to the size of the initial development was that it provided early integration of database persistence into a working web page. This allowed for the developer to find methods of integration that worked well for integrating all of the frameworks used in the project.

The rigorous defining of requirements and the development of a generic design with an emphasis on the application domain data were essential to the timely completion of LARA. Without an in-depth understanding of the requirements and a solid database and entity design, the implementation phase would have been a much more difficult to develop. Fortunately, the requirements and design provided a solid foundation with which to experiment, learn, and eventually build on with respect to the web development portion of the first prototype. Without the steady foundation that the software development process provided, the implementation would likely have been plagued with errors routing from the improper requirements or design designs which would have hindered the developer's ability to debug the web development layers in isolation.

The additional constraints section in the design document proved its worth in the implementation phase by making validation as easy as possible since the implementation

of the constraints was almost a mechanical transformation. This was important since the constraints were devised when the application domain was the clearest in the developer's mind and not when it was clouded with the concerns of web development.

The usage of web frameworks proved to be an important group of decision that enabled the project to be completed. Seam, Hibernate, Hibernate Validator, RichFaces, and Facelets aided in the rapid development of the first prototype by solving many of the issues the developer had in an unrelated software development project using JSP and JSF. Although the choices of web frameworks were good choices, the developer is unsure whether some of the choices or the usages of these frameworks were the best choice available. Seam was absolutely vital to the completion of the first prototype by providing solutions for many of the issues with JSF that made writing code long and cumbersome. Although Seam was a good choice, the developer thinks that Spring might have been a better choice since Spring is lighter-weight, better supported in the Java community, and provides many of the same features Seam provides. It is important to note that since the developer has yet to use Spring that it is still unclear which would have been the best choice.

Although many things could have been done better, there is one main decision that the developer would change, more specifically one decision that the developer should have made earlier in the development process. Although the implementation language is a decision for the late design phase or early implementation phase, the choice of Java over PHP and Ruby should have been made much earlier in the development process in order to facilitate the completion of the project within nine month time frame. Since this decision was not made until December of 2009, the developer was left with a very small window to learn JSP, JSF and the related frameworks. If the choice had been made earlier in late September or early October, the developer could have spent time learning web development when the developer was less busy since the developer often had down time in between meetings with the sponsor. This time could have been better spend developing small, experimental applications to give the developer experience with Java web development and would have allowed the developer more time to read materials and

books about various web frameworks. If the developer would have had the time to experiment with Seam and Spring during this time, the developer may have been able to make a more informed decision about which framework would have been best in this situation. Although mistakes were made in the software development process, this project was a great tool for the developer to learn from mistakes made and to improve on them in the future. LARA was an excellent example to the developer about why the software development process exists and why it will continue to exist so long as there are problems to solve and solutions to find. The software engineering process is a proven road map from problem to solution.

As a result of the software development process, the end product LARA was successfully developed. Once deployed, LARA will be a great first step for the local chapter of Campus Crusade for Christ on the University of Wisconsin - La Crosse campus in solving the logistical problems faced by the full-time staff in managing student leader and alumni information.

Bibliography

[1] Dan Allen, *Seam In Action*, 1st ed., Cynthia Kane and Liz Welch, Eds. Greenwich, CT, United States of America: Manning Publications Co., 2009.

[2] Hans Bergsten, *JavaServer Faces*, 1st ed., Mike Loukides and Colleen Gorman, Eds. Sebastopol, CA, United States of America: O'Reilly Media, Inc., 2004.

[3] Hans Bergsten, *JavaServer Pages*, 3rd ed., Brett McLaughlin and Sarah Sherman, Eds. Sebastopol, CA, United States of America: O'Reilly Media, Inc., 2004.

[4] Jason Brittain and Ian F Darwin, *Tomcat: The Definitive Guide*, 2nd ed., Simon St.Laurent and Loranah Dimant, Eds. Sebastopol, CA, United States of America: O'Reilly Media, Inc., 2008.

[5] Madhushree Ganguli, *Making Use of JSP*, 1st ed., Ben Ryan and Kathryn A. Malm, Eds. New York, NY, United States of America: Wiley Publishing, Inc., 2002.

[6] JBoss Community. (2010, April) Hibernate - JBoss Community. [Online]. http://www.hibernate.org/

[7] JBoss Community. (2010, April) RichFaces Project Page - JBoss Community. [Online]. http://www.jboss.org/richfaces

[8] JBoss Seam. (2010, April) Seam Framework - JBoss Seam. [Online]. http://seamframework.org/

[9] Eric A. Mayer, *CSS: The Definitive Guide*, 3rd ed., Tatiana Apandi and Rachel Monaghan, Eds. Sebastopol, CA, United States of America: O'Reilly Media, Inc., 2007.

[10] Tommy Olsson and Paul O'Brien, *The Ultimate CSS Reference*.

[11] Oracle. (2010, April) Java Persistence API. [Online].

http://java.sun.com/javaee/technologies/persistence.jsp

[12] Oracle. (2010, April) JavaServer Faces Technology. [Online]. http://java.sun.com/javaee/javaserverfaces/

[13] Jennifer Niederst Robbins, *Learning Web Design*, 3rd ed., Linda Laflamme and Philip Dangler, Eds. Sebastopol, CA, United States of America: O'Reilly Media, Inc., 2007.

[14] Robert W. Sebesta, *Programming the World Wide Web*, 5th ed., Michael Hirsch, Ed. Boston, MA, United States of America: Addison-Wesley, 2010.

[15] David Shea and Molly E. Holzschlag, *The Zen of CSS Design: Visual Enlightenment for the Web*, 1st ed., Cheryl England and Hilal Sala, Eds. Berkely, CA, United States of America: Peachpit Press, 2005.

[16] W3Schools. (2010, April) W3Schools.com. [Online]. http://w3schools.com/

Appendix A: Use Case Diagrams

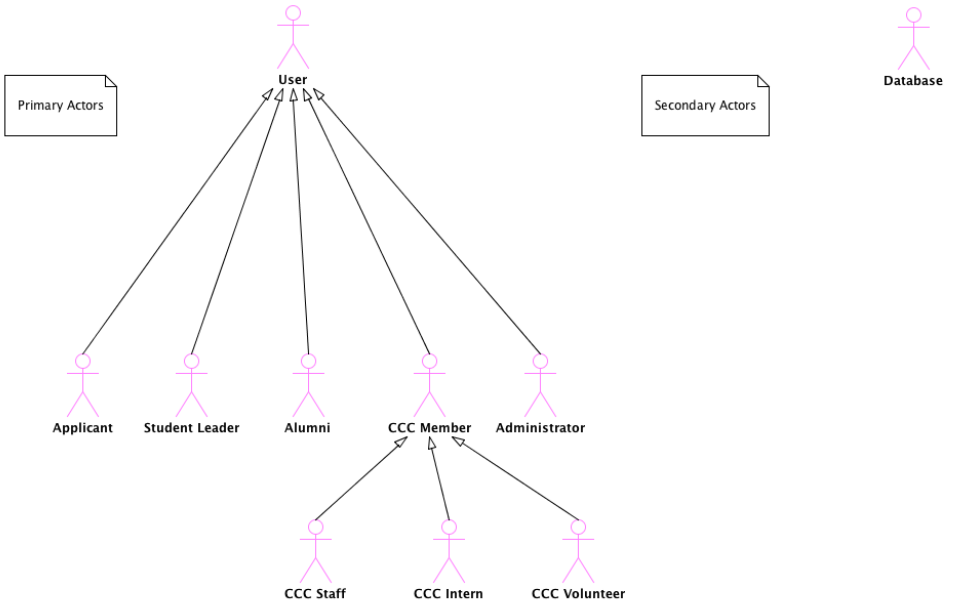


Figure 12. Use Case Diagram describing the actors of the system

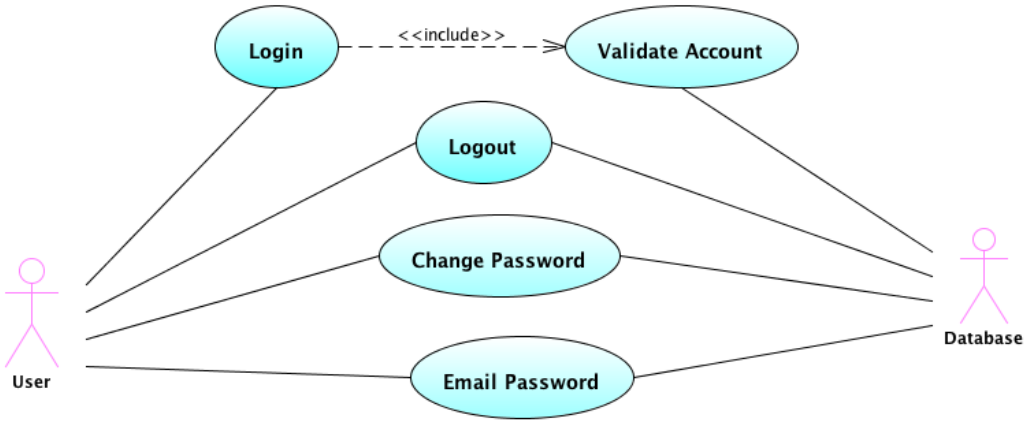


Figure 13. Use Case Diagram for User functionality

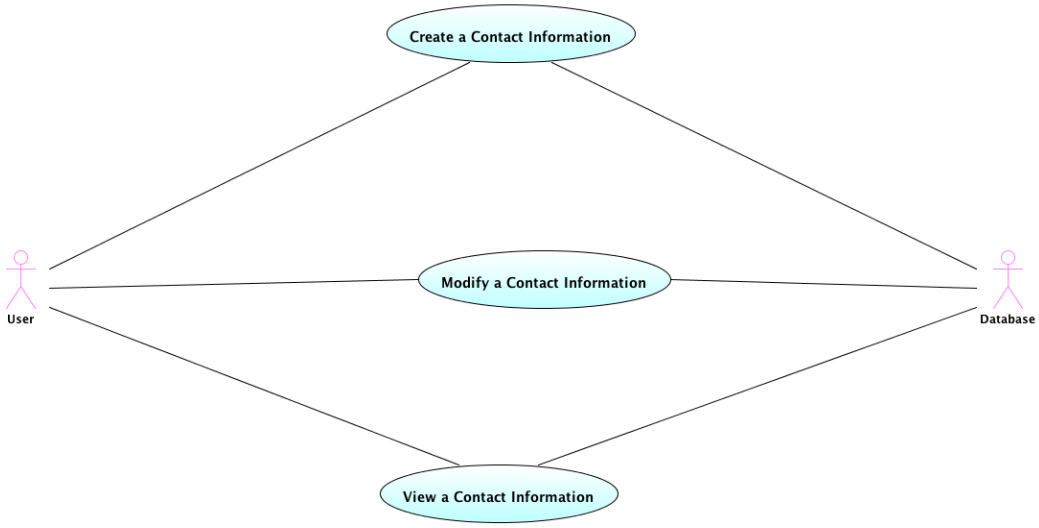


Figure 14. Use Case Diagram for User Contact Information Management

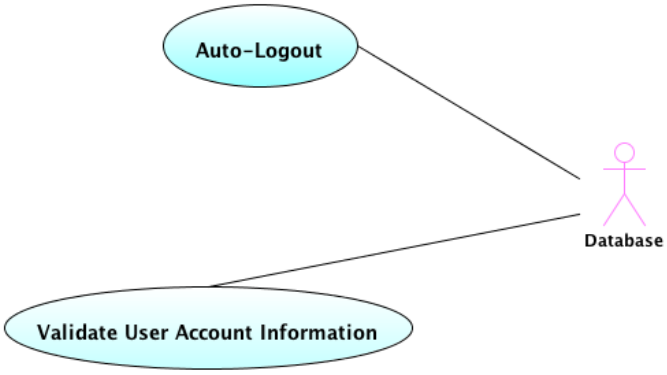


Figure 15. Use Case Diagram for System Security

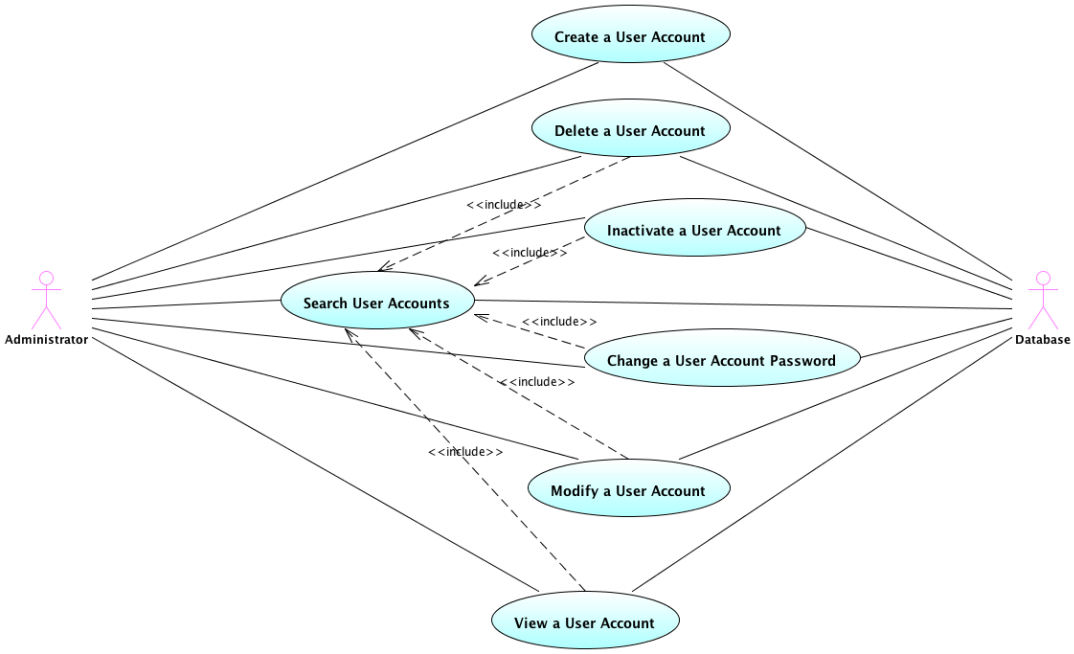


Figure 16. Use Case Diagram for Administrator User Account Management

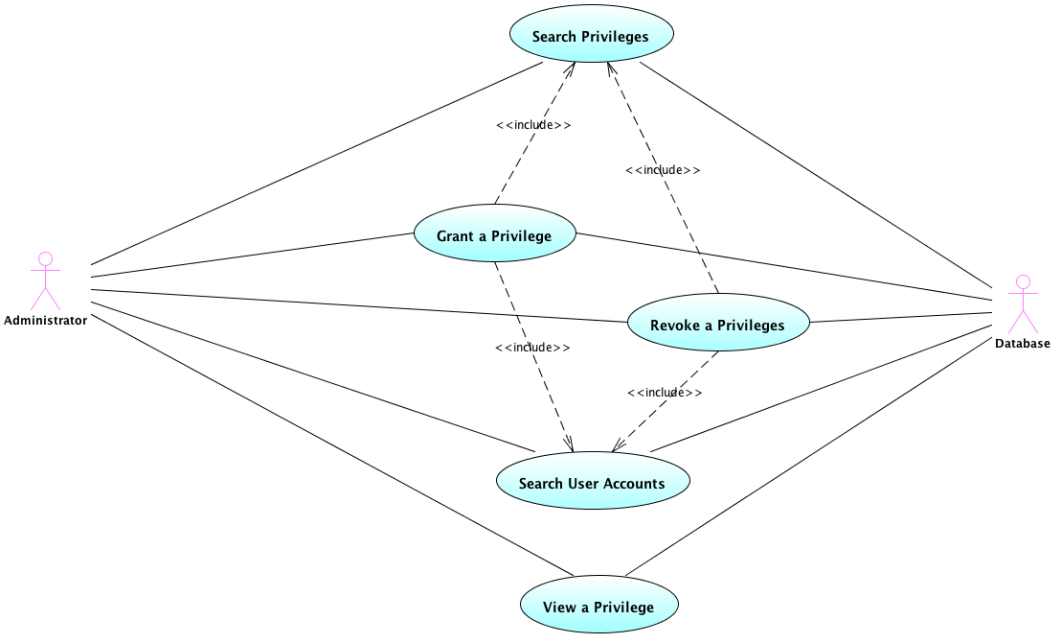


Figure 17. Use Case Diagram for Administrator Privilege Management

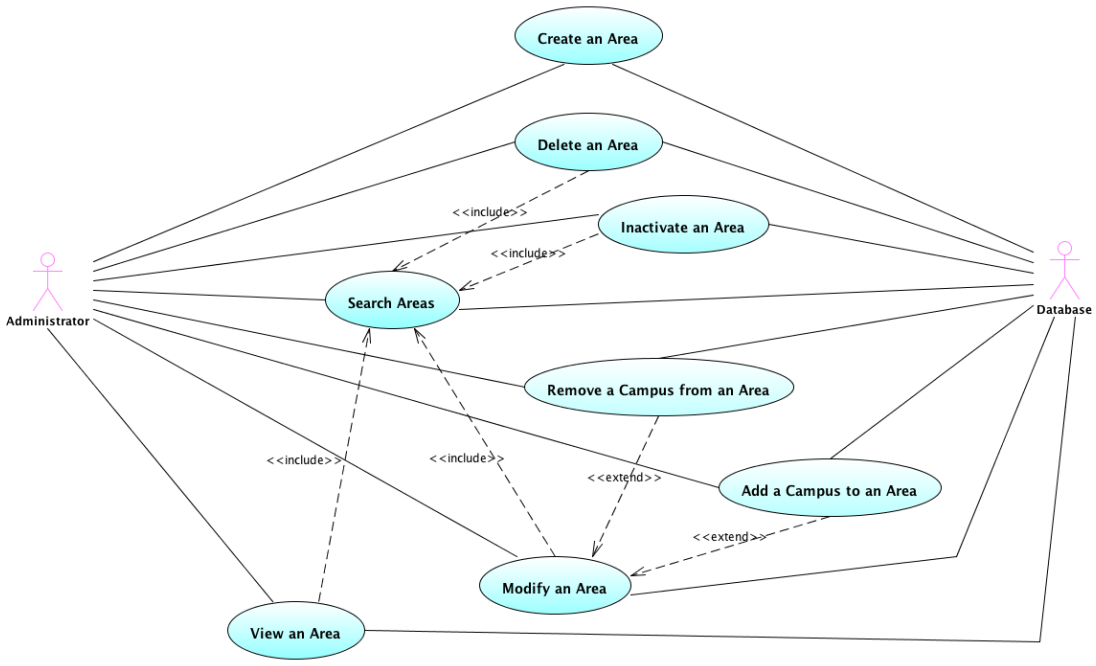


Figure 18. Use Case Diagram for Administrator Area Management

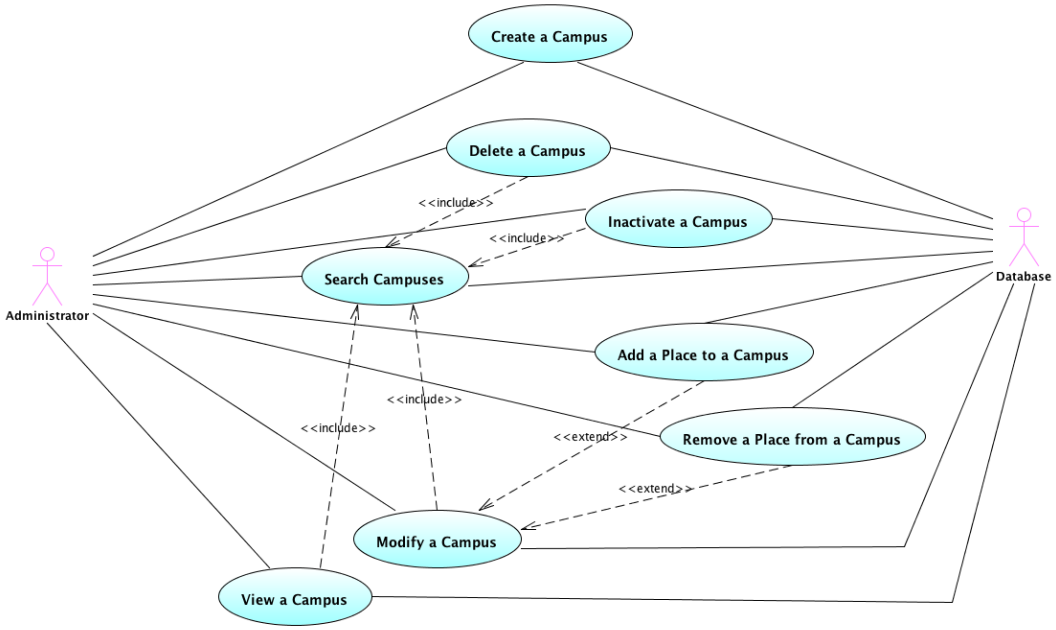


Figure 19. Use Case Diagram for Administrator Campus Management

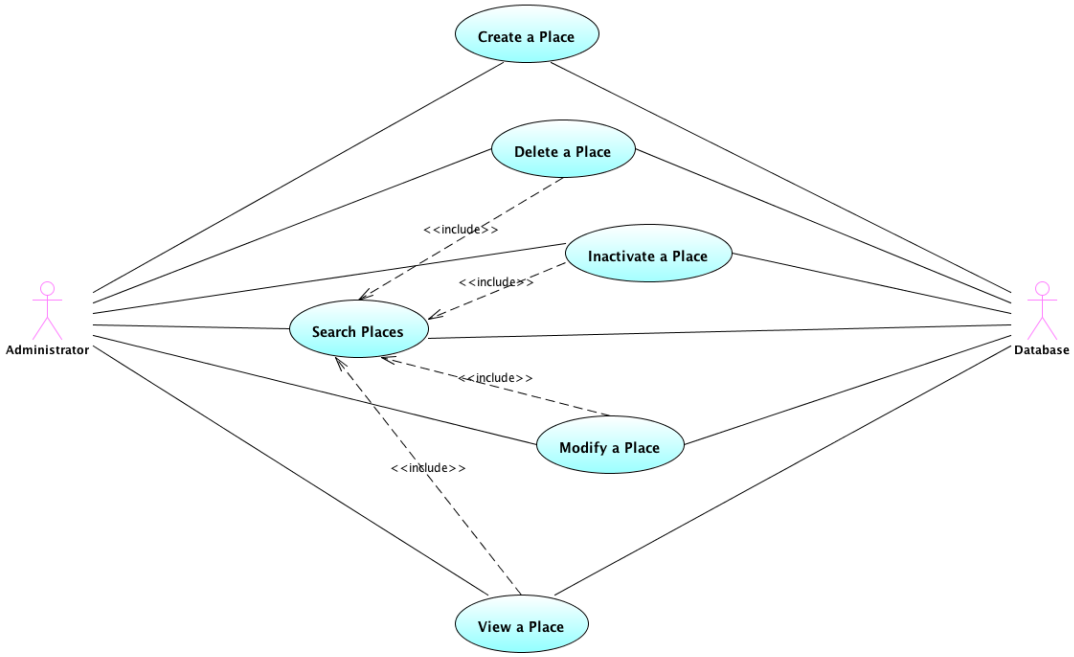


Figure 20. Use Case Diagram for Administrator Place Management

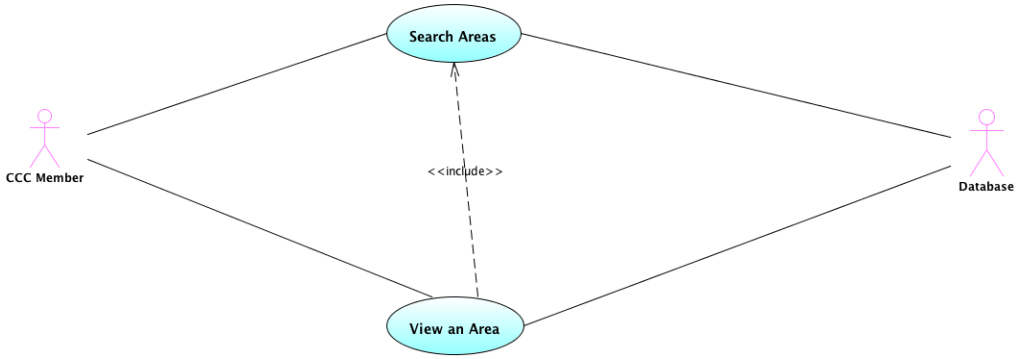


Figure 21. Use Case Diagram for CCC Member Area Management

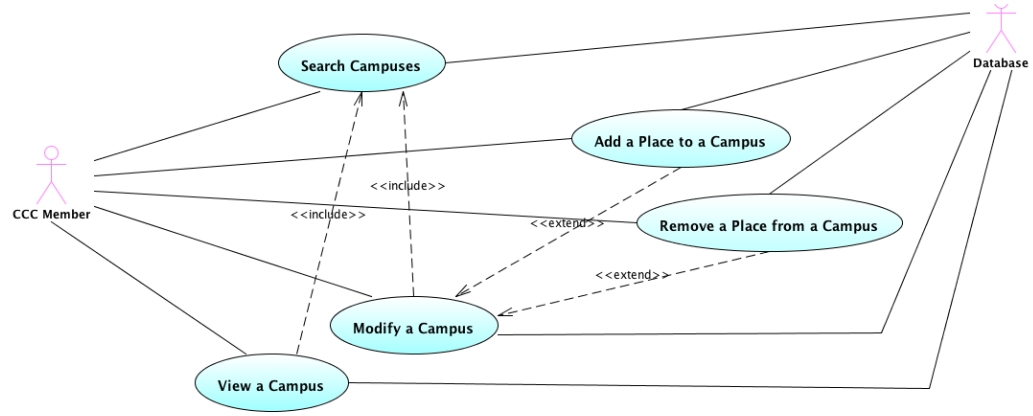


Figure 22. Use Case Diagram for CCC Member Campus Management

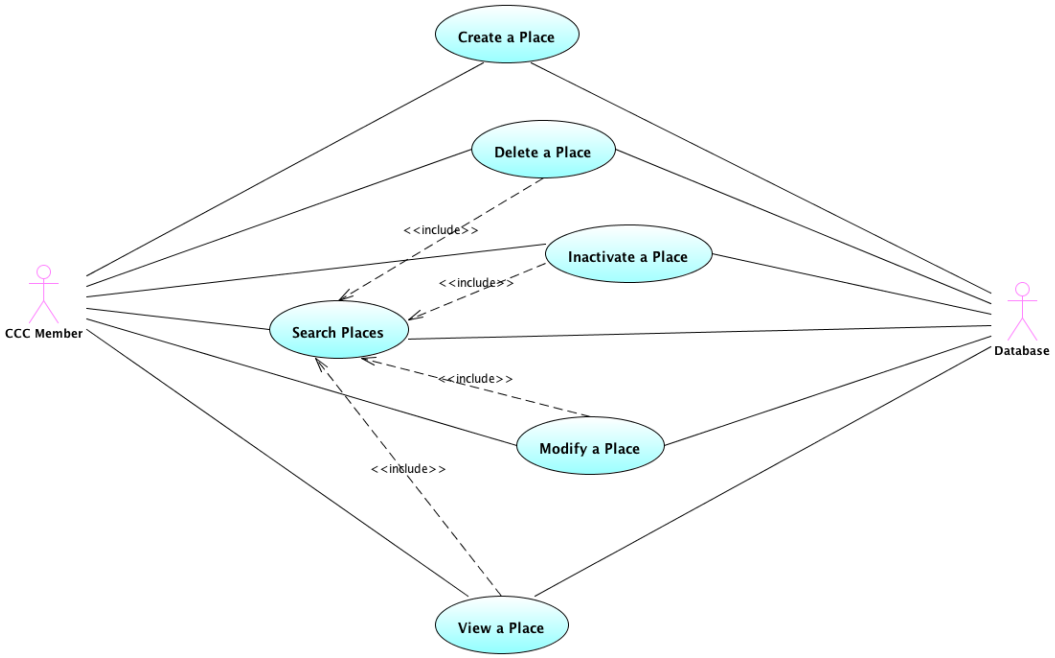


Figure 23. Use Case Diagram for CCC Member Place Management

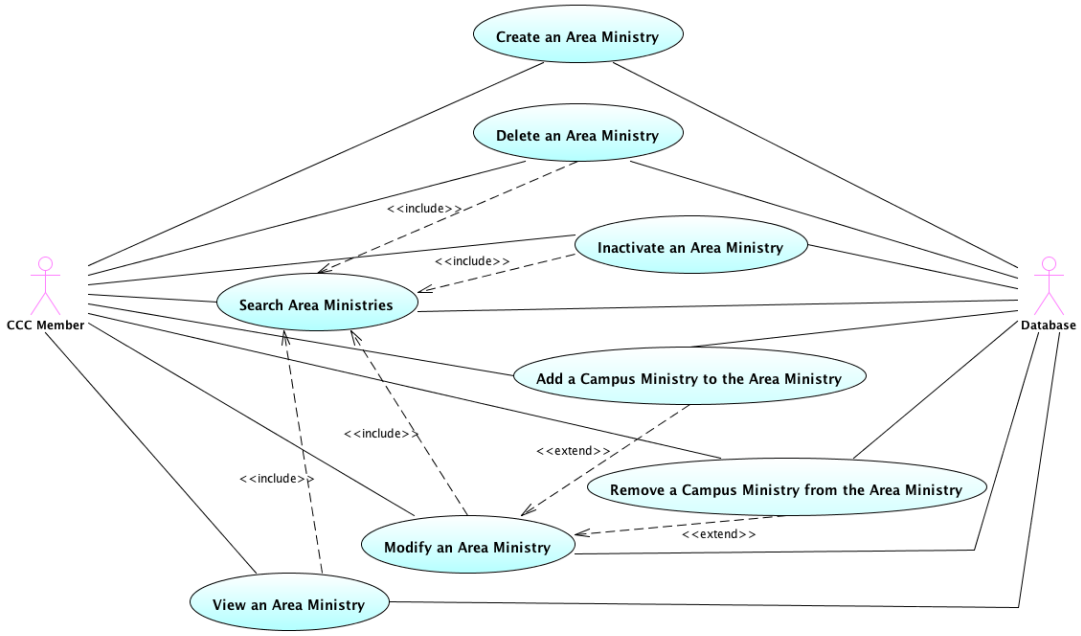


Figure 24. Use Case Diagram for CCC Member Area Ministry Management

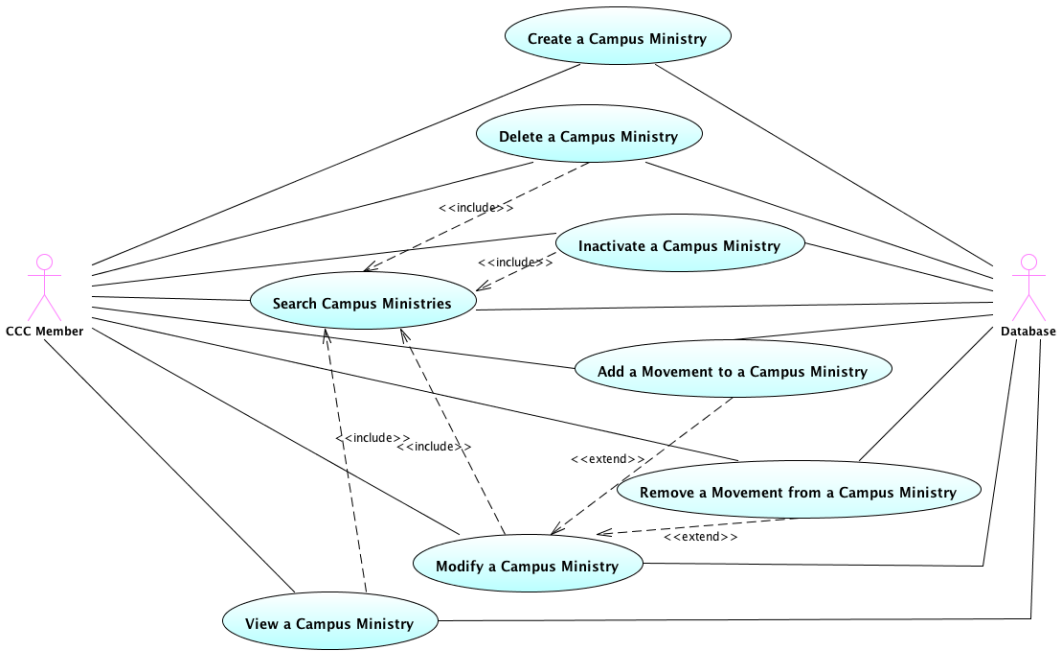


Figure 25. Use Case Diagram for CCC Member Campus Ministry Management

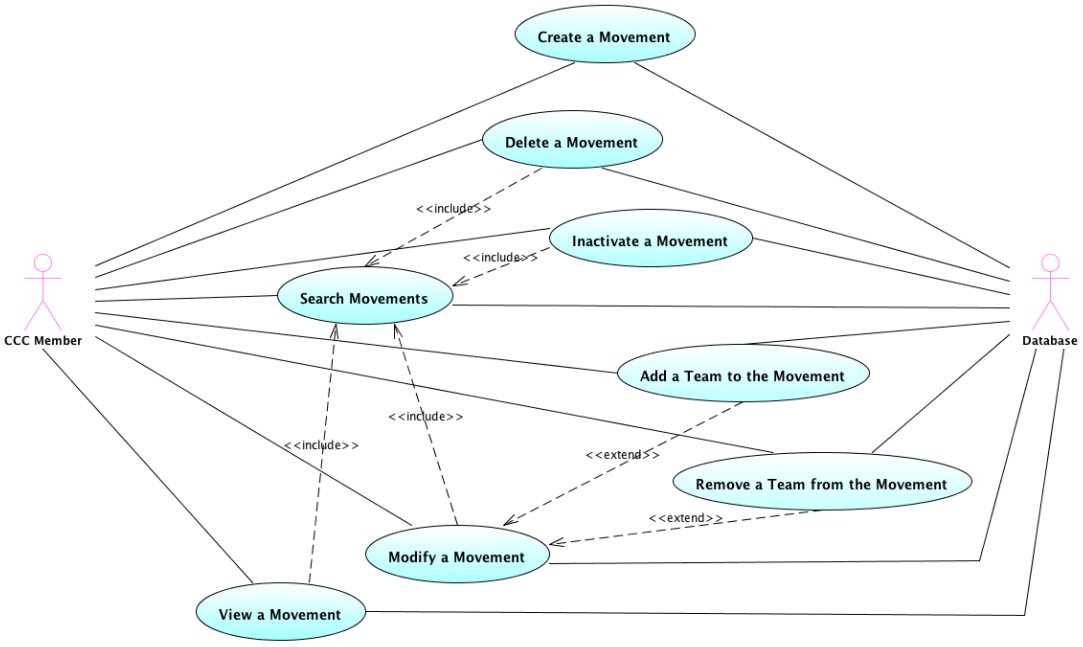


Figure 26. Use Case Diagram for CCC Member Movement Management

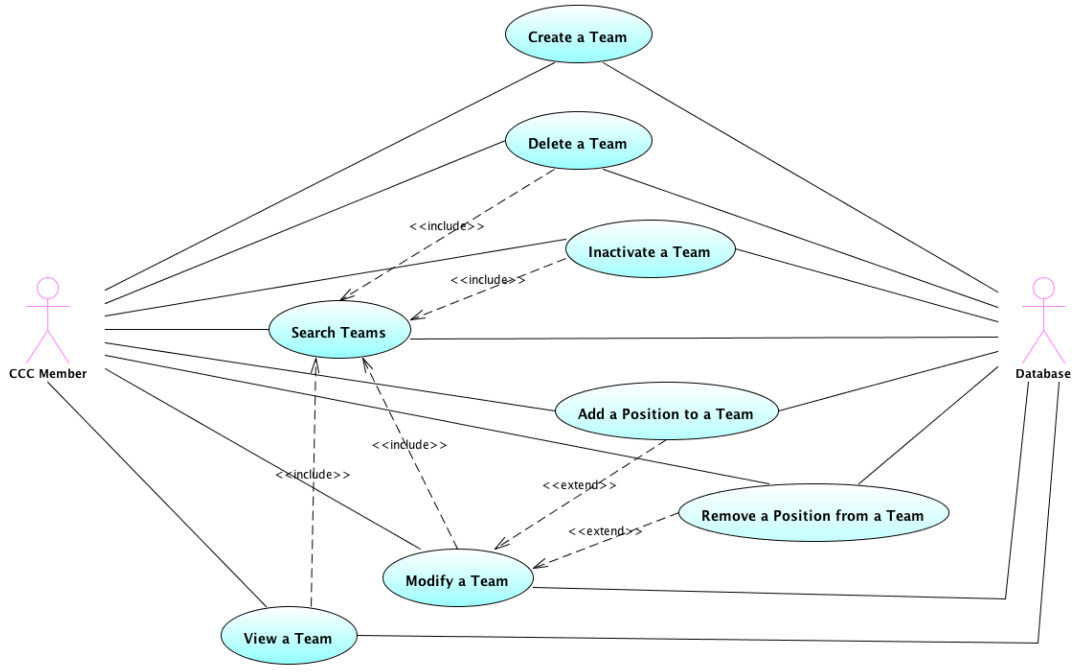


Figure 27. Use Case Diagram for CCC Member Team Management

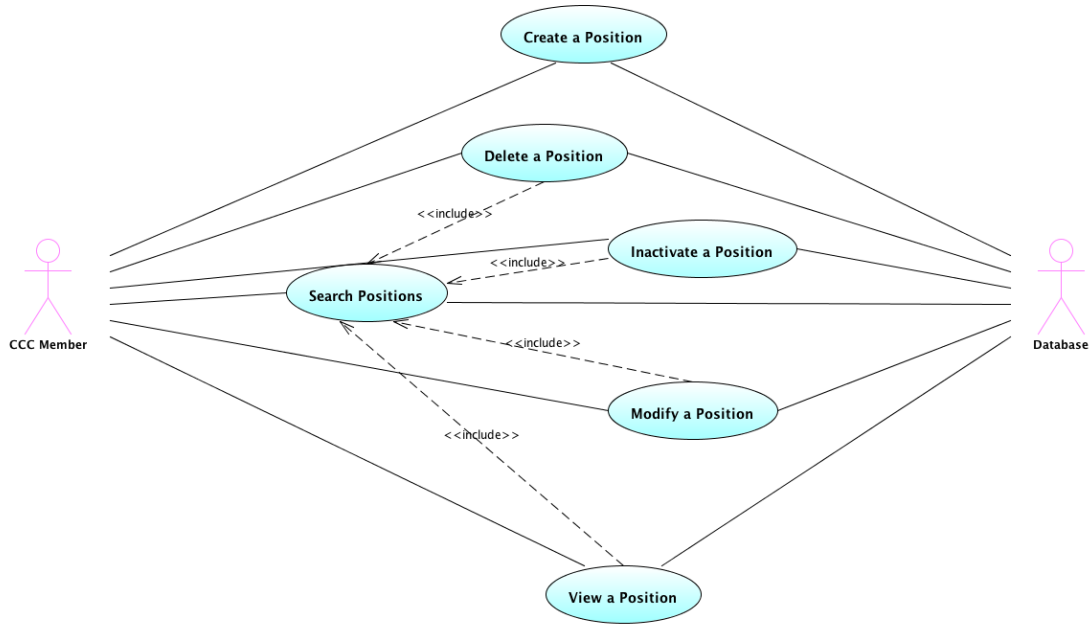


Figure 28. Use Case Diagram for CCC Member Position Management

Figure 29. Use Case Diagram for CCC Member Training Management

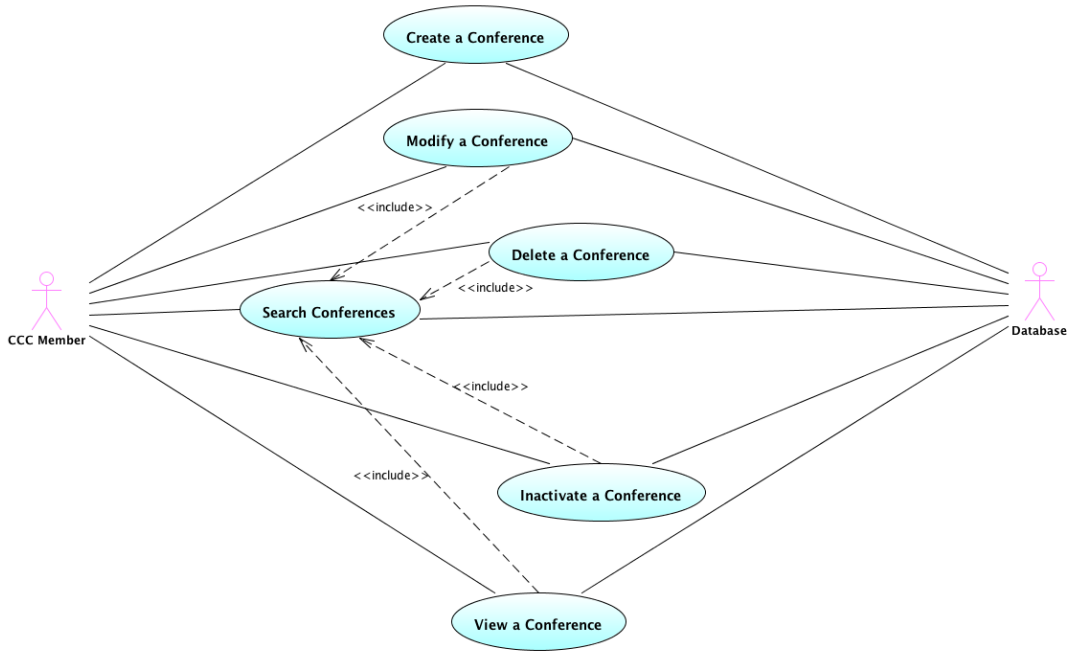


Figure 30. Use Case Diagram for CCC Member Conference Management

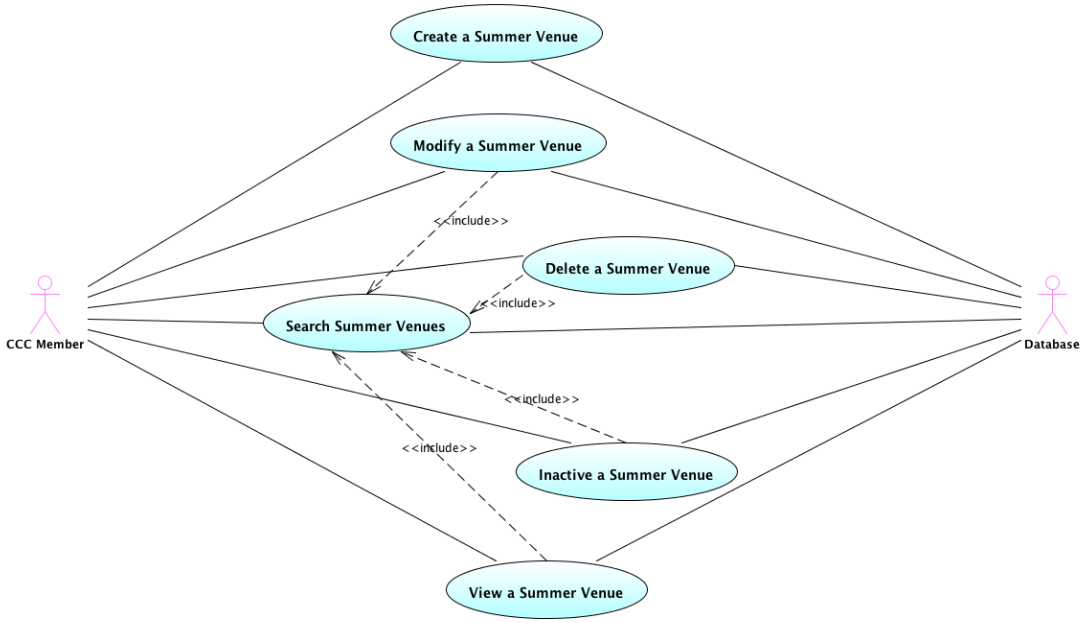


Figure 31. Use Case Diagram for CCC Member Summer Venue Management

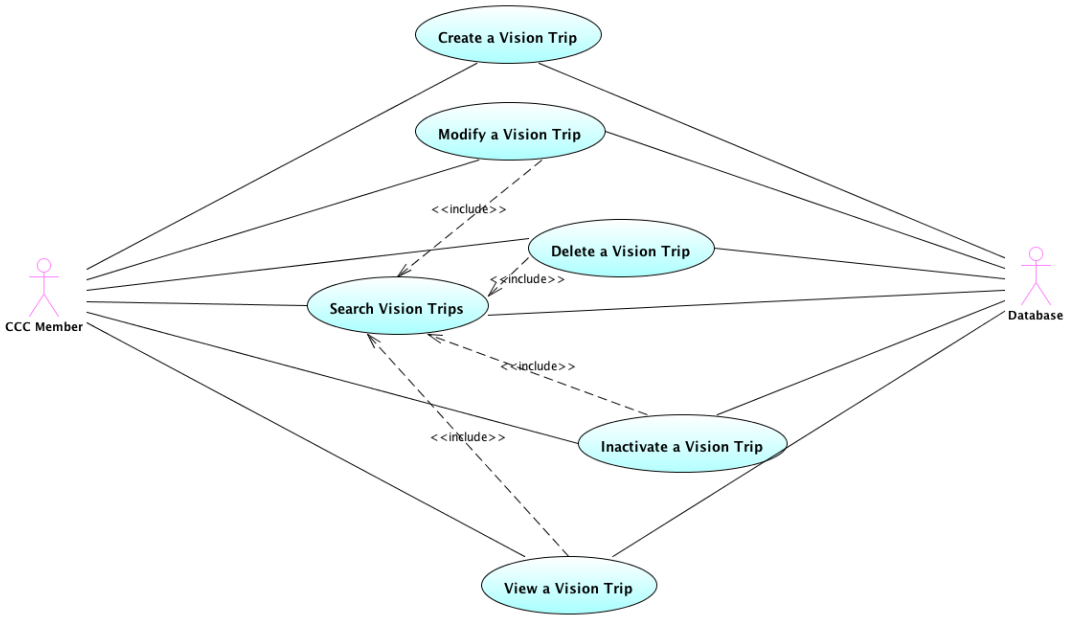


Figure 32. Use Case Diagram for CCC Member Vision Trip Management

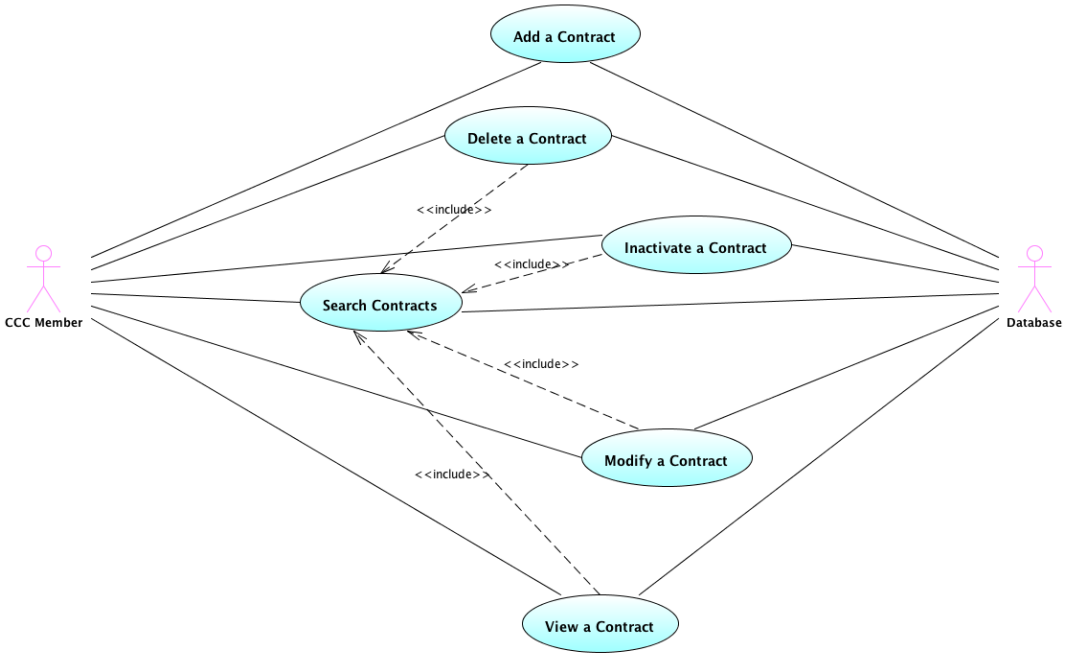


Figure 33. Use Case Diagram for CCC Member Contract Management

Figure 34. Use Case Diagram for CCC Member Application Template Management

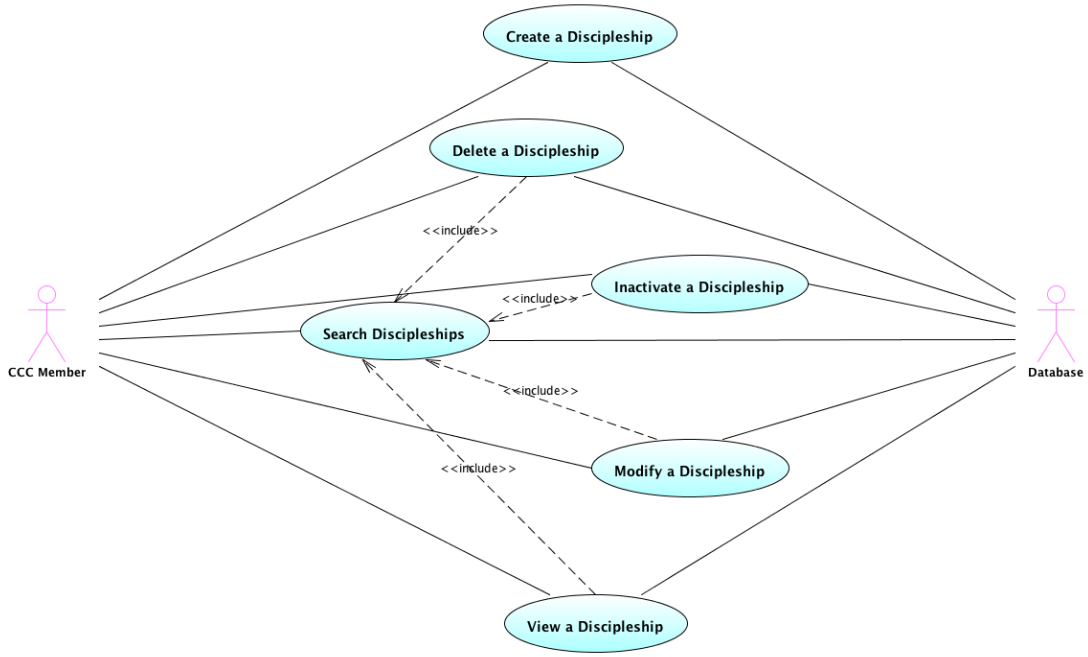


Figure 35. Use Case Diagram for CCC Member Discipleship Management

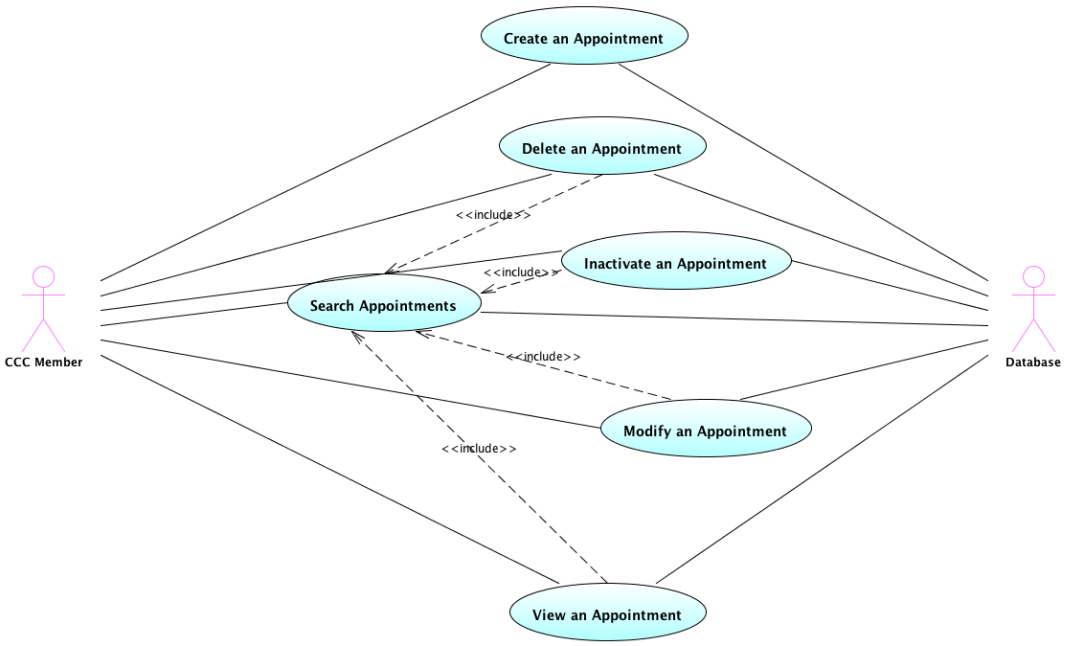


Figure 36. Use Case Diagram for CCC Member Appointment Management

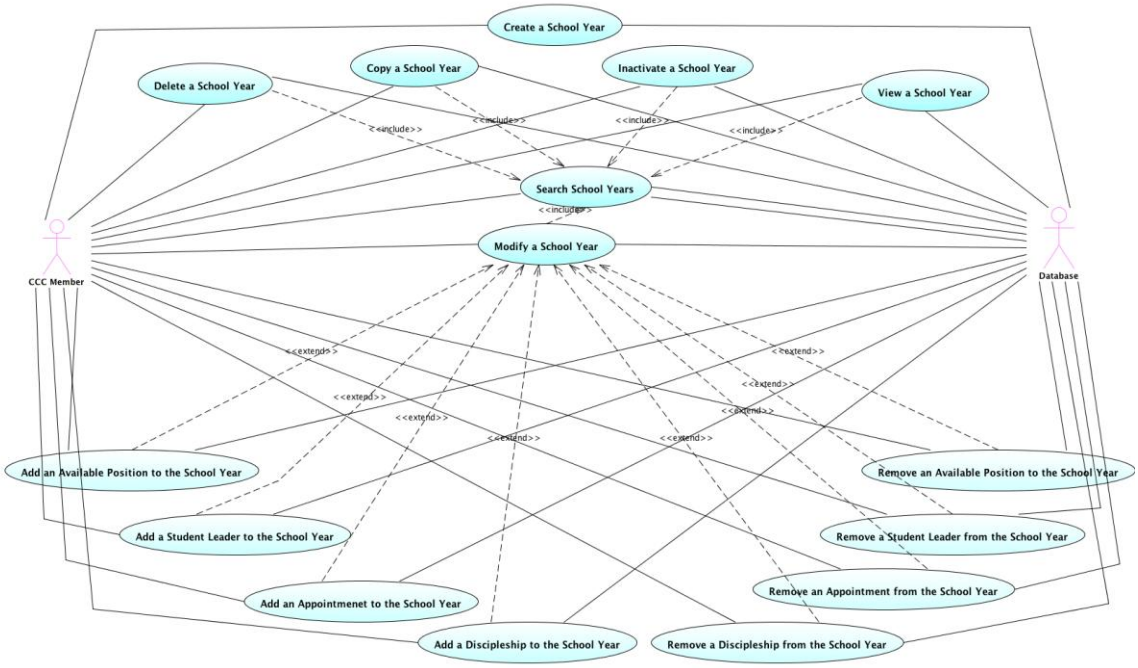


Figure 37. Use Case Diagram for CCC Member School Year Management

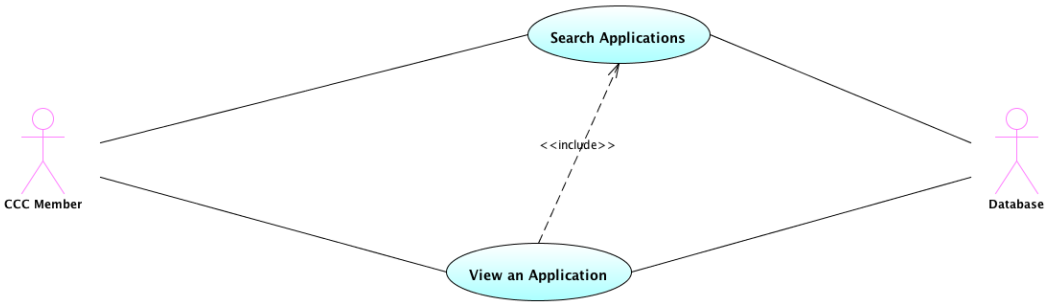


Figure 38. Use Case Diagram for CCC Member Application Management

Figure 39. Use Case Diagram for Applicant Registration

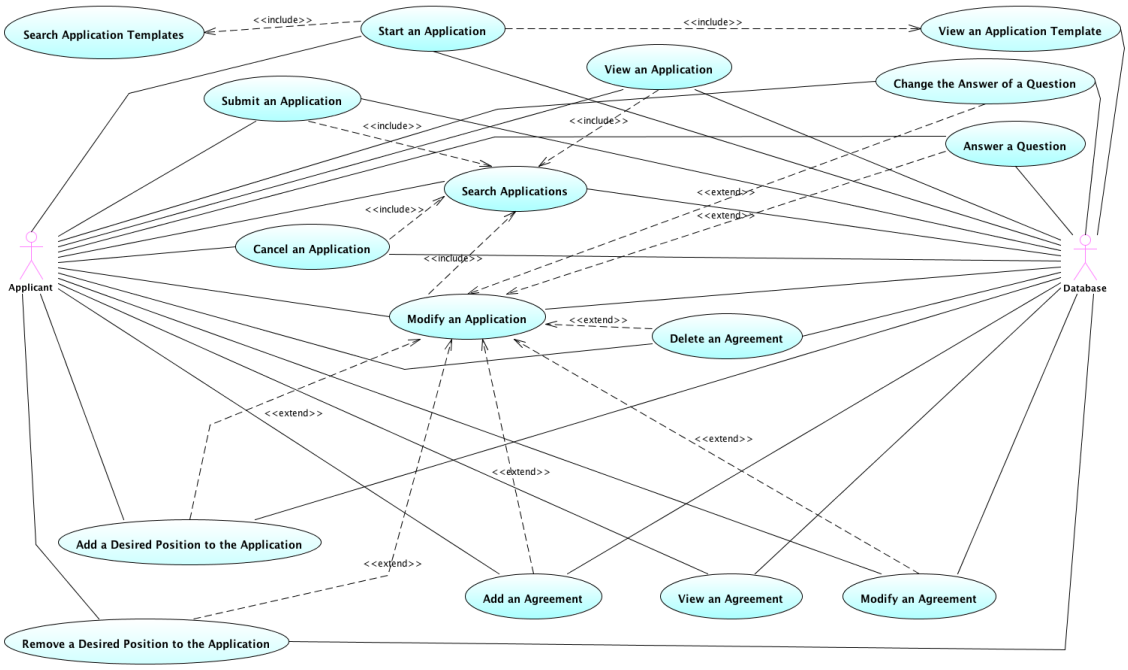


Figure 40. Use Case Diagram for Applicant Application Management

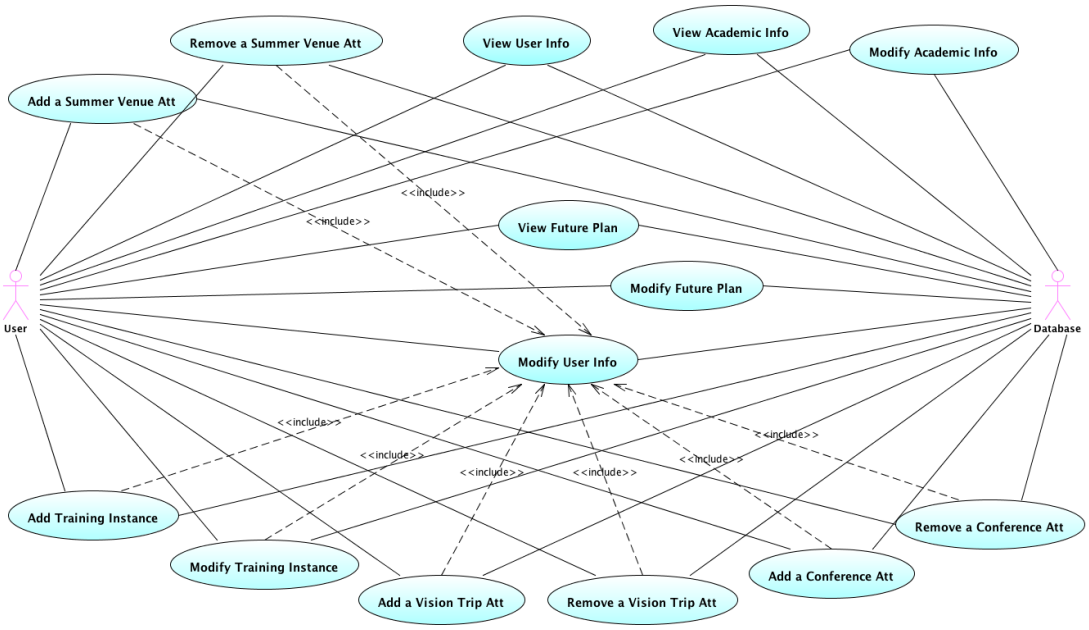


Figure 41. Use Case Diagram for User Profile Management

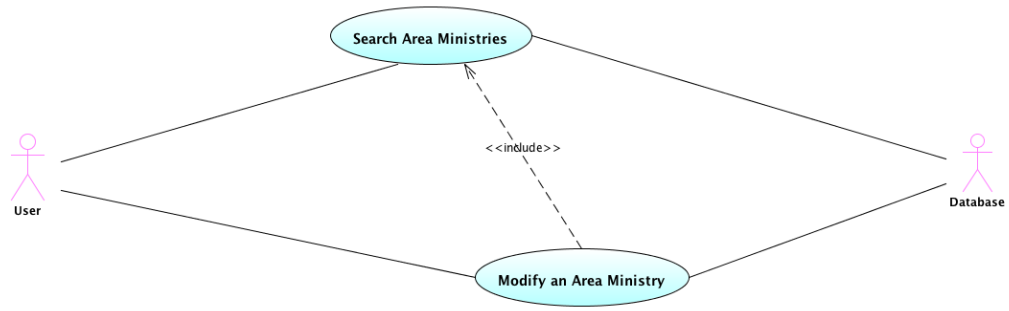


Figure 42. Use Case Diagram for User Area Ministry Management

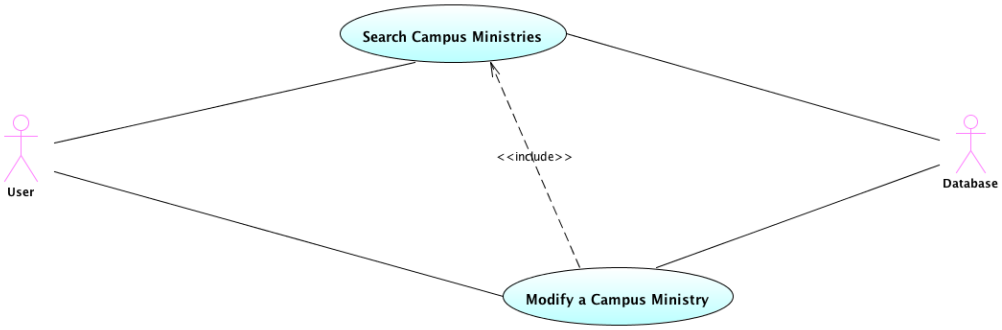


Figure 43. Use Case Diagram for User Campus Ministry Management

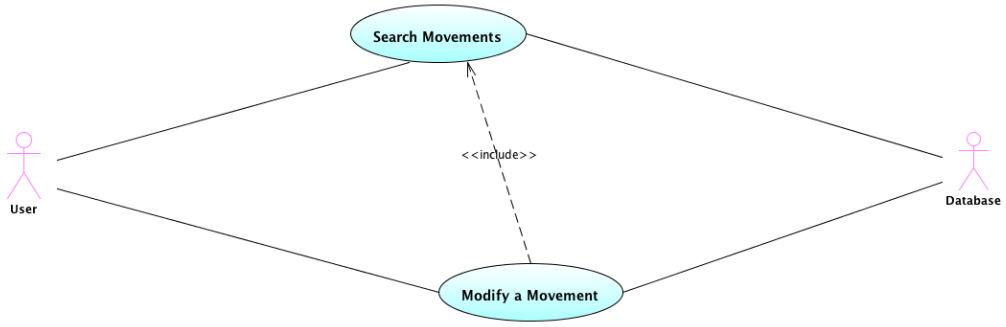


Figure 44. Use Case Diagram for User Movement Management

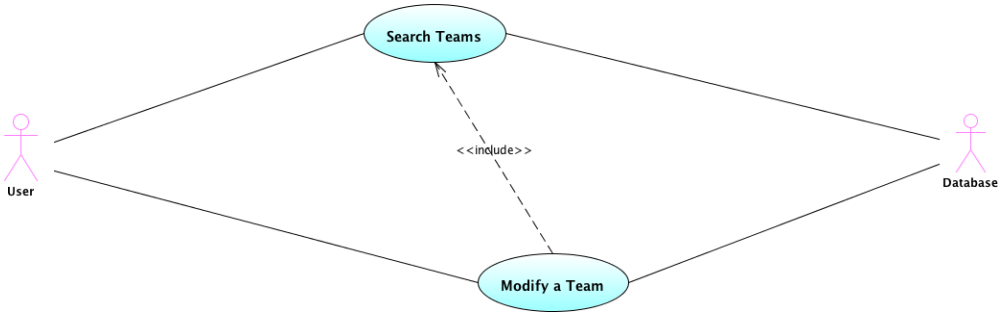


Figure 45. Use Case Diagram for User Team Management

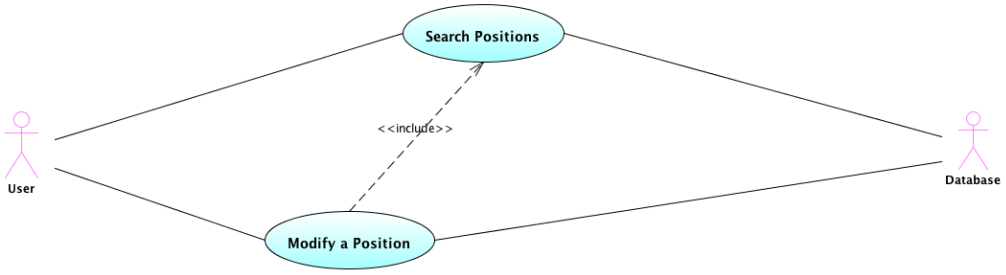


Figure 46. Use Case Diagram for User Position Management

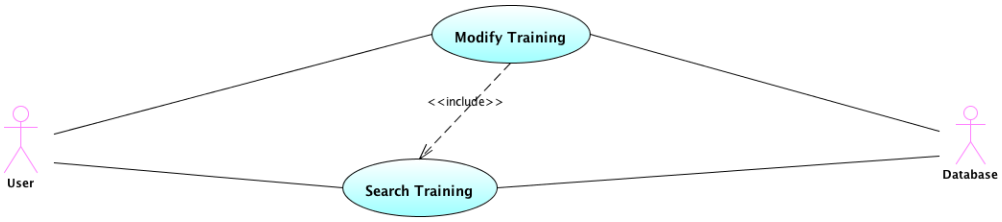


Figure 47. Use Case Diagram for User Training Management

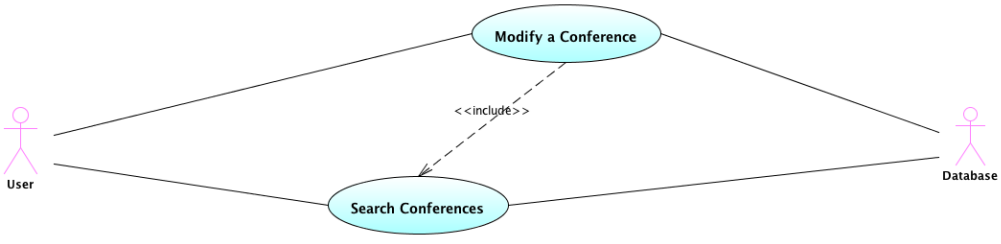


Figure 48. Use Case Diagram for User Conference Management

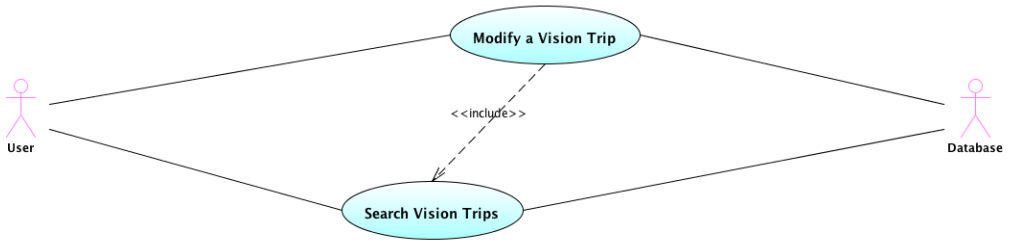


Figure 49. Use Case Diagram for User Vision Trip Management

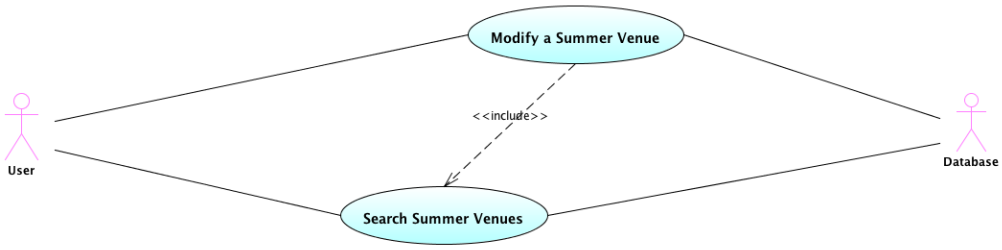


Figure 50. Use Case Diagram for User Summer Venue Management

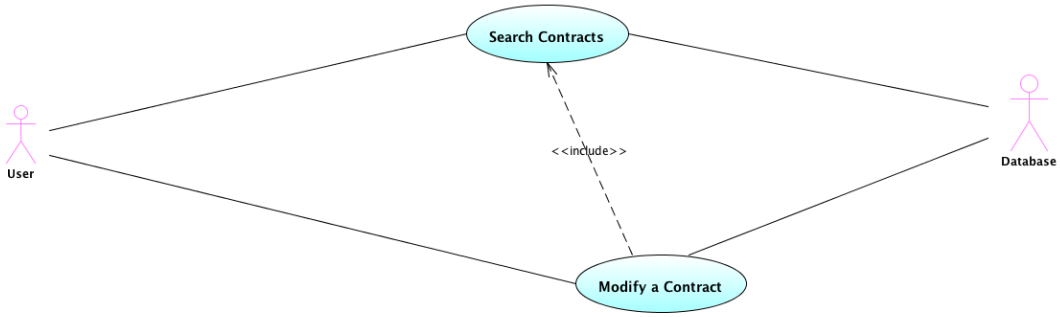


Figure 51. Use Case Diagram for User Contract Management

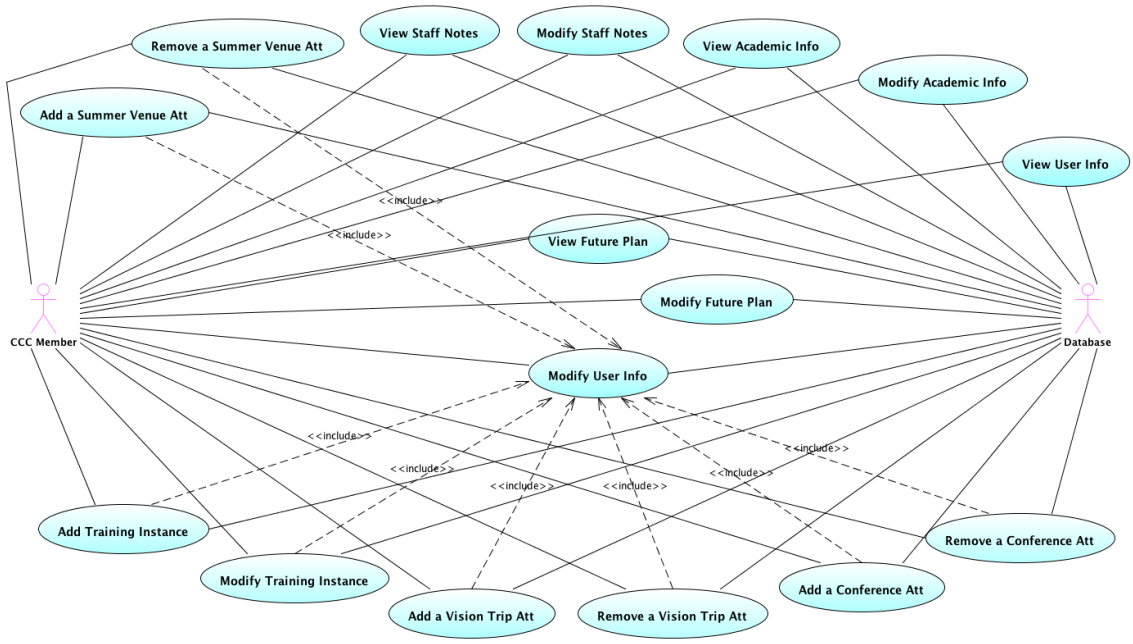


Figure 52. Use Case Diagram for CCC Member User Profile Management

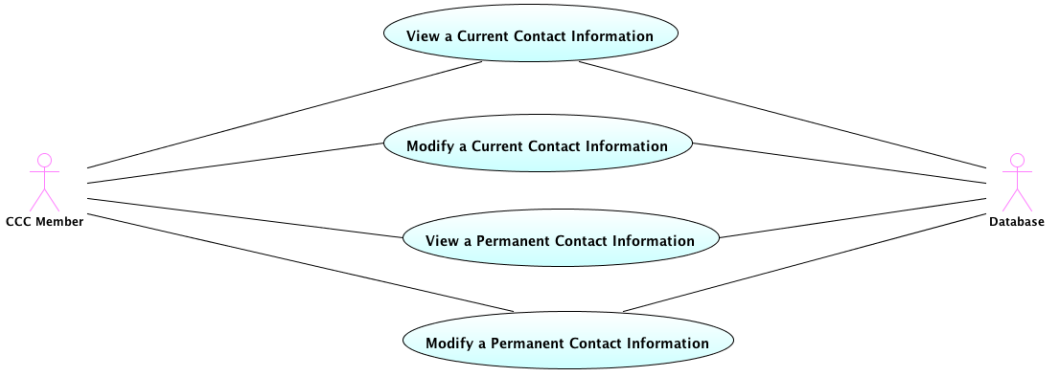


Figure 53. Use Case Diagram for CCC Member Contact Information Management

Appendix B: Use Case Point Analysis

Actor Name	Complexity	Weight Factor
User	Complex	3
CCC Member	Complex	3
CCC Staff	Complex	3
CCC Intern	Complex	3
CCC Volunteer	Complex	3
Applicant	Complex	3
Student Leader	Complex	3
Alumni	Complex	3
Database	Simple	1

Table 2. Actors in Use Case Point Analysis

Section	Use Case	Complexity	WF
Account Management	Login	Simple	5
Account Management	Logout	Simple	5
Account Management	Change Password	Simple	5
Account Management	Email Password	Simple	5
Contact Information Management	Create a Contact Information	Simple	5
Contact Information Management	Modify a Contact Information	Simple	5
Contact Information Management	View a Contact Information	Simple	5
Security	Validate User Account	Average	10
Security	Validate User Account Permissions	Complex	15
Security	Auto-logout	Simple	5
User Account Management	Create a User Account	Simple	5
User Account Management	Modify a User Account	Simple	5
User Account Management	Change a User Account Password	Simple	5
User Account Management	Delete a User Account	Simple	5
User Account Management	Inactivate a User Account	Simple	5
User Account Management	Search User Account	Simple	5
User Account Management	View a User Account	Simple	5
Privileges Management	Grant Privileges	Simple	5
Privileges Management	Revoke Privileges	Simple	5
Privileges Management	Search Privileges	Simple	5
Privileges Management	View Privileges	Simple	5
Area Management	Create an Area	Simple	5
Area Management	Modify an Area	Simple	5

Area Management	Delete an Area	Simple	5
Area Management	Inactivate an Area	Simple	5
Area Management	Search Areas	Simple	5
Area Management	View an Area	Simple	5
Area Management	Add a Campus to the Area	Simple	5
Area Management	Remove a Campus from the Area	Simple	5
Campus Management	Create a Campus	Simple	5
Campus Management	Modify a Campus	Simple	5
Campus Management	Delete a Campus	Simple	5
Campus Management	Inactivate a Campus	Simple	5
Campus Management	Search a Campuses	Simple	5
Campus Management	View a Campus	Simple	5
Campus Management	Add a Place to the Campus	Simple	5
Campus Management	Remove a Place from the Campus	Simple	5
Place Management	Create a Place	Simple	5
Place Management	Modify a Place	Simple	5
Place Management	Delete a Place	Simple	5
Place Management	Inactivate a Place	Simple	5
Place Management	Search Places	Simple	5
Place Management	View a Place	Simple	5
Area Ministry Management	Create an Area Ministry	Simple	5
Area Ministry Management	Modify an Area Ministry	Simple	5
Area Ministry Management	Delete an Area Ministry	Simple	5
Area Ministry Management	Inactivate an Area Ministry	Simple	5
Area Ministry Management	Search Area Ministries	Simple	5
Area Ministry Management	View an Area Ministry	Simple	5
Area Ministry Management	Add a Campus Ministry to the Area Ministry	Simple	5
Area Ministry Management	Remove a Campus Ministry from the Area Ministry	Simple	5
Campus Ministry Management	Create a Campus Ministry	Simple	5
Campus Ministry Management	Modify a Campus Ministry	Simple	5
Campus Ministry Management	Delete a Campus Ministry	Simple	5
Campus Ministry Management	Inactivate a Campus Ministry	Simple	5
Campus Ministry Management	Search Campus Ministries	Simple	5
Campus Ministry Management	View a Campus Ministry	Simple	5
Campus Ministry Management	Add a Movement to the Campus Ministry	Simple	5
Campus Ministry Management	Remove a Movement from the Campus Ministry	Simple	5
Movement Management	Create a Movement	Simple	5
Movement Management	Modify a Movement	Simple	5

Movement Management	Delete a Movement	Simple	5
Movement Management	Inactivate a Movement	Simple	5
Movement Management	Search Movements	Simple	5
Movement Management	View a Movement	Simple	5
Movement Management	Add a Team to the Movement	Simple	5
Movement Management	Remove a Team from the Movement	Simple	5
Team Management	Create a Team	Simple	5
Team Management	Modify a Team	Simple	5
Team Management	Delete a Team	Simple	5
Team Management	Inactivate a Team	Simple	5
Team Management	Search Team	Simple	5
Team Management	View a Team	Simple	5
Team Management	Add a Position to the Team	Simple	5
Team Management	Remove a Position from the Team	Simple	5
Position Management	Create a Position	Simple	5
Position Management	Modify a Position	Simple	5
Position Management	Delete a Position	Simple	5
Position Management	Inactivate a Position	Simple	5
Position Management	Search Position	Simple	5
Position Management	View a Position	Simple	5
Training Management	Add Training	Simple	5
Training Management	Modify Training	Simple	5
Training Management	Delete Training	Simple	5
Training Management	Inactivate Training	Simple	5
Training Management	Search Training	Simple	5
Training Management	View Training	Simple	5
Conference Management	Create a Conference	Simple	5
Conference Management	Modify a Conference	Simple	5
Conference Management	Delete a Conference	Simple	5
Conference Management	Inactivate a Conference	Simple	5
Conference Management	Search a Conference	Simple	5
Conference Management	View a Conference	Simple	5
Summer Venue Management	Create a Summer Venue	Simple	5
Summer Venue Management	Modify a Summer Venue	Simple	5
Summer Venue Management	Delete a Summer Venue	Simple	5
Summer Venue Management	Inactivate a Summer Venue	Simple	5
Summer Venue Management	Search a Summer Venue	Simple	5
Summer Venue Management	View a Summer Venue	Simple	5
Vision Trip Management	Create a Vision Trip	Simple	5

Vision Trip Management	Modify a Vision Trip	Simple	5
Vision Trip Management	Delete a Vision Trip	Simple	5
Vision Trip Management	Inactivate a Vision Trip	Simple	5
Vision Trip Management	Search a Vision Trip	Simple	5
Vision Trip Management	View a Vision Trip	Simple	5
Contract Management	Create a contract	Simple	5
Contract Management	Modify a contract	Simple	5
Contract Management	Delete a contract	Simple	5
Contract Management	Inactivate a contract	Simple	5
Contract Management	View a contract	Simple	5
Contract Management	Search a contract	Simple	5
Application Template Management	Create an Application Template	Simple	5
Application Template Management	Modify an Application Template	Simple	5
Application Template Management	Add a Question Section	Simple	5
Application Template Management	Modify a Question Section	Simple	5
Application Template Management	Add a Question	Simple	5
Application Template Management	Modify a Question	Simple	5
Application Template Management	View a Question	Simple	5
Application Template Management	Delete a Question	Simple	5
Application Template Management	Search a Question	Simple	5
Application Template Management	Delete a Question Section	Simple	5
Application Template Management	Search a Question Section	Simple	5
Application Template Management	View a Question Section	Simple	5
Application Template Management	Add a User Profile Update to the Application Template	Simple	5
Application Template Management	Remove a User Profile Update from the Application Template	Simple	5
Application Template Management	Add a Available Position to the Application Template	Simple	5
Application Template Management	Remove a Available Position from the Application Template	Simple	5
Application Template Management	Add a Contract to the Application Template	Simple	5
Application Template Management	Remove a Contract to the Application Template	Simple	5
Application Template Management	Delete an Application Template	Simple	5
Application Template Management	Publish an Application Template	Simple	5
Application Template Management	Search an Application Template	Simple	5
Application Template Management	View an Application Template	Simple	5
Application Template Management	Copy an Application Template	Simple	5
Discipleship Management	Create a Discipleship	Simple	5
Discipleship Management	Modify a Discipleship	Simple	5
Discipleship Management	Delete a Discipleship	Simple	5

Discipleship Management	Inactivate a Discipleship	Simple	5
Discipleship Management	Search Discipleships	Simple	5
Discipleship Management	View a Discipleship	Simple	5
Appointment Management	Create an Appointment	Simple	5
Appointment Management	Modify an Appointment	Simple	5
Appointment Management	Delete an Appointment	Simple	5
Appointment Management	Inactivate an Appointment	Simple	5
Appointment Management	Search Appointments	Simple	5
Appointment Management	View an Appointment	Simple	5
School Year Management	Create a School Year	Simple	5
School Year Management	Modify a School Year	Simple	5
School Year Management	Delete a School Year	Simple	5
School Year Management	Inactivate a School Year	Simple	5
School Year Management	Search School Years	Simple	5
School Year Management	View a School Year	Simple	5
School Year Management	Add an Available Position to the School Year	Simple	5
School Year Management	Remove an Available Position from the School Year	Simple	5
School Year Management	Add an Appointment to the School Year	Simple	5
School Year Management	Remove an Appointment from the School Year	Simple	5
School Year Management	Add a Discipleship to the School Year	Simple	5
School Year Management	Remove a Discipleship from the School Year	Simple	5
School Year Management	Add a Student Leader to the School Year	Simple	5
School Year Management	Remove a Student Leader from the School Year	Simple	5
School Year Management	Copy a School Year	Simple	5
Registration	Register for a User Account	Simple	5
Application Management	Start an application	Simple	5
Application Management	Modify an application	Simple	5
Application Management	Answer a Question	Simple	5
Application Management	Change the answer of a Question	Simple	5
Application Management	Add desired position	Simple	5
Application Management	Remove desired position	Simple	5
Application Management	Add an Agreement	Simple	5
Application Management	Edit an Agreement	Simple	5
Application Management	Delete an Agreement	Simple	5
Application Management	View an Agreement	Simple	5
Application Management	Cancel an application	Simple	5
Application Management	Submit an application	Simple	5
Application Management	Search applications	Simple	5

Application Management	View an application	Simple	5
User Profile Management	View User Information (User & Profile)	Simple	5
User Profile Management	Modify User Information (User & Profile)	Simple	5
User Profile Management	View Academic Information	Simple	5
User Profile Management	Modify Academic Information	Simple	5
User Profile Management	View Future Plan	Simple	5
User Profile Management	Modify Future Plan	Simple	5
User Profile Management	Add a Training Instance	Simple	5
User Profile Management	Modify a Training Instance	Simple	5
User Profile Management	Add a Conference Attendance	Simple	5
User Profile Management	Remove a Conference Attendance	Simple	5
User Profile Management	Add a Summer Venue Attendance	Simple	5
User Profile Management	Remove a Summer Venue Attendance	Simple	5
User Profile Management	Add a Vision Trip Attendance	Simple	5
User Profile Management	Remove a Vision Trip Attendance	Simple	5
Other User Profile Management	View User Information (User & Profile)	Simple	5
Other User Profile Management	Modify User Information (User & Profile)	Simple	5
Other User Profile Management	View Academic Information	Simple	5
Other User Profile Management	Modify Academic Information	Simple	5
Other User Profile Management	View Future Plan	Simple	5
Other User Profile Management	Modify Future Plan	Simple	5
Other User Profile Management	Add a Training Instance	Simple	5
Other User Profile Management	Modify a Training Instance	Simple	5
Other User Profile Management	Add a Conference Attendance	Simple	5
Other User Profile Management	Remove a Conference Attendance	Simple	5
Other User Profile Management	Add a Summer Venue Attendance	Simple	5
Other User Profile Management	Remove a Summer Venue Attendance	Simple	5
Other User Profile Management	Add a Vision Trip Attendance	Simple	5
Other User Profile Management	Remove a Vision Trip Attendance	Simple	5
Other User Profile Management	Modify a Staff Notes	Simple	5
Other User Profile Management	View a Staff Notes	Simple	5

Table 3. Use Cases

Technical Complexity Factor	**Weight**	**Relevance**	**TFactor**
Distributed System	2	5	10
Performance	1	3	3
End-User Efficiency	1	5	5
Complex internal processing	1	4	4
Reusability	1	5	5

Easy to install	0.5	3	1.5
Easy to use	0.5	5	2.5
Portability	2	4	8
Easy to change	1	4	4
Concurrency	1	3	3
Special security features	1	5	5
Provides direct access for 3rd Parties	1	0	0
Special User Training Facilities required	1	0	0

Table 4. Technical Factors

Environmental Factor	**Weight**	**Relevance**	**EFactor**
Familiarity with UML and/or OO design	1.5	5	7.5
Part-time workers	-1	0	0
Analyst capability	0.5	5	2.5
Application Experience	0.5	5	2.5
Object-oriented experience	1	5	5
Motivation	1	5	5
Difficulty in using the programming language	0.1	3	0.3
Stable Requirements	2	3	6

Table 5. Environmental Factors

Unadjusted Actor Weights (UAW)	25
Unadjusted Use Case Weights (UUCW)	1045
Unadjusted Use Case Points (UUCP)	1070
Technical Complexity Factor (TCF)	1.11
Environmental Factor (EF)	0.536
Use Case Points (UCP)	636.6072
Effort (Person Hours)	1273.2144
Effort (Man Weeks)	31.83036
Effort (Man Months)	7.275510857

Table 6. Use Case Point Analysis Results

Appendix C: Entity Relationship Diagrams

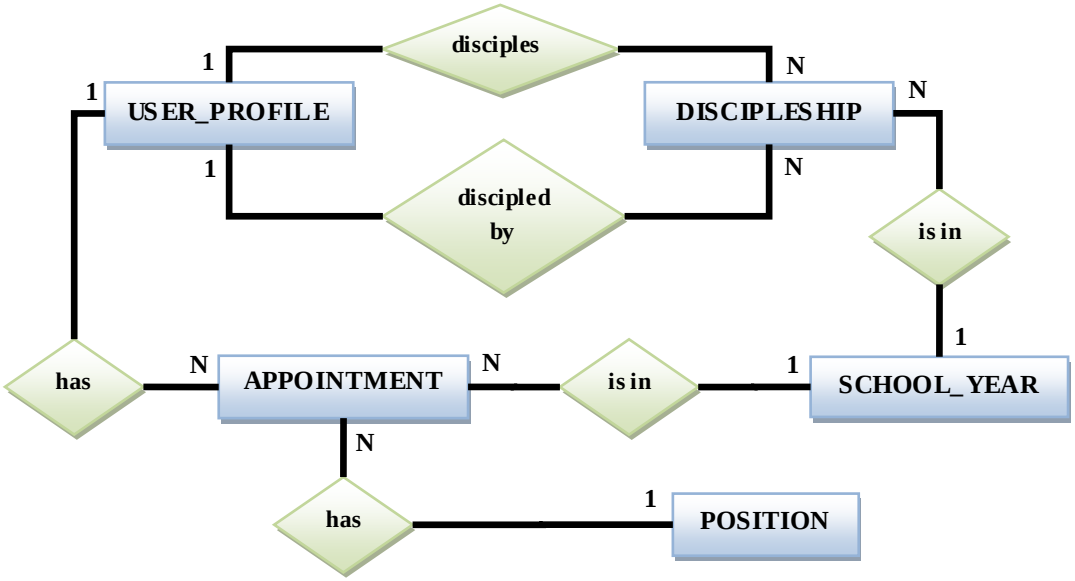


Figure 54. ERD for Yearly Historical Information

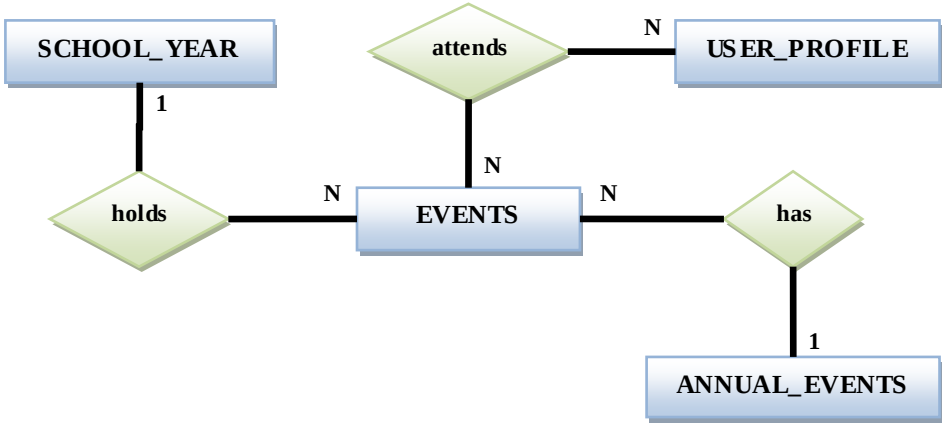


Figure 55. ERD for Events

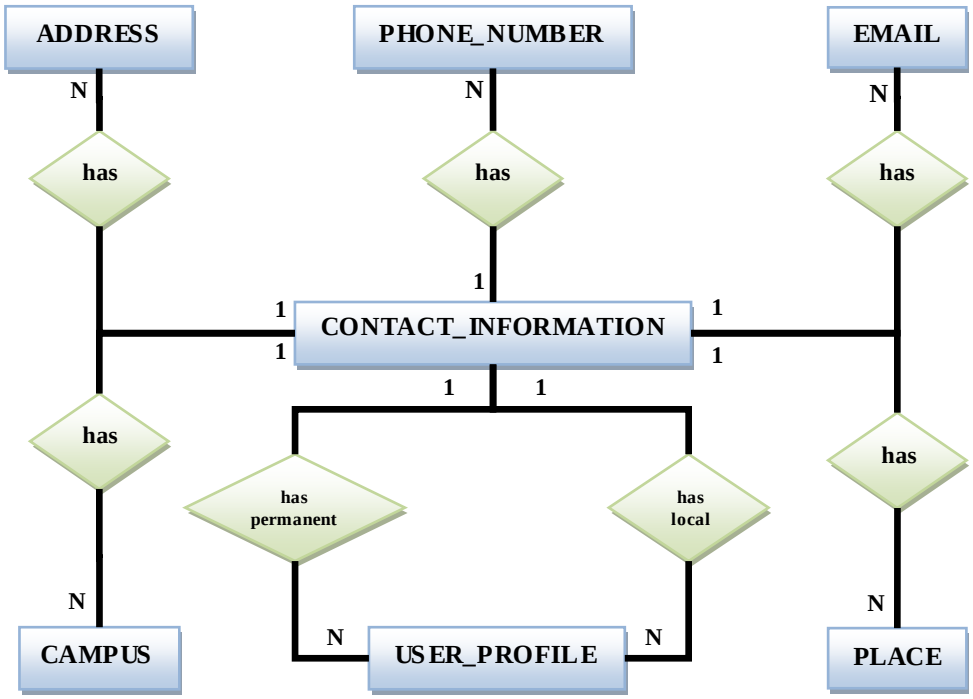


Figure 56. ERD for Contact Information

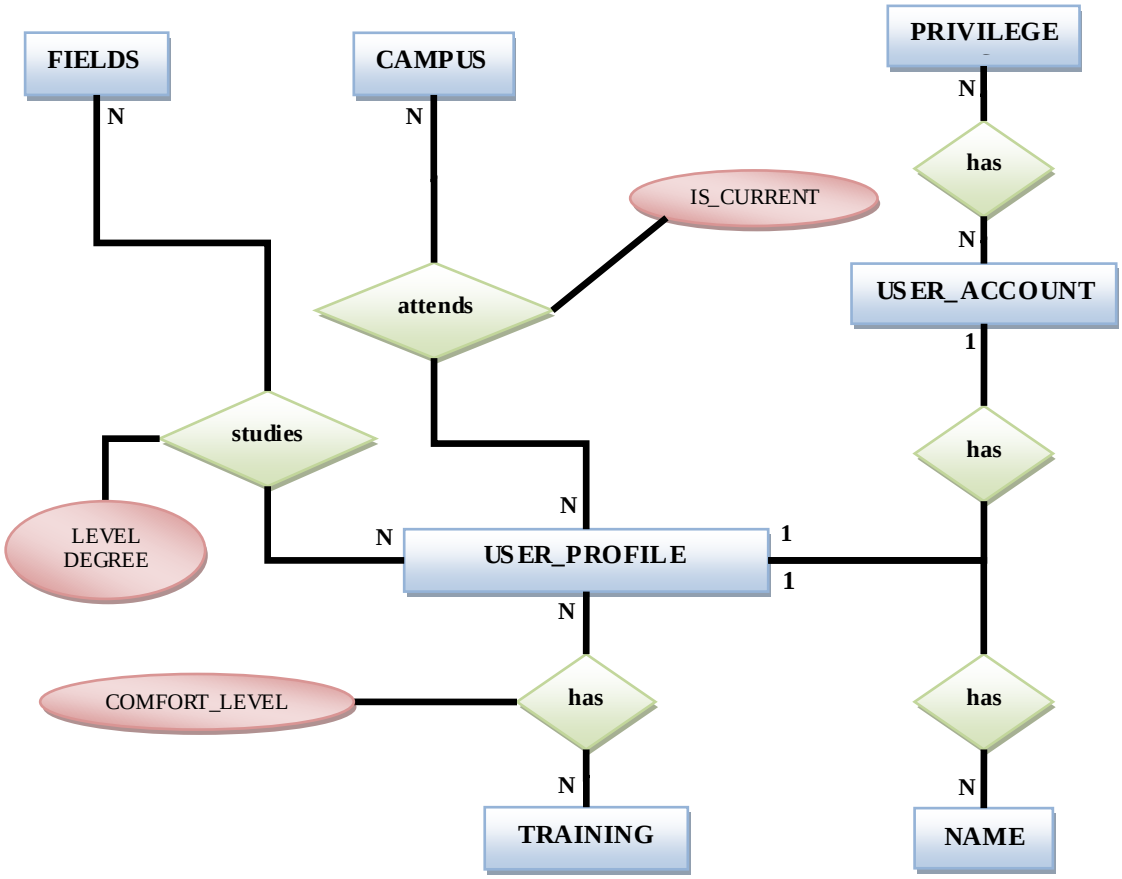


Figure 57. ERD for Academic Information and Account Information

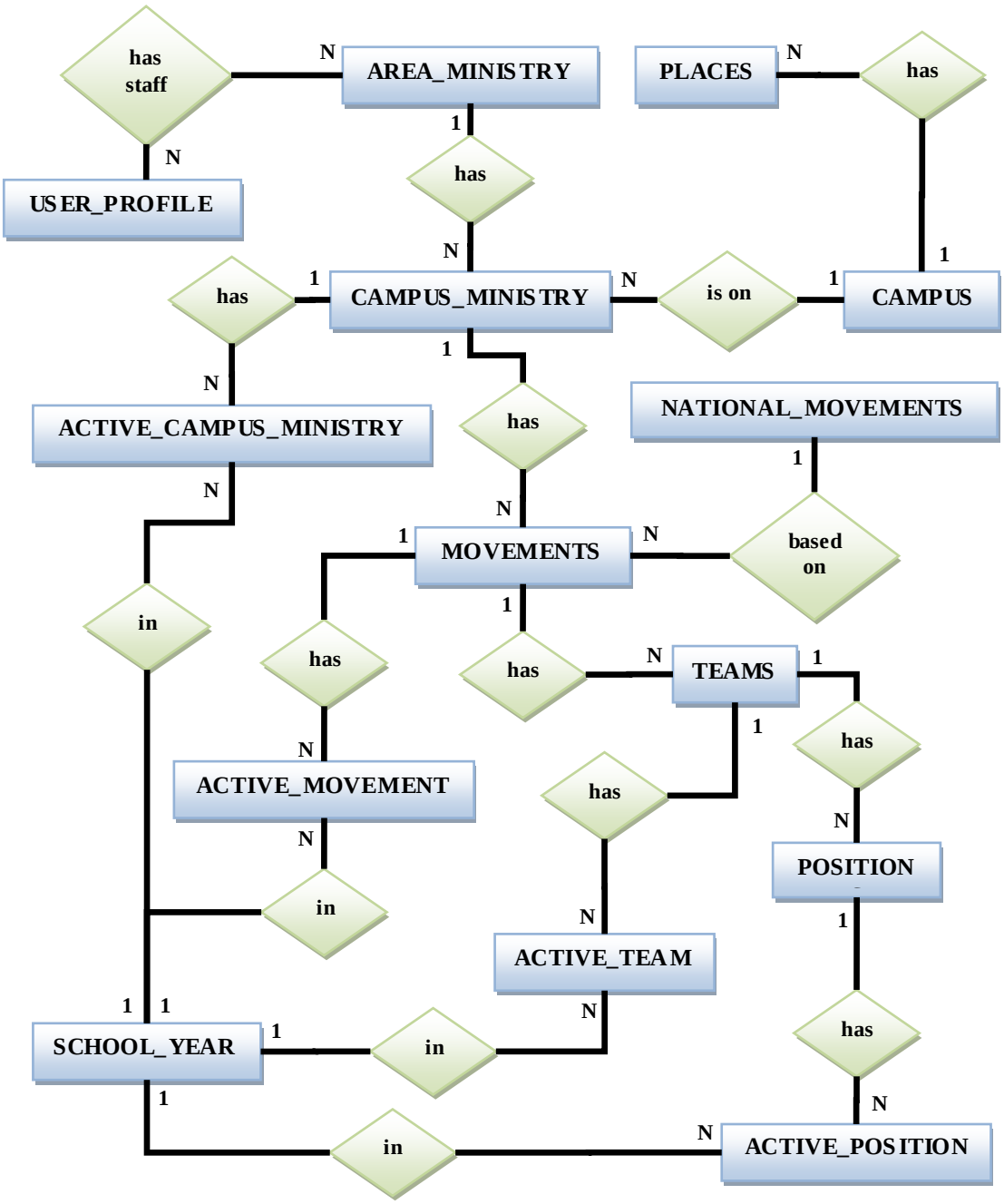


Figure 58. ERD for the Ministry Structure

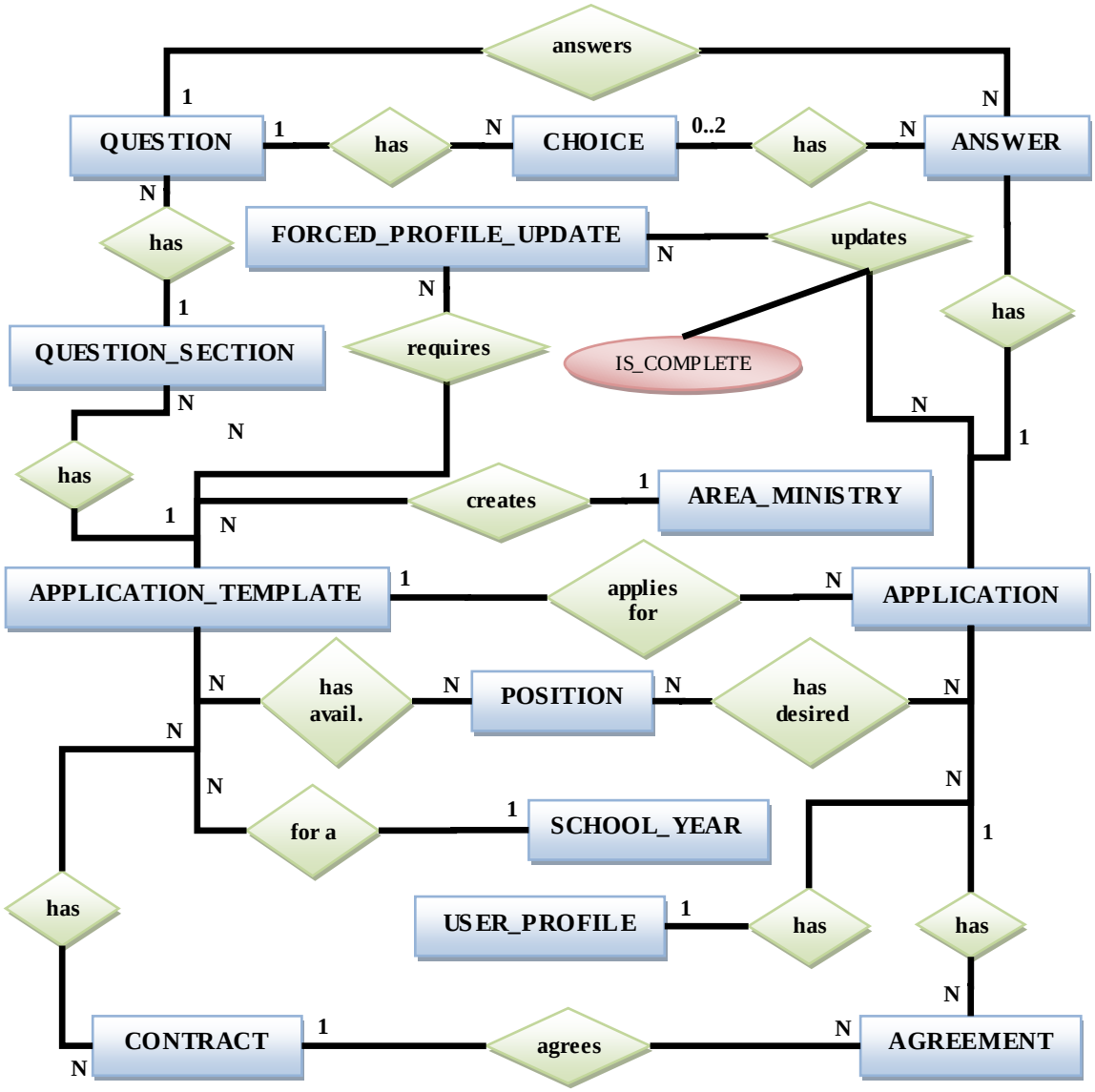


Figure 59. ERD for Application Template

Appendix D: Class Diagrams

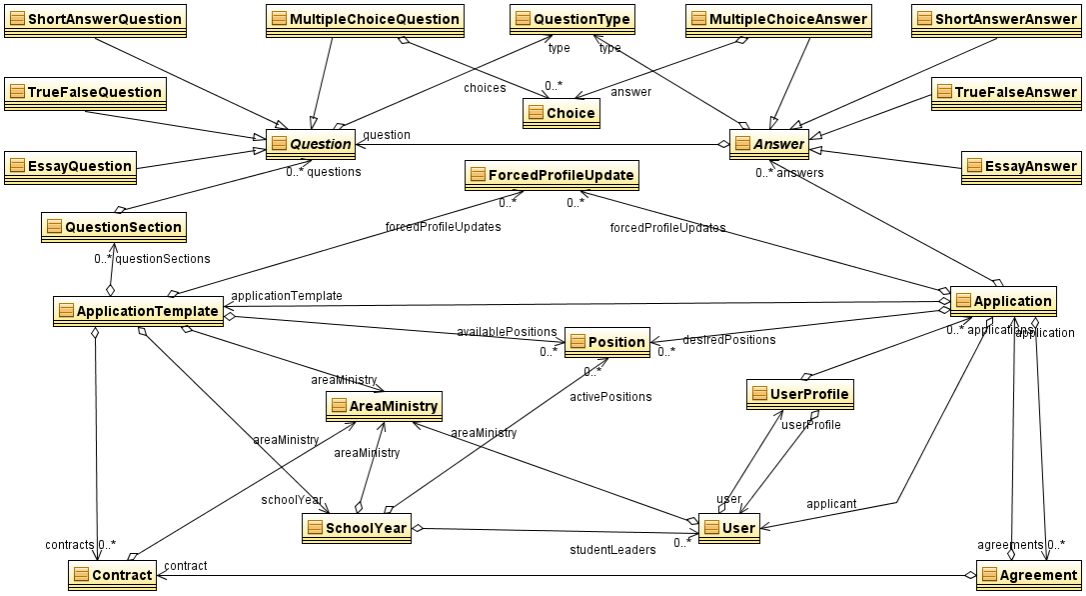


Figure 60. Class Diagram of Application Entities

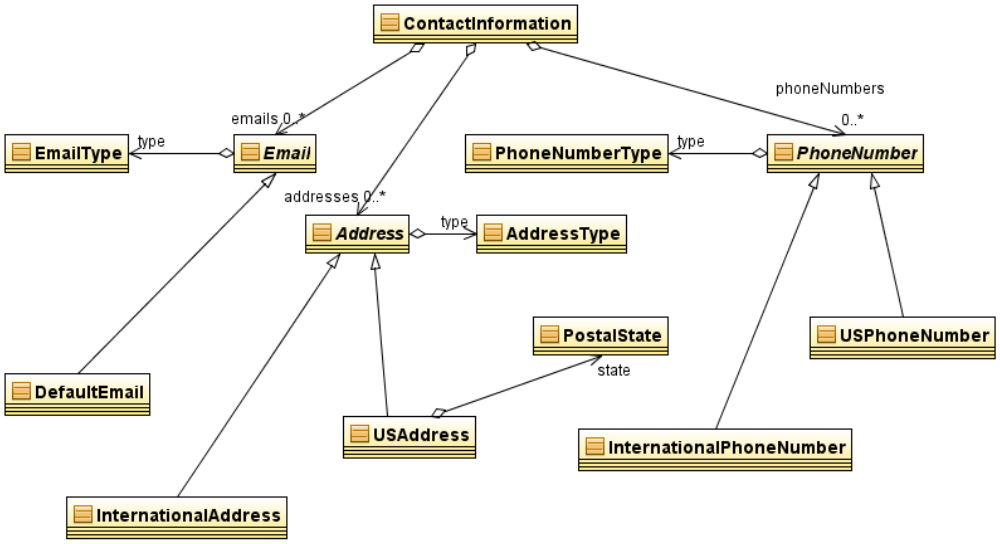


Figure 61. Class Diagram of Contact Information Entities

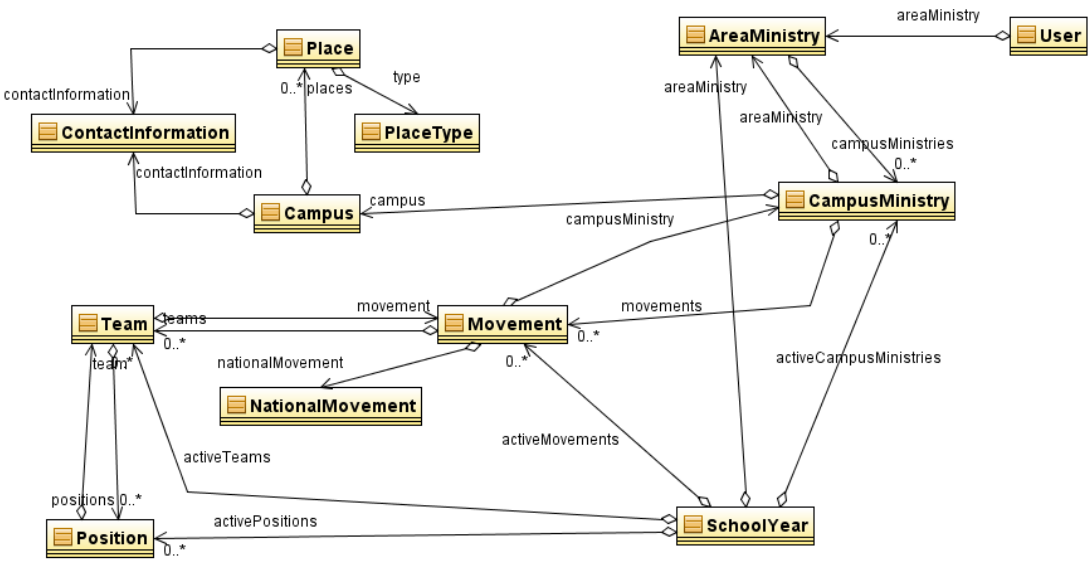


Figure 62. Class Diagram for Ministry Entities

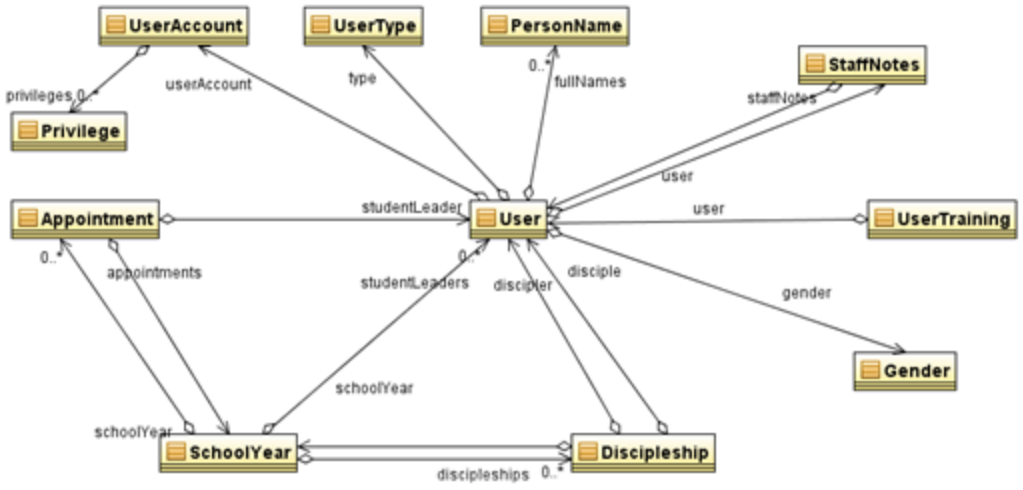


Figure 63. Class Diagram for User Entity

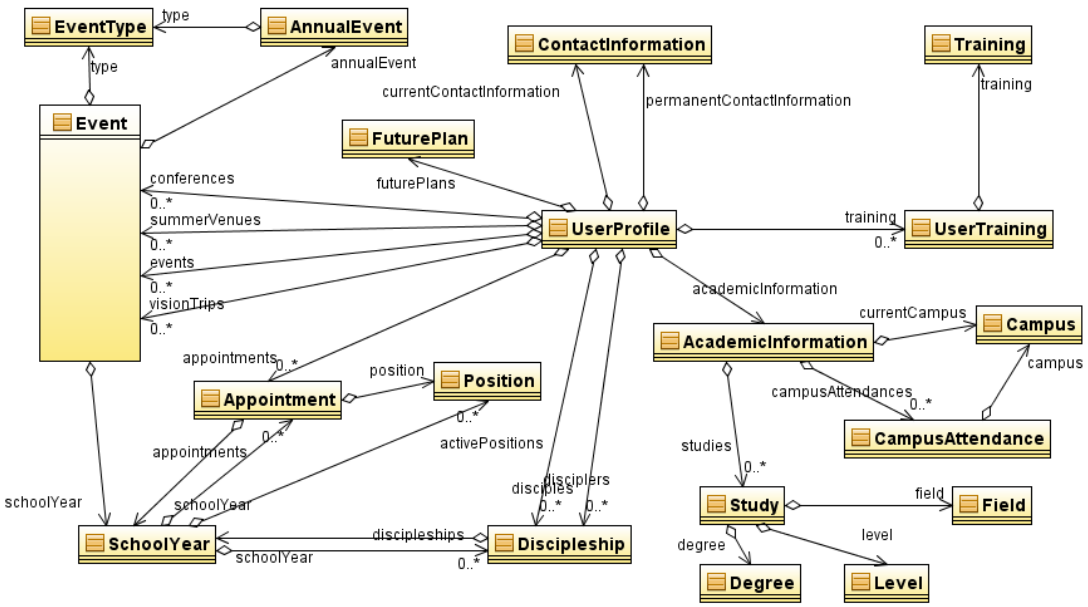


Figure 64. Class Diagram for User Profile Entity