ClubUML

Technical Design Document

*Version 1.4*

Contents

[1. Introduction 4](#_Toc342487031)

[1.1 Purpose 4](#_Toc342487032)

[1.2 Scope 4](#_Toc342487033)

[1.3 Document Organization 4](#_Toc342487034)

[1.4 Audience 5](#_Toc342487035)

[2. Design Overview 5](#_Toc342487036)

[2.1 Approach 5](#_Toc342487037)

[2.2 Architectural Goals and Constraints 5](#_Toc342487038)

[2.3 Guiding Principles 5](#_Toc342487039)

[2.3.2 Flexibility 5](#_Toc342487040)

[2.3.3 Standardization 5](#_Toc342487041)

[2.3.4 Extensibility 5](#_Toc342487042)

[2.3.5 Modular Architecture 5](#_Toc342487043)

[2.4 Design Patterns 6](#_Toc342487044)

[2.5 Design Principles 6](#_Toc342487045)

[3. System Diagram 6](#_Toc342487046)

[4. Application architecture 11](#_Toc342487047)

[5. Database Architecture 20](#_Toc342487048)

Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Date | Description | Author |
| 1.0 | 11/13/2012 | Initial Draft | Tony/Soumya |
| 1.1 | 11/21/2012 | 1st Iteration of Refinement | Mangesh |
| 1.2 | 11/26/2012 | 2nd Iteration of Refinement | Tony |
| 1.3 | 11/28/2012 | Cleanup typos, formalize style | Betty |
| 1.4 | 12/4/2012 | 3rd Iteration of Refinement | Tony |

# 1. Introduction

## 1.1 Purpose

The purpose of this document is to outline the technical design of the ClubUML tool and provide an overview for the ClubUML implementation.

Its main purpose is to -

* Provide the link between the Functional Specification and the detailed Technical Design documents
* Detail the functionality that will be provided by each component or group of components and show how the various components interact in the design
* Provide a basis for the ClubUML Tool’s detailed design and development

This document is not intended to address installation and configuration details of the actual mplementation. Installation and configuration details are provided in technology guides produced during the course of project.

As is true with any high level design, this document will be updated and refined based on changing requirements.

## 1.2 Scope

The Application Design outlined in this document builds upon the scope defined in the Requirements phase.

## 1.3 Document Organization

This document is organized into the following sections:

|  |  |
| --- | --- |
| Introduction | Provides information related to this document (e.g. purpose, term definitions etc.) |
| Design Overview | Describes the approach, architectural goals and constraints, guiding principles, and Java design patterns used in design and development |
| Topology Diagram | Describes the various system components and the integration between them |
| Application Architecture | Describes the application architecture in terms of different layers of application. Describes the presentation layer, business layer, data access layer and resource layer and their relationship to each other. |
| Database Architecture | Describes the overall data model for the ClubUML tool |
| Assumptions and Constraints | Details various assumptions made during design and development of the ClubUML tool |

## 1.4 Audience

The intended audience for this document is the ClubUML project development team, future Northeastern University Students, and vendors who may be interested in pursuing a commercial version of this project.

# 2. Design Overview

## 2.1 Approach

The approach used to create the design of ClubUML has been to follow the OpenUP process.

## 2.2 Architectural Goals and Constraints

The goal of the architecture is to allow two or more users to collaborate on the design of a UML class diagram and reach consensus quickly. This will be accomplished by users logging in to a web-based tool to upload and compare different versions of a diagram. Additionally, it is imperative to implement this design in a short amount of time and at a minimal cost to the developers.

There are severe constraints on availability of resources for this project, both financial and chronological. Thus, this design will use open source software like MySQLfor a database and Eclipse for an Integrated Development Environment (IDE).

## 2.3 Guiding Principles

### 2.3.2 Flexibility

Flexibility is the ability of the application to adapt and evolve to accommodate new requirements without affecting the existing operations. This relies on a modular architecture, which isolates the complexity of integration, presentation, and business logic from each other in order to allow for the easy integration of new technologies and processes within the application.

### 2.3.3 Standardization

Portal services will comply with established industry standards. The standards compliance will not only apply to application development but also to design, platform/infrastructure and other parts of the ClubUML application.

### 2.3.4 Extensibility

Since it is possible that the software functionality of this application could increase over time as more capabilities are added, an open-ended design model was used to allow the code to be easily extended.

### 2.3.5 Modular Architecture

A modular architecture has been maintained for this application so as to facilitate ease of development by the various teams of developers. It also allows the teams to design classes based upon the functional requirements of the code module.

## 2.4 Design Patterns

A Model-View-Controller (MVC) architecture is used throughout the design. This allows the separation of business logic from data and from presentation code. This means that there will be separate code for the Controller (servlets), and for the Presentation (JavaScript/JSP). Similarly, the data will be stored in JavaBeans.

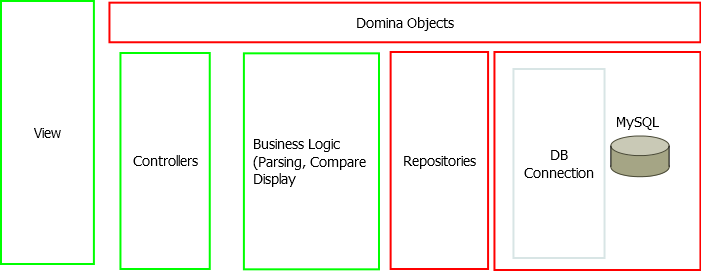
## 2.5 Design Principles

The primary principle that was followed was the Single Responsibility Principle, which states that any class that has two “reasons for change” must have its functionality split into two classes. A “reason for change” is considered any single responsibility; therefore, each class must be limited to one responsibility.

# 3. System Diagram

This section will provide an illustration of the System Architecture along with various system components that will be used in designing ClubUML.

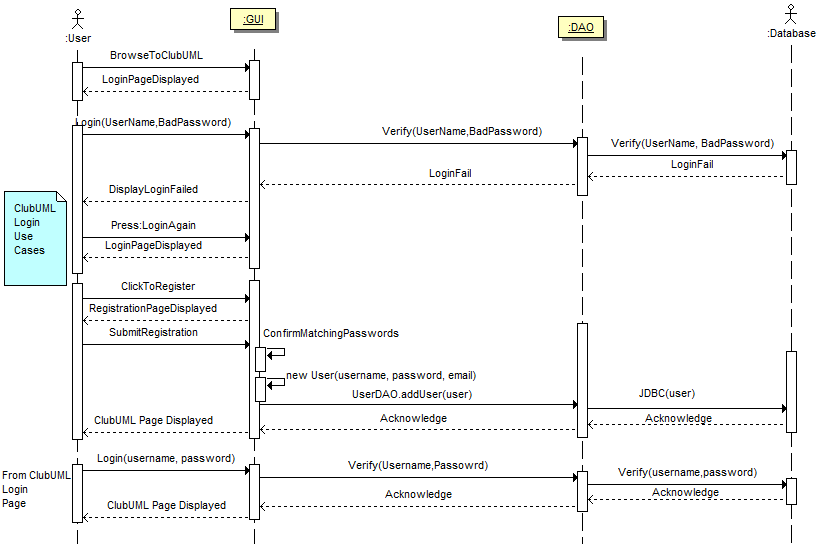
Below is a model of how the system is encapsulated, using the MVC architecture. Each piece of the architecture is separate, but interfaces with others:



The system can be broken down into a few major components.

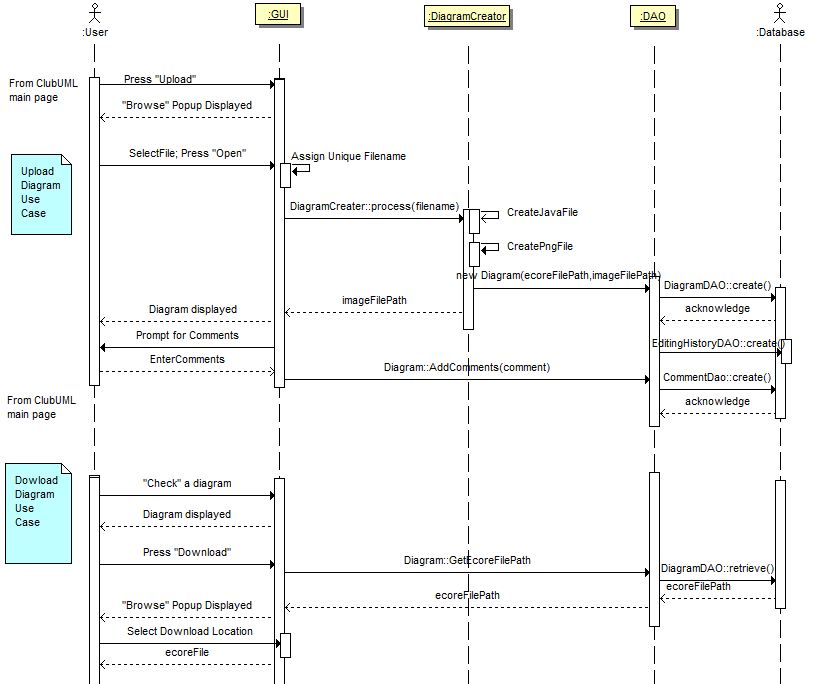
1. Login

The login component is the first entry into the system. This is how the user accesses the system:



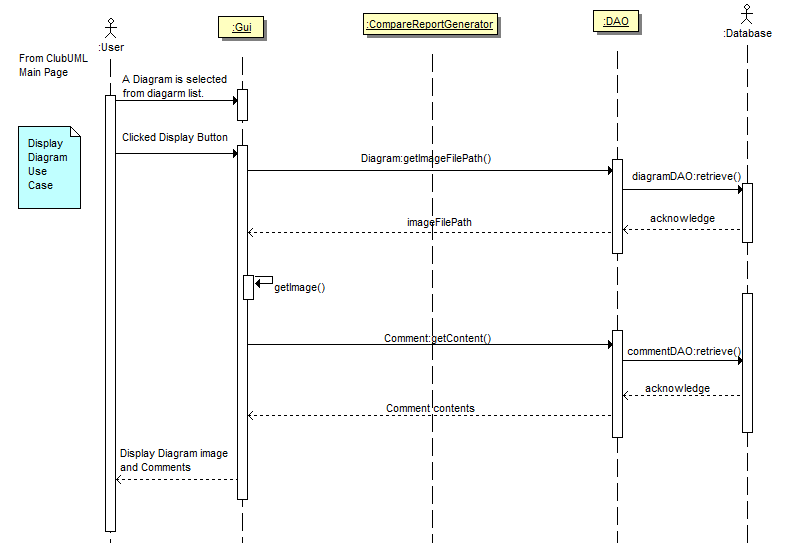
1. Upload/Download Diagram

The Upload/Download component allows the user to upload a diagram to the system for comparison, as well as save a diagram to disk for local editing. Users could then download a file to their own computer to modify.



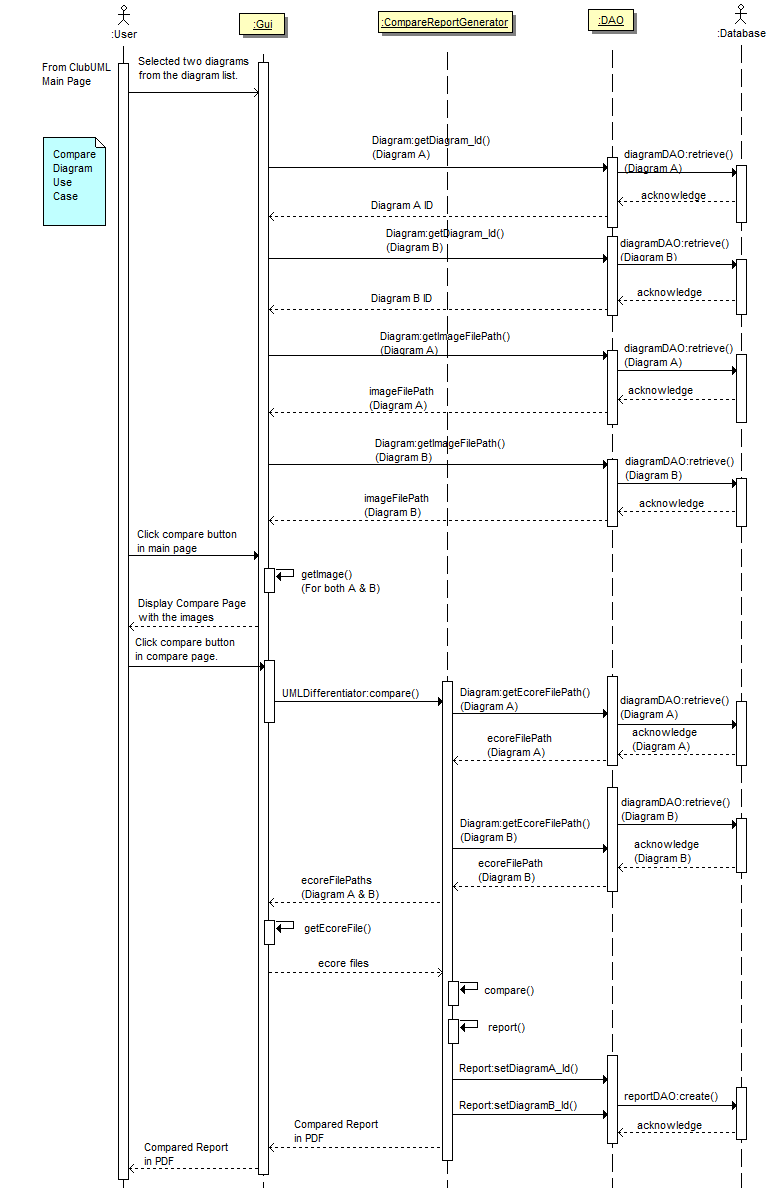
1. Display Diagram

The Display Diagram component is the piece that presents a visual diagram created from Ecore files:



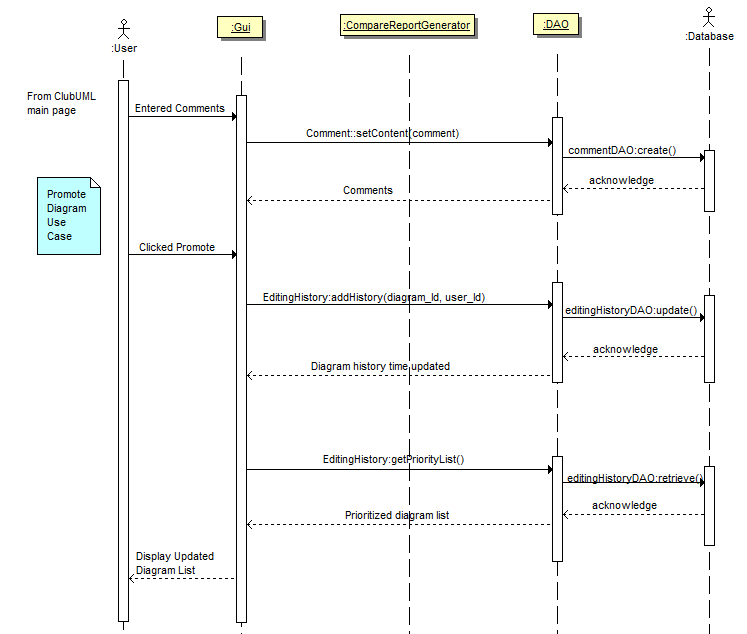
1. Compare

The Compare component is the algorithmic ‘meat’ of the system. It examines the two diagrams and returns a report with analysis:



1. Promote

The Promote component is a mechanism to choose the more accurate of the two diagrams, after the user has seen the results of the comparison:



# 4. Application architecture

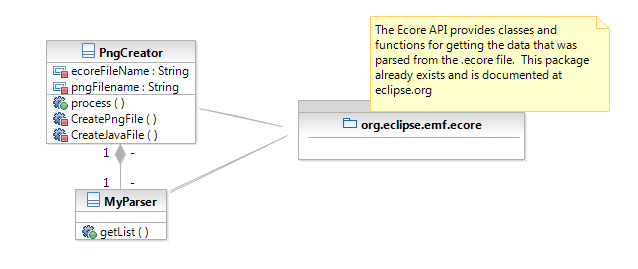
Application architecture defines the various components and their interactions in context of a whole system. Application architecture is the critical software that bridges the architectural gap between the application server and the application’s business logic, thereby eliminating the complexities and excessive costs of constructing, deploying and managing distributed enterprise applications.

There following major components will be explained in more detail, in order to define the functionality that will be provided to the customer.

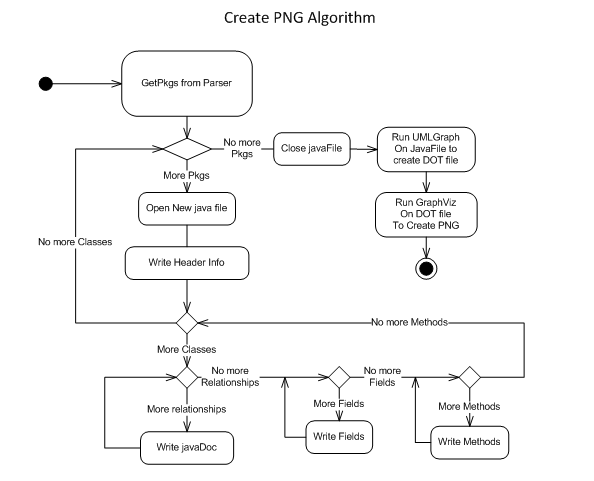
1. Display Diagram

In order to display a diagram, ClubUML must first read an Ecore file, parse it, and then do the actual drawing. All of these steps are included in the ‘Display Diagram’ component.

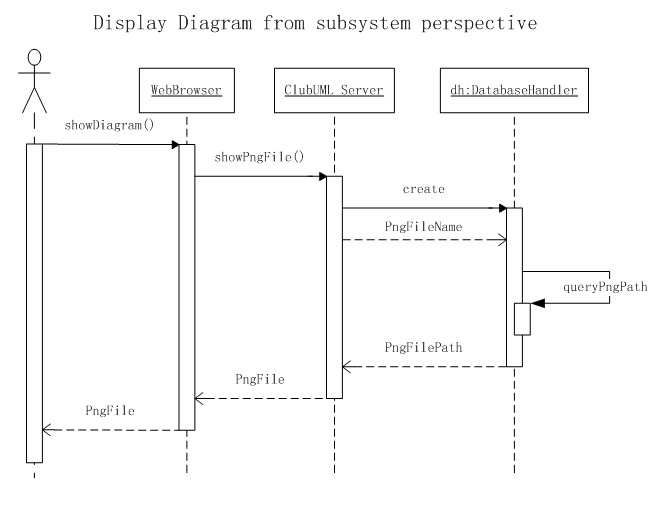
Below are the classes needed to this:

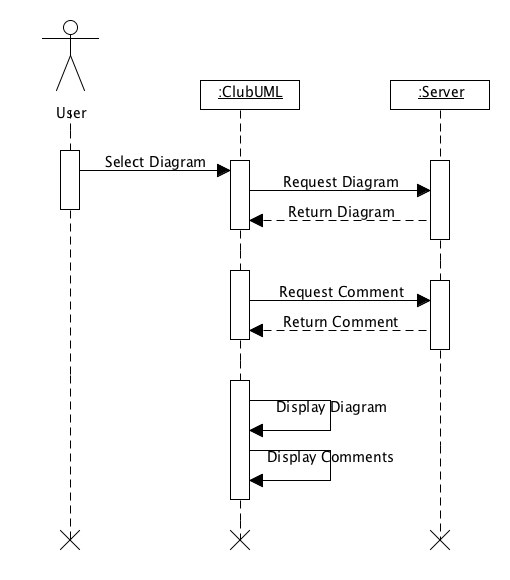


The algorithm that handles the actual creation of the graphic file is shown below.



Sequence diagrams for this are shown below. The first is from the subsystem perspective and the second is from a user perspective:

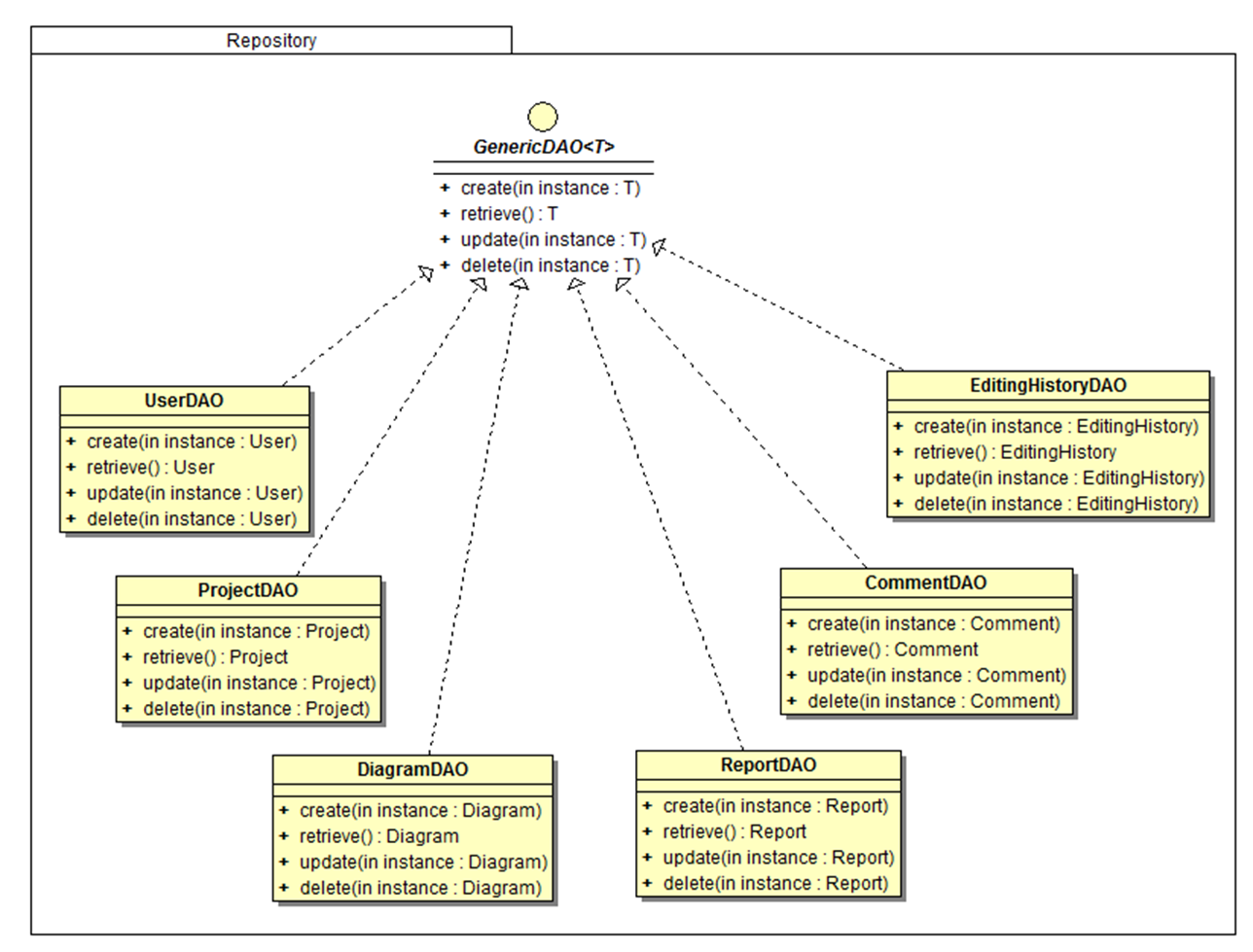




1. Web Services

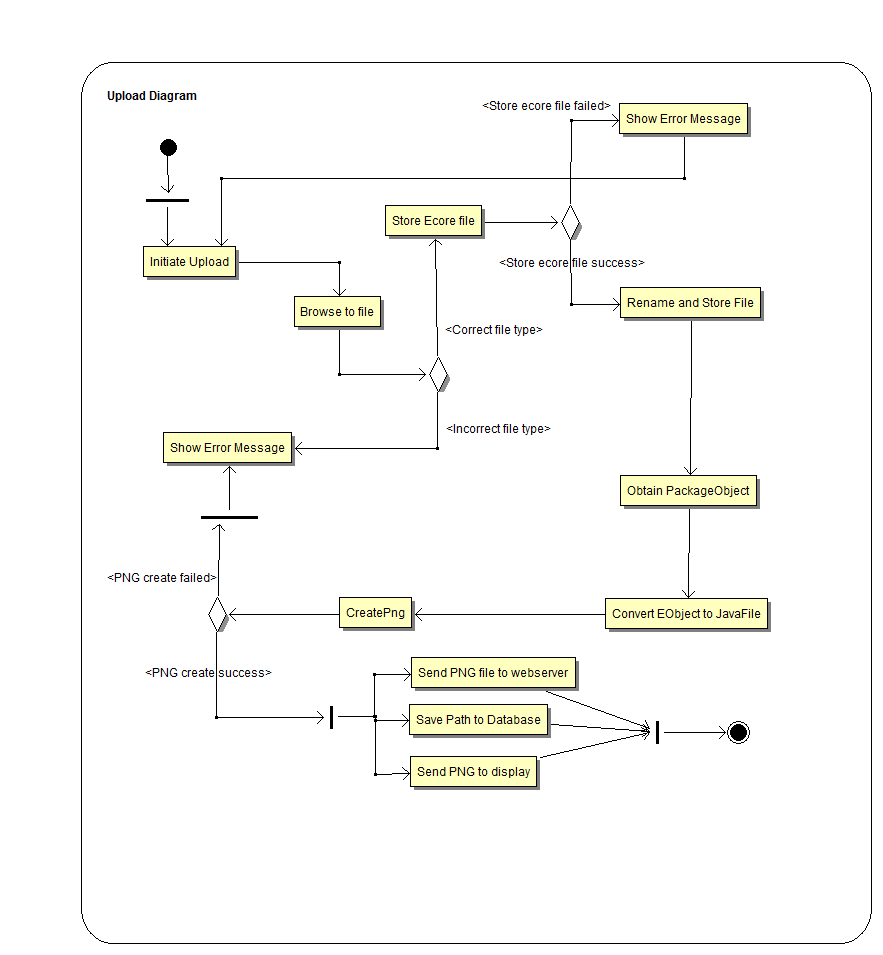
As described in the architecture notebook, ClubUML will use servlets on top of an Apache Tomcat web server to host the web pages. This is considered the Controller in the MVC pattern.

In an MVC pattern, the controller cannot access the data directly. Therefore, ClubUML provides Data Access Objects, or DAOs, for the Controller to get at the data. The following diagram shows the DAOs for ClubUML:

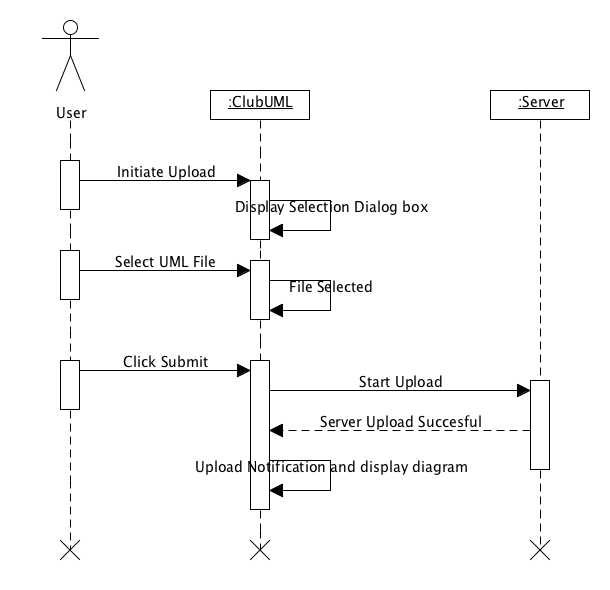
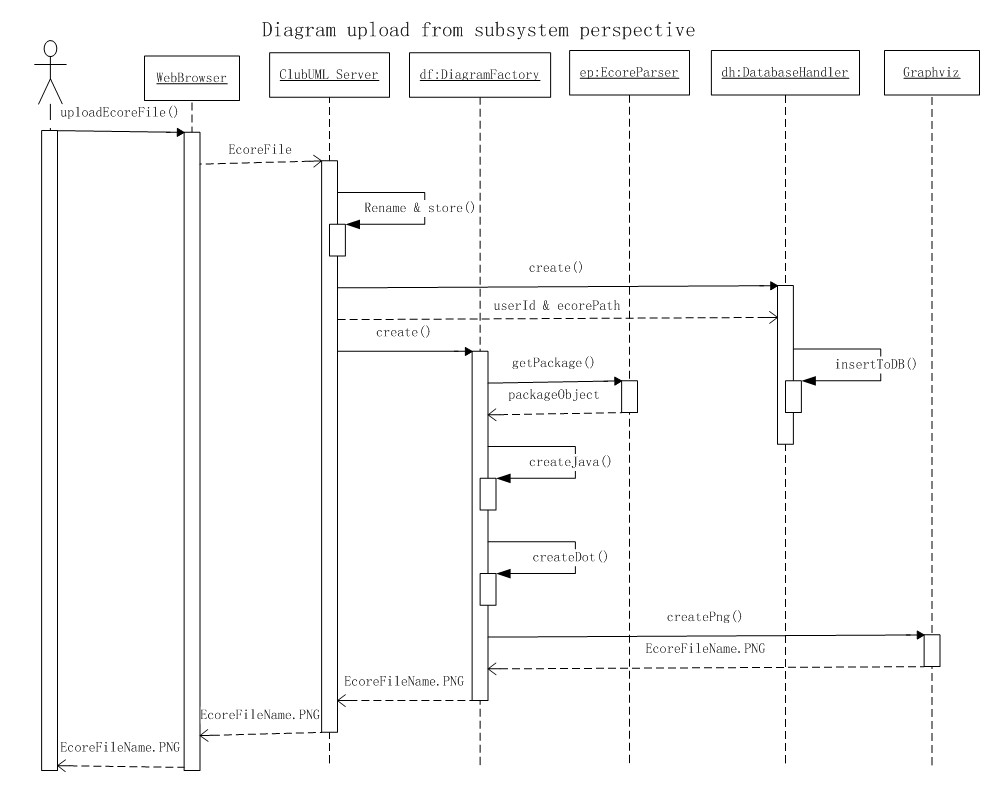


1. Upload/Download Diagram

Uploading a diagram is an integral part of ClubUML. This is the mechanism with which the application gets the diagrams for comparison. The algorithm to do this is as follows:



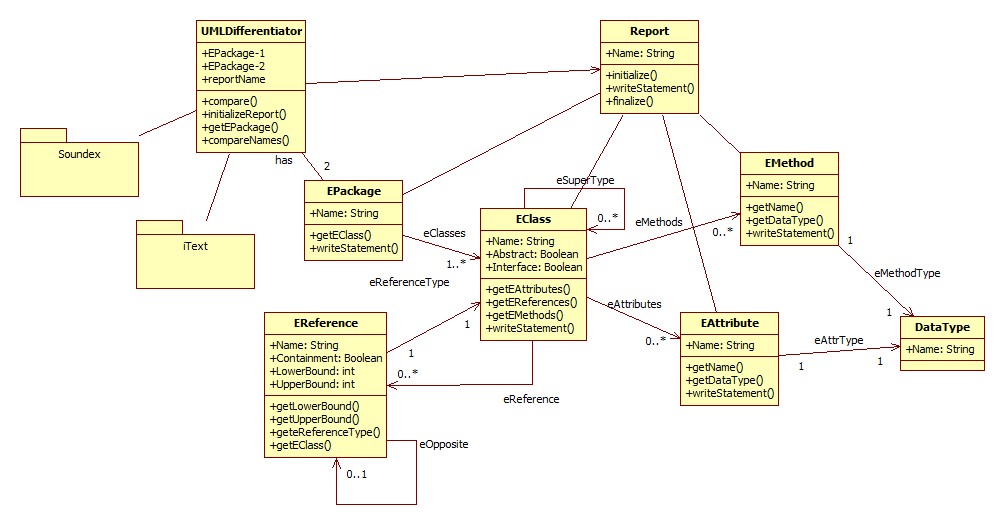
Sequence diagrams for this are shown below. The first is from the subsystem perspective and the second is from a user perspective:



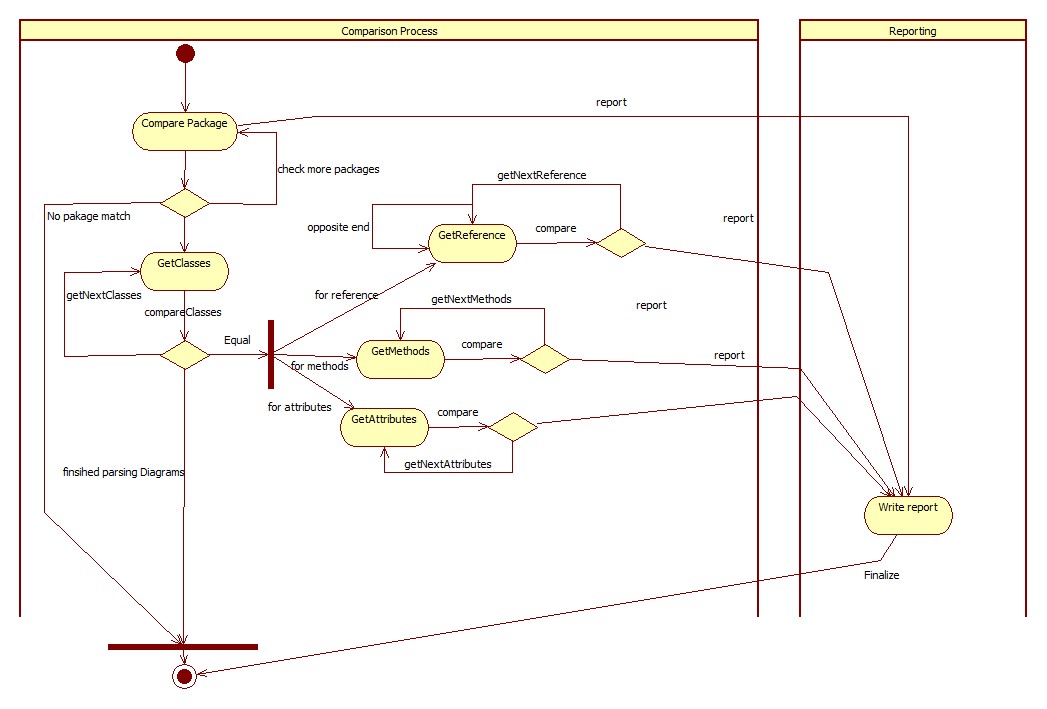
1. Compare

Finally, the comparison component is the heart of ClubUML. This component takes a parsed Ecore file, and then compares all the pieces of the class diagram, and finds similarities and differences. Ultimately, a report is generated with these insights.

In order to do this, the following classes are required:



These classes are used to instantiate the following algorithm:



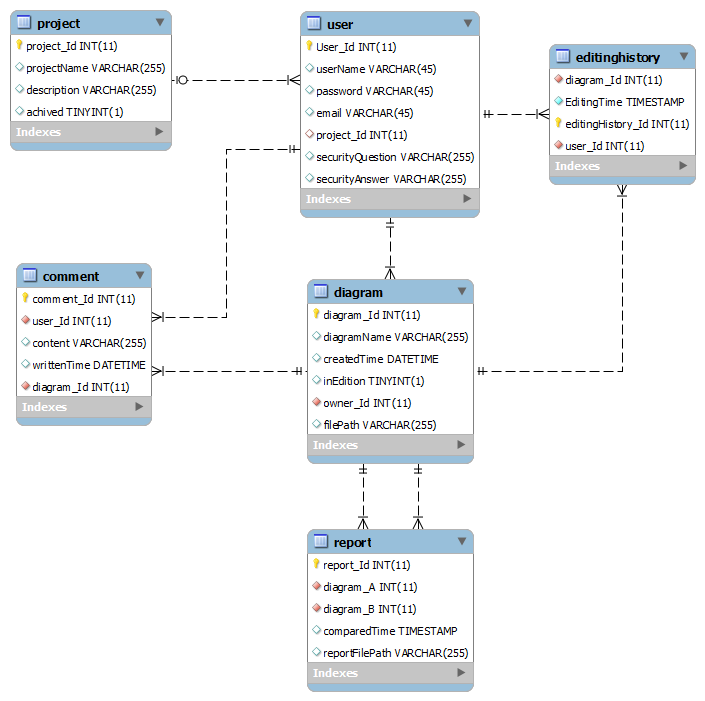
# 5. Database Architecture

Our database was carefully designed to meet the requirements that were defined early on in the process, and then refined over time. ClubUML allows its users to upload E-core format of UML diagrams, translate E-core to an image file and generate a report based on comparison of two diagrams. Users could also comment on any diagram and promote any diagram with a higher priority. For future enhancement, users should be able to create and join projects. However for the first version, the task was narrowed down to only one default project. Although there is a Project table in the current implementation, there is only one project record per instance of the system.

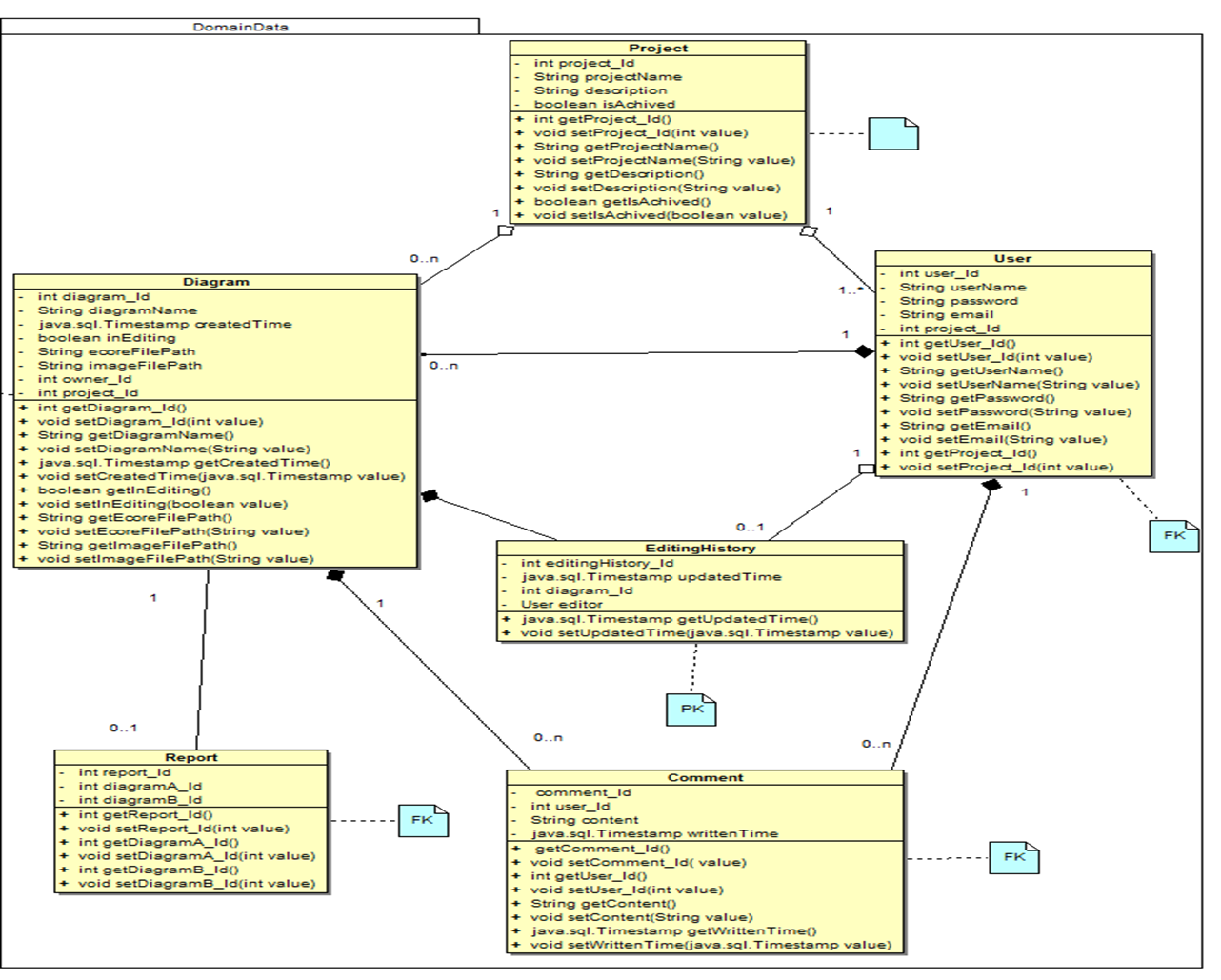
To prevent the content of an Ecore file and image file from taking too much space in the Database, the path/directory for the diagram is stored instead of taking two BLOB columns. In Java code, different file extensions will distinguish the path between the Ecore and the image file.

The Editing History Table is to maintain a priority list of all diagrams based on latest editing time or latest uploading time. In this way, users will be shown the current preferred diagram dynamically changed by promoting or uploading activities.

Other relationships between entities are quite straightforward. Based on the above constraints six tables and foreign constraints were created to set up the following relational DB schema.



Alternatively, this can be viewed in UML via the accessor classes:



**6. Assumptions and Constraints**

1. The user is knowledgeable with the terms and functionality of UML and can create a UML class diagram.
2. It is also assumed that the optimal design of a UML diagram can be achieved through discourse and comparison of two variations of the same diagram.