ClubUML

Architecture Notebook

There is guidance within this template that appears in a style named InfoBlue. This style has a hidden font attribute that allows you to toggle whether it is visible or hidden in this template. Use the Microsoft® Word® menu **Tools > Options > View > Hidden Text** check box to toggle this setting. There is also an option for printing: **Tools > Options > Print**.

# Purpose

This document describes the philosophy, decisions, constraints, justifications, significant elements, and other overarching aspects of the system that shape the design and implementation.

# Architectural goals and philosophy

The goal of the architecture is to allow two or more users to collaborate on the design of a UML 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 the diagram. Additionally, it is imperative to implement this design in a short amount of time and at a minimal cost to the developers.

Therefore, the major non-functional requirements are the following:

* To support a very short development cycle,
* To keep a low cost to developers,
* For ClubUML to be usable on all machine architectures,
* For ClubUML to be accessible from anywhere.

These non-functional requirements impact the architecture in different ways, as explained below:

### The language will be Java.

Java is a cross-platform language that is not tied to specific chip architecture. This means that it can run on any machine, which is a goal of ClubUML. Also, Java is the most familiar language to the developers, which reduces any learning curve in the development cycle.

### ClubUML will use the Ecore model.

The Ecore model, and the associated Java libraries, are, tested, open-source software. Development time will be saved by not having to design and implement a custom meta-model or a parser. .

### ClubUML will not merge two diagrams.

The tool will only compare two diagrams and report similarities and differences to the users. Users will need to make the decision on how to modify their diagram based on this report. A future enhancement to ClubUML may attempt to automate a converged model, or perhaps to implement some sort of interactive convergence tool. However, the short development cycle makes this impossible to attempt in this version. Therefore, ClubUML will compare the diagrams and publish a report that will allow the user to take further action.

### The IDE will be Eclipse.

Eclipse is a free, powerful Integrated Development Environment (IDE) that supports Java. Like Java, it is also available for any architecture. In addition, most of the developers are already familiar with Eclipse which will enable a quicker development cycle.

### The database will be MySQL.

MySQL is a fully-functional relational database that is free and open-source and with which many on the development team are familiar.

### The web server will be Apache Tomcat.

Tomcat is an open-source Java-based web server. Again, many on the team have experience with Tomcat

### A Linux server provided by Northeastern will host the application.

This will be free to ClubUML and will be accessible from anywhere using the internet.

### The UML diagrams will be created using Eclipse Modeling Framework (EMF).

This was decided for many of the same reasons as the decision to use Eclipse as the development environment. It is free and open source. EMF will generate UML diagrams in the Ecore meta-model. ClubUML will be parse the Ecore text files to create a comparison reports.

### The UML diagrams will be drawn using UMLGraph with GraphViz.

UMLGraph and GraphViz are free and open source. Together they will create a graphic image of the Ecore text file that will be uploaded to ClubUML. The use of existing applications to draw an image of the text based Ecore will save time in the development cycle.

# Assumptions and dependencies

The architecture assumes the following:

* A Linux server will be provided by Northeastern University.
* Software packages may be remotely administered and installed on this server.
* The team is fluent in Java.
* The team can create and manage a mySQL database.
* The team has Java-based web service experience: JavaScript, JSP, etc.
* The users will have Eclipse with EMF (or equivalent) to create UML diagrams in .ecore format.
* The users will have a computer that will be able to access the Linux server on the internet.

# Architecturally significant requirements

Please refer to the use cases for the set of requirements.

# Decisions, constraints, and justifications

* Comparisons/convergence will be only be made between two diagrams at a time. Only one user will be allowed to access the comparison tool at a time.
  + This was done for system simplicity. By disallowing concurrent convergence, ClubUML will not get out of sync while converging.
* The only type of UML diagram that will be handled by ClubUML is the class diagram.
  + This decision was made for system simplicity. Future enhancements could include different types of UM diagrams.
* ClubUML will only have one project.
  + Again, this was done for simplicity, although some code is in place to allow multiple projects as a future enhancement.
* MySQL will only store links to the stored UML diagrams in the database, not the diagrams themselves.
  + This will save on the size of the database, which should make it faster. In addition, a MySQL database doesn’t handle ‘blob’ data like some other database applications.
* UMLGraph with GraphViz will generate a graphic file for each diagram
  + Initially, JointJS, a JavaScript diagramming library, was investigated.  JointJS is able to use text based scripts to display classes and arrows.  However, the placement of the elements would need to be calculated programmatically.
  + A second option considered was UMLGraph, in combination with GraphViz.  UMLGraph uses a file that is very similar to a Java file, using the Javadocs above each class to represent relationships between the classes and elements.  The advantage of this method was that by piping the output of UMLGraph through GraphViz the layout of the class diagrams looks good with minimal effort.
  + Either method would need to parse the .ecore file into classes, methods, fields, etc UMLGraph/GraphViz was chosen because the placement of the classes and elements in the diagram is already done in GraphViz.
* Java will be the programming language of choice.
  + This was chosen because it is portable, and it is familiar to the development team.
* Java web services will be used for the front end.
  + This includes JSP (Java Server Pages), as well as generic web languages like HTML and CSS. These mesh well with Java, are handled well with Eclipse, and are more familiar to the developers than their Microsoft counterparts.
* Eclipse will be the preferred IDE.
  + Eclipse was chosen because it is a powerful, free IDE for use with Java. Netbeans was a viable alternative, but Eclipse was chosen as the development team had more experience with it.
* MySQL will be used as our database.
  + MySQL is a relational database that is open source and free. Oracle was considered, however, the licensing is very expensive. PostGREs was also considered, but MySQL was chosen as it met the requirements and the developers have more experience in it.
* Apache Tomcat will be our web server.
  + Tomcat is Apache’s web server with built-in Java support. It is a standard in the field. No alternatives were considered.
* Servlets will be used as the controller mechanism.
  + This is a standard method of writing web services in Java. No alternatives were considered.
* SQL will be the database query language.
  + This is the standard for modern relational databases, and is what MySQL requires. No alternatives were considered.
* The Java Database Connectivity (JDBC) API will interface between Java and MySQL.
  + An abstraction layer/library was considered, such as myBatis, however, the developers were familiar with JDBC and did not want to introduce another learning curve by adding an additional library.
* The diagrams will be uploaded in the Ecore UML file language.
  + This was decided upon because there are Java libraries already written that can handle the parsing of the diagram into a well-defined model.
* The UML diagrams will be generated into .png files and will be saved to disk.
  + This will prevent any redrawing of the same diagrams twice, which will give speed at the cost of disk space.
* The architecture will be a layered architecture.
  + The basic model of the layered architecture will be:

Web Browser

Web Server

Storage

Client

Uses

# Architectural Mechanisms

## Architectural Mechanism 1 – Web server

This will drive the entire project. An Apache Tomcat web server instance will be running on a Northeastern-provided Linux server. Tomcat is the Apache web server that supports Java-based applications. This has been prototyped and is working.

## Architectural Mechanism 2 - Database

All user and project data will either be stored in the database, or have a a reference to their location stored (as in the case of the UML files, both .core and .png). MySQL is the database, because it’s an open source relational database that is familiar to most of the development team.

This has been prototyped and is working

## Architectural Mechanism 3 – UI/Front End (view)

A combination of HTML/CSS will be used to design the webpage User Interface (UI), as most of the team has had some training in this as part of their academic program.

This has been prototyped and has entered the initial phases of development.

## Architectural Mechanism 4 – Comparison Algorithm

This consists of a series of algorithms based on the names of the individual components of UML class diagrams to compare. ClubUML will present a report based on this comparison and will suggestions on the convergence of diagrams.

This has been prototyped and is in development.

## Architectural Mechanism 5 – UML Diagram Parsing

ClubUML will only process diagrams created in the Ecore language. This will allow the use of existing Java libraries to do the parsing.

This has been prototyped and is working.

## Architectural Mechanism 6 – Comments

Comments will be saved as one block of text for an entire diagram. This will eliminate managing the association of comments to UML diagram pieces, as well as some sort of visual interface to select the parts of a diagram.

This has yet to enter development.

## Architectural Mechanism 7 – Diagram Drawing

UMLGraph with GraphViz will create a graphic file for each diagram.  In order to use UMLGraph, the code will need to parse the .ecore file and create a simple .java file which will be the input to UMLGraph. The output of UMLGraph will be processed through GraphViz, which will produce the diagram as a .png file. This file will be saved (and a reference to it stored in the database) and its existence will be checked before attempting to redraw again.

This has been successfully prototyped and is now in development in earnest.

## Architectural Mechanism 8 – MVC Architecture (model-view-controller)

The MVC architecture is used throughout the application. This allows a separation of business logic from data and from presentation code. This means that there will be separate code for the Controller (servlets), and that does not mix with our Presentation code (JavaScript/JSP). Similarly, the data is stored in JavaBeans, which are again separate.

## Architectural Mechanism 9 – Servlet

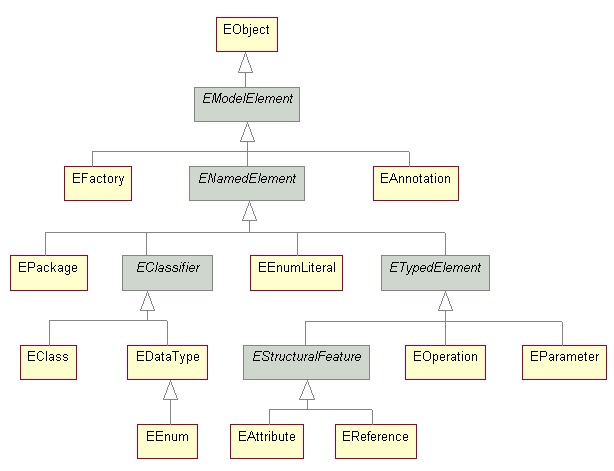
Java servlets will provide our web mechanism. This is where all of the application logic is done. This is considered the ‘Controller’ portion of our M-V-C architecture.

## Architectural Mechanism 10 – Model

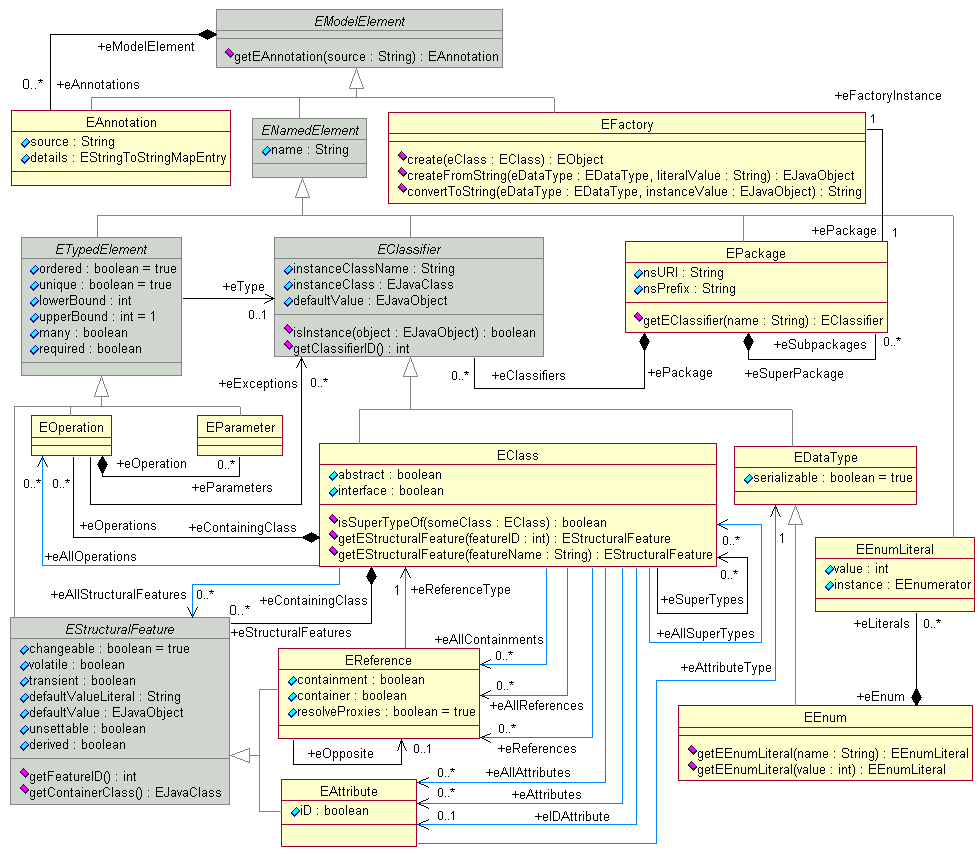
The JavaBeans model is used in the software, which makes the code a system of reusable software components. This is done by writing classes in a particular convention. Using JavaBeans allows us to encapsulate the data and have a standard method of accessing it, which means it can be completely separate from the view or controller portions of the code. These JavaBeans are the only code pieces that directly manipulate data in the system.

## Architectural Mechanism 11 – Ecore Data Model

The Ecore Data Model, as defined here, will model the diagrams in code:



Also, the meta-model for Ecore is given here:



# Key abstractions

* ClubUML is a web application that serves multiple users.
* Users are linked together via the concept of ‘project’.
* Users can upload UML Class diagrams to a project.
* ClubUML provides the ability to compare/contrast two UML diagrams. The report generated will aid in convergence.

# Layers or architectural framework

ClubUML will use the Layer framework. A high level view of the layers is given here:

Web Browser

Client-Side Logic

Web Server

Server-Side Logic

Storage

Client

Uses

Interface

Application Logic

# Architectural views

## Recommended views

* **Logical:** 
  + Layered Architecture
    - The following is a diagram of the layers of the system

Interface

Application Logic

Storage

DisplayDiagram

WebClient

WebServer

Compare

Parse Ecore

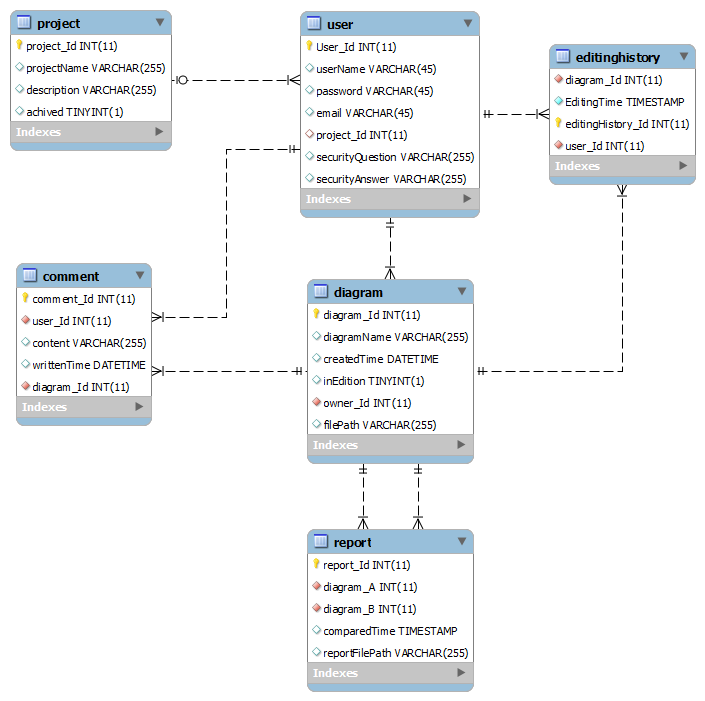
FileManagement

Database

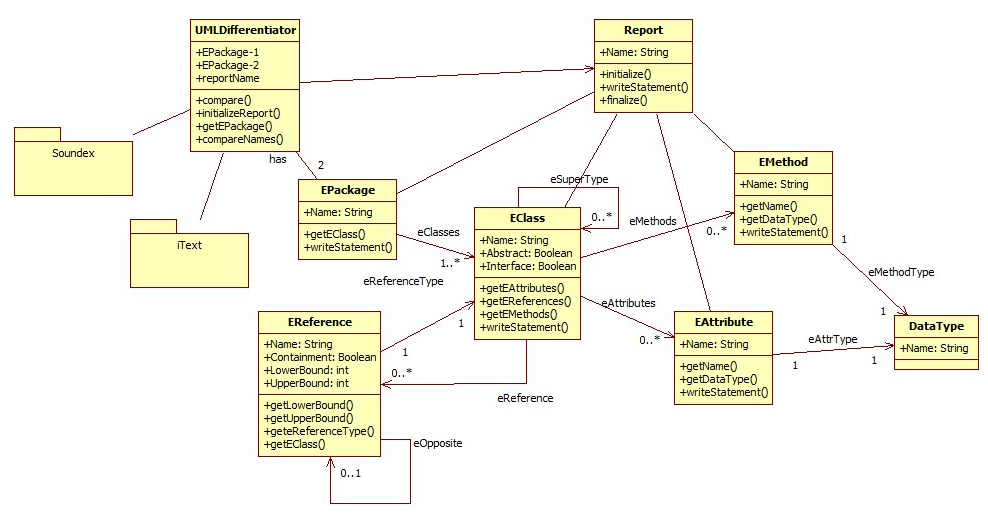
Disk

UserManagement

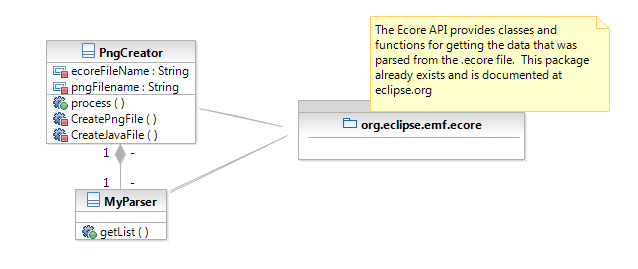
* + Data Model - Ecore Model
    - ClubUML will use the Ecore model, as defined above in section 5, because of the existing libraries that will handle the parsing for the project. It also retains all the relationships between entities, as can be seen up above.
  + Database Model
    - The database model is as follows:



* + Compare Package architecture is given in the class diagram below:



* + Diagram Package architecture is given in the following class diagram:



* **Operational:**
  + ClubUML runs in a single process, aside from the web server which handles multiple connections to the server at a time. That is done by Tomcat automatically, and is not controllable by ClubUML
* **Use case:** 
  + The use cases can be seen in the diagram below, and are all architecturally significant.

