ClubUML

CSYE7945 Spring 2013

Final Report

Beltran, Christian; Do, Richard; Shukla, Prashant; Shen, Yuanwu; Liang, Yidu; Serrano, Christopher; Meng, Xuesong; Li, Shuo; Rezaei Mazinani, Maziar; Zhang, Zhe; Wang, Yingjie; Guo, Dong; Daita Satya, Phani Datta Srinivas Naveen; Huang, Steven; Tan, Huichao; Song, Minhee; Hou, Rui; Vichare, Sarvesh; Huang, Di; Chen, Long

April 2013

Table of Contents

Table of Figures

Revision History

Revision	Date	Author(s)	Notes
1.0	4/8/13	Chris Serrano	Detailed outline with some notes
2.0	4/14/13	Contributions from whole class	Several sections filled in
3.0	4/15/13	Christian Beltran	Updated with Subversion, Analysis of the Process and Lessons Learned sections
4.0	4/18/13		Sections filled in except: Open Issues, GUI images, Test Results, Conclusions. Other minor edits needed.
5.0	4/20/13		Sections complete except: Open Issues, GUI images, Test Results
Final	4/22/13		Final version submitted: Updated Use Case Diagram, added list of team member roles in SE Process section, added GUI Images in Client-Side design section, added Open Issues, updated Test Cases, added Test Results, added deployment instructions

Abstract

This report documents the ClubUML project developed by the CSYE7945 Software Engineering Project class in the Spring 2013 semester. The purpose of the project is to develop an application which provides a way to compare different versions of UML diagrams in order to decide on a final version of the diagram. An initial version of the ClubUML software was developed in Fall 2012, so the work this semester consisted of maintenance. The Spring 2013 version of the software supports uploading UML 2.0 class or sequence diagrams and the ability to merge two class diagrams into a new version, among other enhancements.

Project Overview

Problem Definition

The following is an excerpt from the initial project requirements document provided by Professor Kokar at the beginning of the semester:

Large software engineering project teams include many (sometimes even thousands) of developers. Communication among the developers is a very important activity; it is absolutely necessary to communicate in order to achieve the project goals! However, at the same time, if the communication is inefficient, it can lead to both delays and the increase of the cost of the project. One of the aspects of communication is ensuring that communication events are properly driven by the need to make rational decisions. This is not an easy task since what looks rational to one person may look irrational to another. When developers disagree and spend a lot of time discussing their disagreements, the whole project may suffer. However, at the same time, we need to make sure that communication is occurring; that developers express their personal views on the project related issues. Free communication is needed to ensure that various ideas are explored. A rationale driven process needs to ensure that the best ideas are chosen.

Rationale management, communication, issue resolution and conflict resolution are topics in software engineering. These topics were covered in the textbook used for the Software Engineering class. For instance, Chapters 3 and 12 cover some of these topics. This textbook describes some of the procedures on the management rationale and managing communication processes. However, the textbook does not mention any of the tools that would support such processes. The charge for the Spring 2013 Software Engineering Project team is to work on such a tool.

Although the customer does not want to impose too many constraints on the solution approach, the problem to be solved, at least partially, falls within the “social network” domain. After all, a large team of software developers constitutes a network (network of people, not communication nodes). Thus some inspiration for this project can be sought in the social networks domain.

The social networks most people are familiar with are typically associated with social interactions – sharing pictures, stories, cooking recipes and such. Often social networks fulfill the (hidden) goals of the owners or investors–providing information about the people involved that can then be utilized by the merchants or politicians. At the same time, there are many (hidden) goals of the social network participants (e.g., showing off). The goal of the network for this project needs to be the improvement of the convergence of making decisions in the area of software engineering.

While in the known social networks people use such documents as photographs, in the tool for this project you need to focus on such documents as UML diagrams. UML is a huge language, so the team will have to focus on a small subset of UML to deal with. E.g., the team might decide to focus on just the sequence diagrams and on the support for the convergence of discussions about such diagrams. In one such scenario, developers would discuss (on the Internet) different versions of a sequence diagram stored in a UML tool, providing pros and cons for each. The tool developed in this project might be able

to read the UML versions of the two sequence diagrams, identify the common part of the two solutions (i.e., the part that all parties agree on), identify the differences, propose compromise solutions, and possibly more. The sky is the limit on the possible approaches to this problem.

The Fall 2012 Software Engineering Project team has started work on such a tool. So your work will fall under the label of "maintenance" rather than development from scratch. The focus of the 2012 work was to support the exchange of ideas on class diagrams. The main use case for the system developed by the 2012 team was that a user would upload a class diagram to the system's database and then request the system to compare his/her diagram with any of the diagrams that have been uploaded before. The system then would identify "the same" elements in the diagram, as well as the elements that are in one of the diagrams but not in the other.

This would serve as the feedback to the user, which would (hopefully) cause the user to modify his/her class diagram, or accept one of the diagrams in the database as a better one. The expectation is that the users would eventually converge on a diagram that reflects the consensus the team achieves. The real expectation is that such a tool would result in faster conversion, i.e., in a higher productivity of the software development team.

The customer would like to have a tool that supports rationale based decision support for resolving differences of opinions on all of the UML diagrams – from use case to deployment. Thus the current project can extend the work of 2012 in two directions:

1. Covering more UML diagrams, and
2. Adding some "smarts" to the tool, so that the tool not only displays the differences between two class diagrams, but also suggests the aspects that are "better" in one of the diagrams than in the other.

The team is free to make selections of the types of diagrams they want to work on and on the extent of the smarts for assessing the particular diagrams. These decisions should be made rather early in the project, preferably in the second meeting of the class.

Requirements

The following are requirements for the development tools and process defined by the syllabus at the start of the semester:

- Version control: Subversion
- Development environment: Eclipse
- UML - we need to decide on one tool and then use it throughout the project. The 2012 project did not have a standard UML tool; this should be fixed this time.
- Language: Java
- Group interaction: Blackboard. The instructor will create specific Discussion Boards on Blackboard, as needed.
- Process representation: OpenUP, developed under the EPF (the Eclipse Process Framework) effort

Software Engineering Process

OpenUP

The Open Unified Process (OpenUP) was used in the first few weeks of the course to create an initial schedule and assign roles. OpenUP is an iterative agile framework consisting of four phases, with each phase consisting of several smaller activities, and with a stable release at the end of each phase:

- Inception Phase
- Elaboration Phase
- Construction Phase
- Transition Phase

Additionally, the process defines several roles, including:

- Analyst
- Any Role
- Architect
- Developer
- Project Manager
- Stakeholder
- Tester

At the start of the project, we had several analyst roles in order to develop use cases, a few architect roles to determine more technical aspects (such as what UML tool we would use), a few tester roles (initially debugging the existing code base), and a project manager.

Table 1: Team Member OpenUP Roles

Open UP Role	Team Member Name
Project Manager	Chris S.
Project Assistant	Christian B.
Architecture Lead	Maziar R.
Developer Lead	Richard D.
Analysis Lead	Sarvesh V.
Architecture	Prashant S.
Analysis	Long Chen
Analysis	Naveen S.
Analysis	Dong G.
Developer	Rui H.
Analysis	Steven H.
Analysis	Di H.
Analysis	Shuo L.
Analysis	Yidu L.
Analysis	Xuesong M.
Analysis	Yuanwu Shen
Analysis	Huichao T.
Developer	Minhee S.
Analysis	Yingjie W.
Architecture	Zhe Zhang

After the first few weeks of using OpenUP to determine roles and tasks, we introduced the Scrum process. Since Scrum allows all team members to define and switch tasks from meeting to meeting, we phased out the OpenUP process to avoid the inherent conflicts involved with using two different processes concurrently.

Table 2: Team Member Scrum Roles

Scrum Role (focus)	Team Member Name
Project Manager	Chris S.
Scrum Master	Christian B.
Team Member (Architecture Expert/Test)	Maziar R.
Team Member (Server Developer Expert)	Richard D.
Team Member (Analysis Expert)	Sarvesh V.
Team Member (Analysis/Pathfinding)	Prashant S.
Team Member (Developer)	Long Chen
Team Member (Analysis/Test)	Naveen S.
Team Member (Developer/Pathfinding)	Dong G.
Team Member (Developer)	Rui H.
Team Member (Analysis/Developer)	Steven H.
Team Member (GUI Developer Expert)	Di H.
Team Member (GUI Developer)	Shuo L.
Team Member (DB Developer Expert)	Yidu L.
Team Member (DB Developer)	Xuesong M.
Team Member (Developer)	Yuanwu Shen
Team Member (Analysis)	Huichao T.
Team Member (Analysis/Developer)	Minhee S.
Team Member (Developer/Pathfinding)	Yingjie W.
Team Member (Developer/Pathfinding)	Zhe Zhang

Scrum

Scrum, another iterative agile framework, became the software development process used for the majority of the semester. Since most of us did not really know each other and had different work/school schedules, Scrum allowed for us to achieve the end goal together. There was no one person or group choosing what had to be done like there can be with a more traditional development approach. We were all learning the process, application and tools together.

We chose to treat this semester like one big sprint. This took out the sprint planning and sprint completion events that happen before and after a sprint. There just was not enough time to learn it and do all of this with what we had. Even though this led to some confusion and organizational difficulty, the team did pick up this limited Scrum process well. Stories were written for every week and tracked as they got completed in the backlog. Stories were added mid-week too as we saw new work that could be done.

We had limited assigned roles from the typical Scrum process. A lack of expert knowledge in this area led to this decision. We only had the key roles: a product owner (Professor Kokar), ScrumMaster, the project manager, and the development team. Some of the development team focused on particular areas like server development, analysis, database development and GUI development (see Figure 2 for details). Other roles we did not include were just auxiliary, helping more with sprint-to-sprint planning that we did not do.

Subversion

Subversion was used as our software versioning and revision control tool. This was a carry-over from last semester's tool of choice. Unlike last year that did all the work on trunk, also called a never-branch system, we decided to use branches when appropriate. This methodology is called a branch when necessary system and is very popular with Subversion use. The users commit their daily work to trunk. The work committed must past testing and of course compile. Also the work committed must not be too large. If it becomes large and affects many areas of the code, we would branch off. We would also branch off when there were multiple people working in the same area. These rules helped keep us productive and maintained better stability of the code by keeping conflicts to a minimum. Another rule we followed was to use the story numbers as the comments for the commits so we could track the changes better. We also utilized the tag feature to easily get builds identified and controlled.

System Analysis

Overview of Fall 2012 System

In order to analyze what enhancements would be best to employ, an overview of the work done in the previous semester (Fall 2012) is necessary. The Fall 2012 project features consisted of:

- Uploading Ecore class diagrams
- Display Ecore class diagrams (though there were some rendering issues)
- Download image of an uploaded diagram
- Compare two diagrams side-by-side
- Generate a PDF report of differences between elements of two diagrams
- Add a comment to a diagram after comparing it with another

GUI Overview

Figure 1 shows a screenshot of the Fall 2012 project, specifically the main page encountered after logging in. The user is shown the currently selected diagram to the left of the screen, though there are problems with rendering the text. Below the diagram are any comments left after going through the compare process. On the top-right of the screen, the user can choose a new file to upload. Below the upload form, there is the list of all diagrams that have been uploaded, with buttons to Display, Download, or Go to compare.

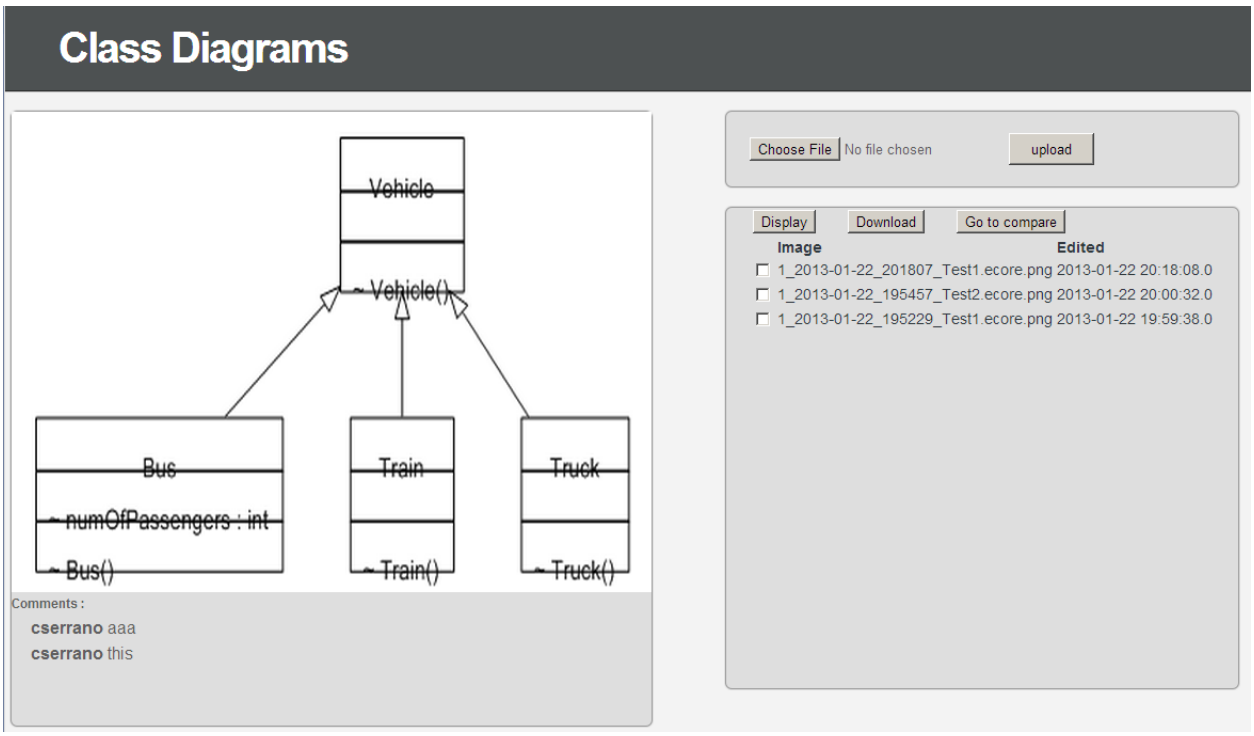


Figure 1: Fall2012 Diagram Display Screenshot

Figure 2 shows the page that appears after the user selects the “Go to compare” button. The software did not always correctly show the selected diagrams. In this instance, the diagram on the left and on the right should be different. This was a bug that needed to be resolved.

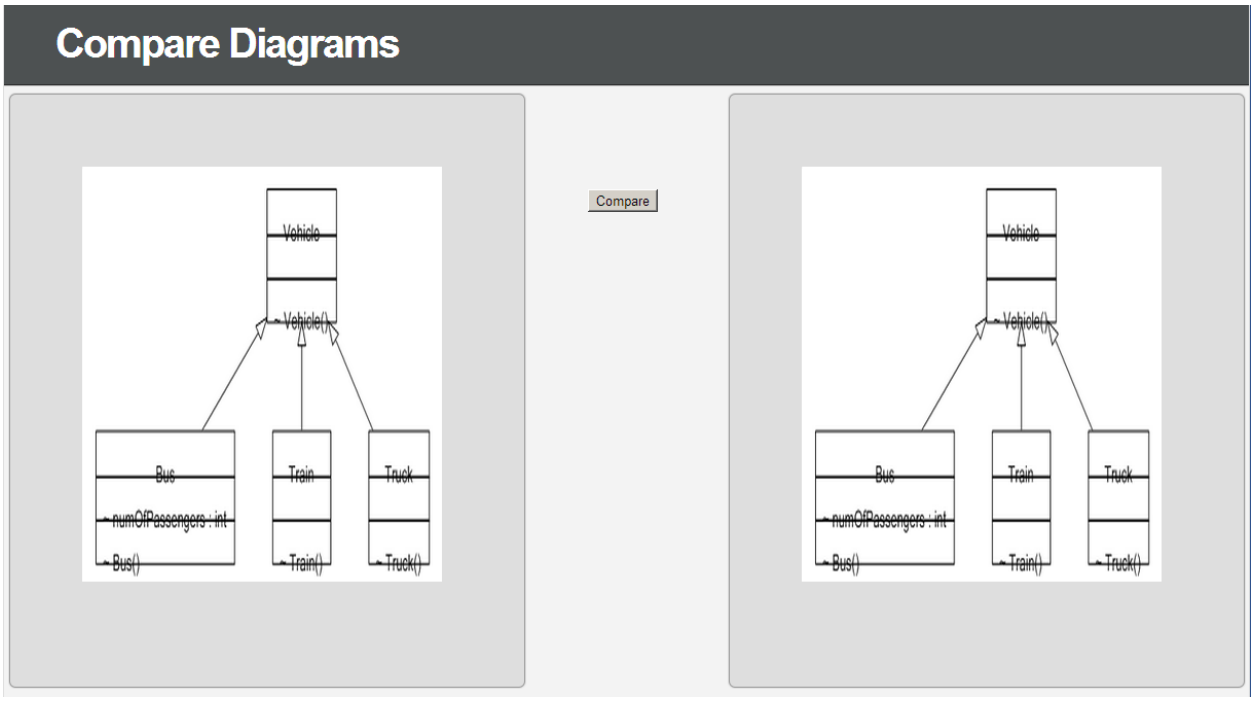


Figure 2: Fall2012 Compare Page Screenshot

After selecting the Compare button in Figure 2, the user arrives at the page in Figure 3. The user can enter a comment and save it by clicking Promote under one (and only one) of the diagrams. Additionally, a PDF pops up with a text-based report of the similarities and differences in Ecore elements pops up in a new window, as shown in Figure 4.

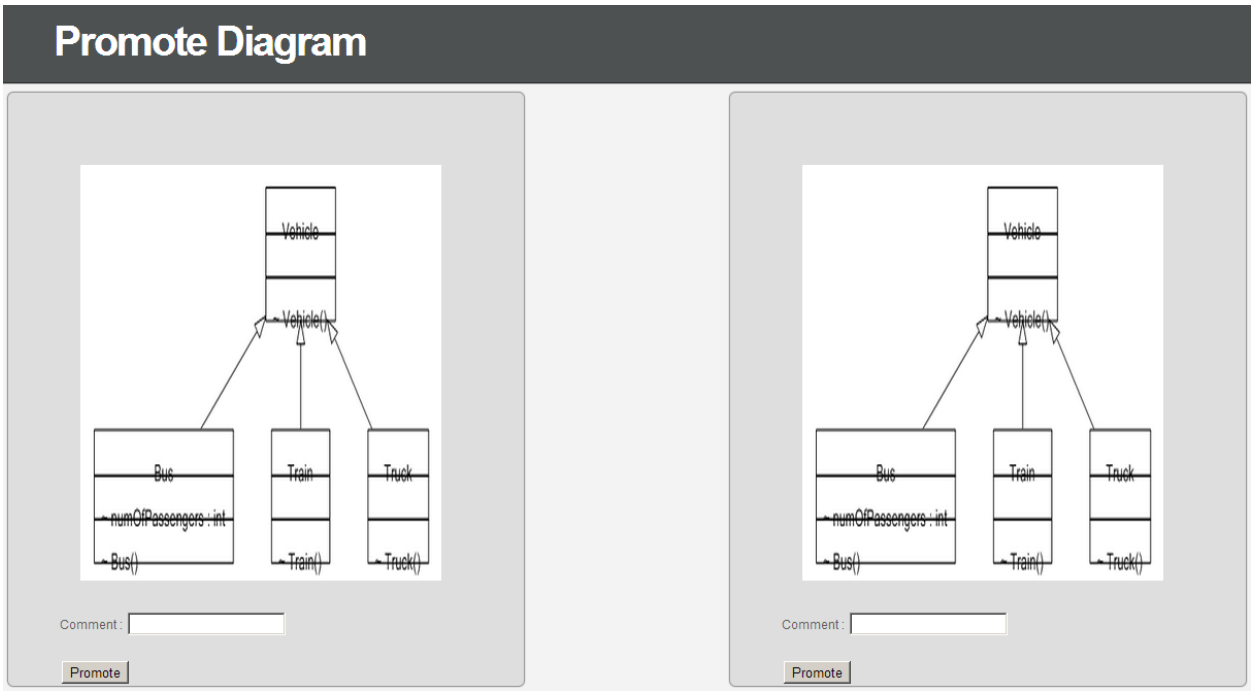


Figure 3: Fall2012 Promote Page Screenshot

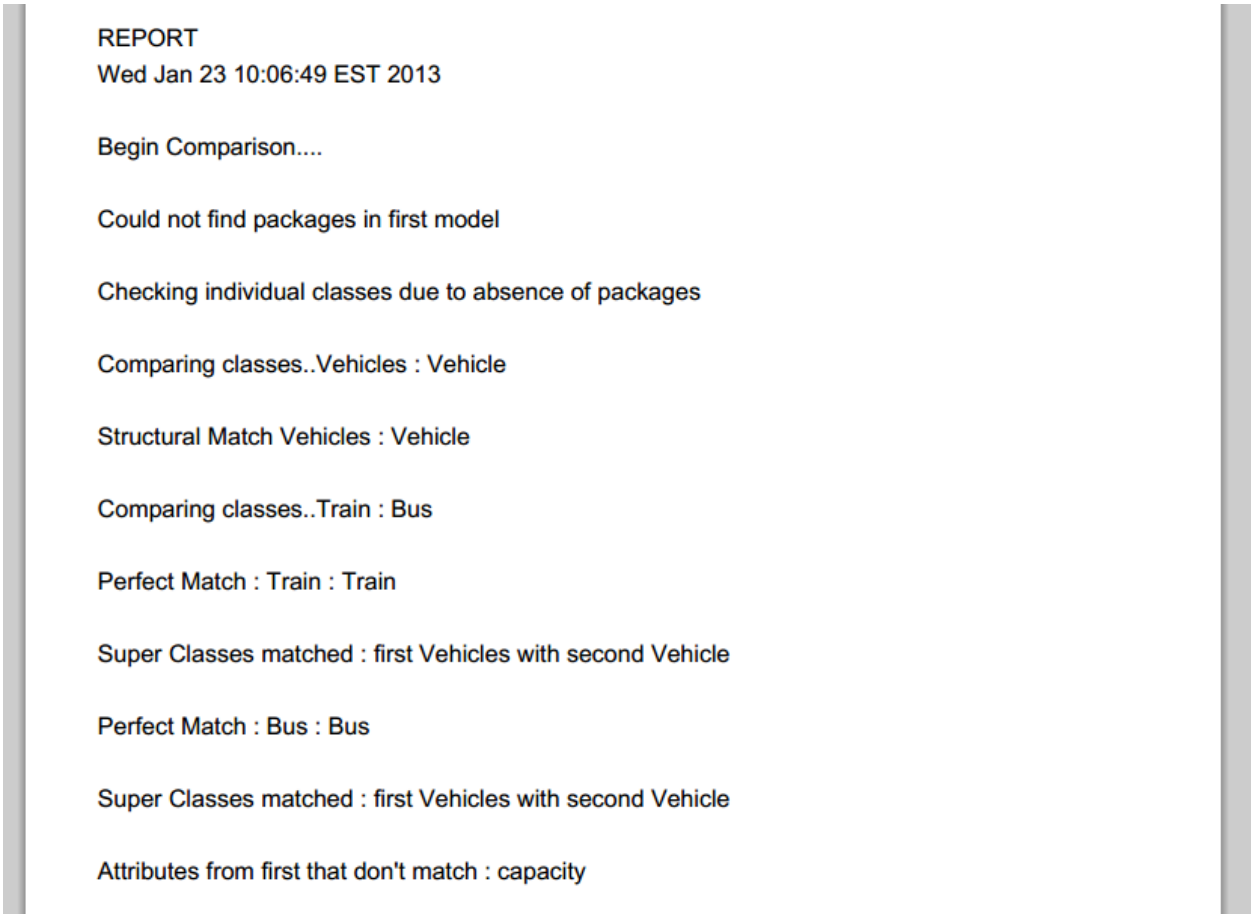


Figure 4: Fall2012 Report PDF Screenshot

Once the user enters a comment and clicks Promote in Figure 3, the user is returned to the main display page as shown in Figure 1.

Codebase Overview

The Fall 2012 codebase consisted of the following packages:

1. Compare
2. Diagram
3. Domain
4. Repository

The Compare package is used to compare Ecore class diagrams and generate reports. The class diagram for the Compare package is shown in Figure 5.

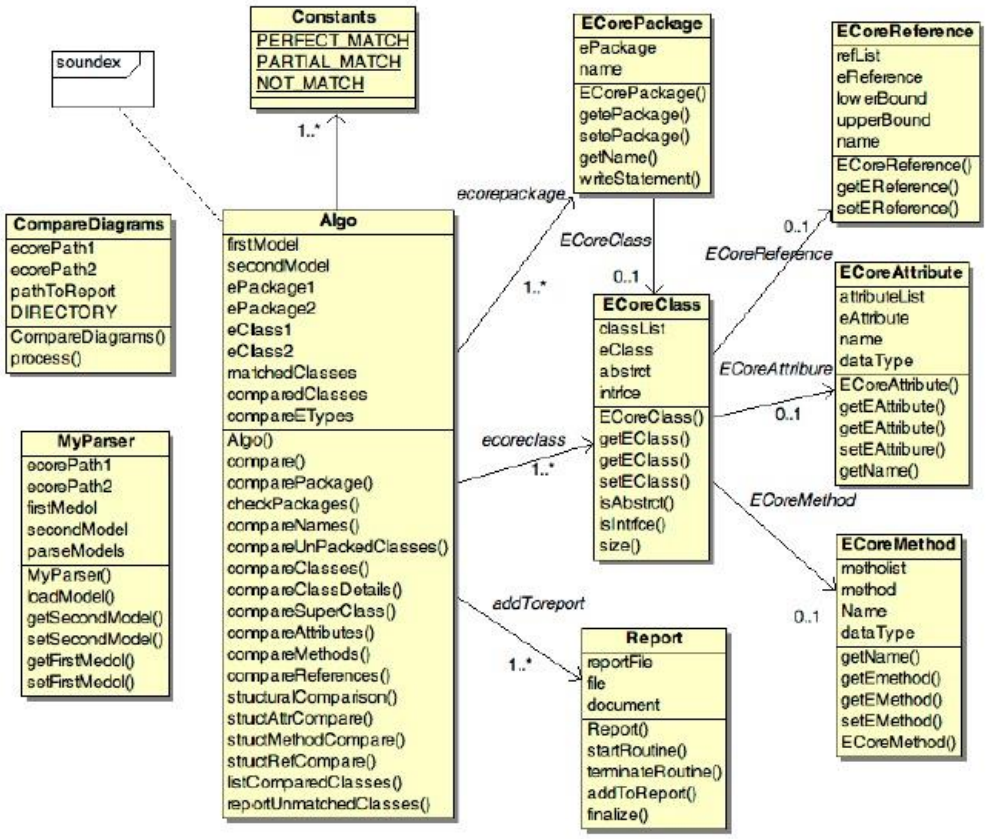


Figure 5: Compare Package Class Diagram

The Diagram package is used to generate PNG images of Ecore files.

Figure 6: Diagram Package Class Diagram

The Domain package contains classes that hold and modify data from the database.

Figure 7: Domain Package Class Diagram

The Repository package contains classes which deal with directly sending and receiving data from the database.

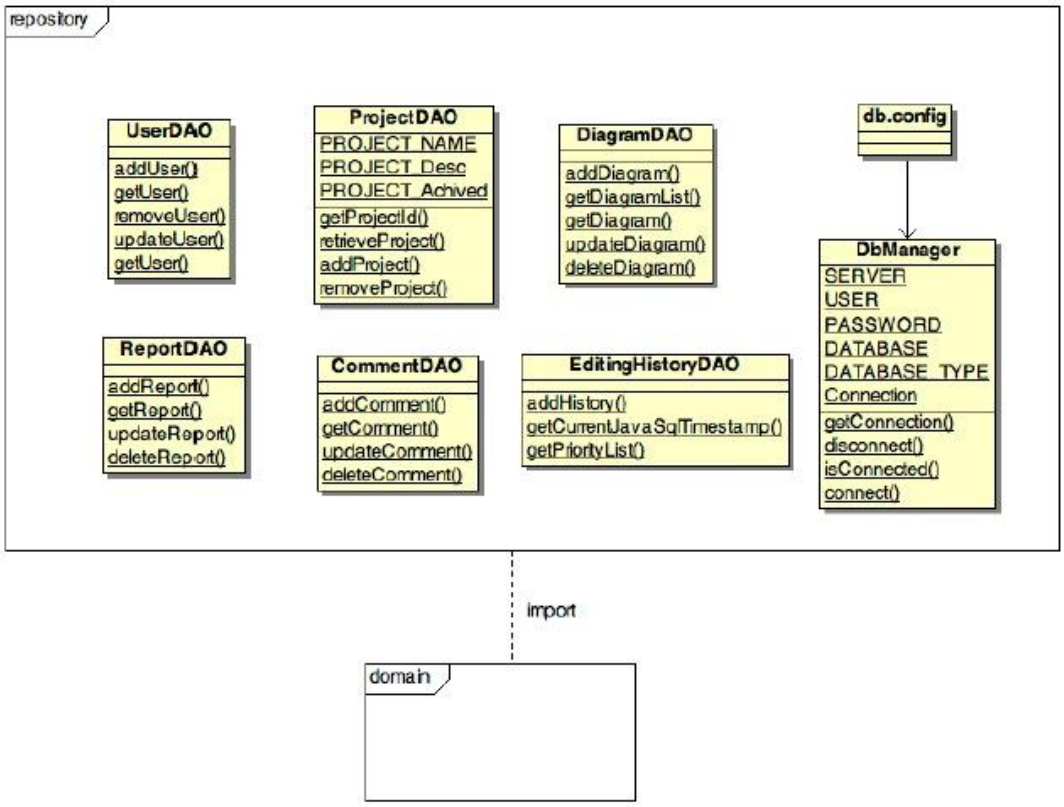


Figure 8: Repository Package Class Diagram

Enhancement Overview

We focused much of our effort on maintenance work to address many aspects of the Fall 2012 project that could use work, including:

- Getting the software to run properly on the College of Engineering's Rho server since many aspects only worked when running the project locally, including:
    - Ability to register a new user
    - Ability to log in
    - Rendering an image of the uploaded diagram
- Properly rendering class diagram images (addressing text alignment issue)
- Correctly displaying the selected image

Beyond debugging and improving the existing features, we focused on several enhancements:

- Uploading and displaying Papyrus XMI class diagrams
- Uploading and displaying Papyrus XMI sequence diagrams
- Comparing and merging Papyrus XMI class diagrams
- Downloading a diagram (including UML files, PNG image files, and any associated files)
- Downloading the entire project (all diagrams and their associated files)

These enhancements are described in more detail in the following sections.

Use Cases

Use Case Diagram

A diagram of all use cases in the ClubUML system is shown in Figure 9. There is a RegisterNewAccount use case for new users to the system who need to register a user name and password. UploadDiagram describes a user which merely wants to upload a diagram to the database, without doing any compare or merge functions. MergeDiagram describes the user manually choosing which elements to keep and which to discard when comparing the differences in two diagrams. CompareDiagram describes the user comparing two diagrams and generating a report of the differences between diagrams (without making a merged diagram). Lastly, the DownloadDiagram and DownloadProject use cases describe the enhanced download options available to the user. All diagrams aside from RegisterNewAccount include the basic Log in use case.

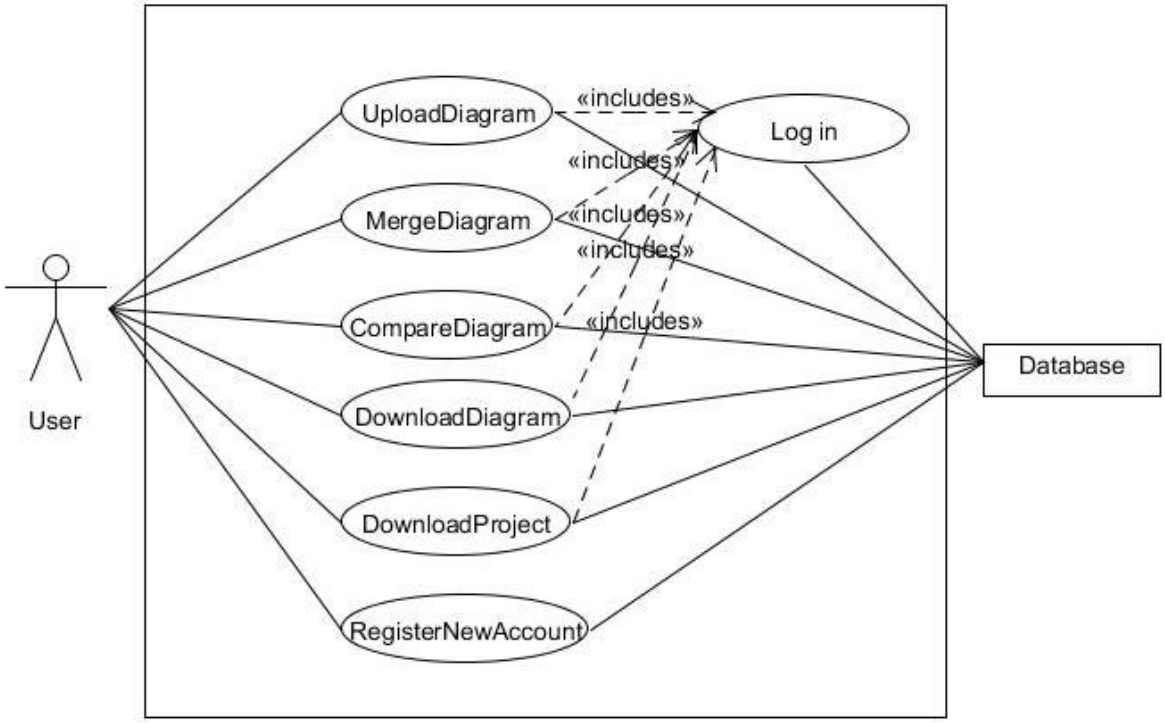


Figure 9: ClubUML System Use Case Diagram

RegisterNewAccount Use Case

Brief Description

This use case describes the account registration process for ClubUML.

Actors

User: The user is the actor who needs to create a new account to access ClubUML.

Preconditions

1. The user does not have an account.
2. The ClubUML webapp is correctly deployed either on a server or on User's local machine
3. The user meets the Java Runtime Environment and browser requirements to run ClubUML.

Flow of events

1. The user navigates to the ClubUML login page.
2. The user clicks the button to register a new account.
3. The user is redirected to the new account creation page.
4. The user enters the desired username, password, security question and answer, and his or her email address (for password recovery).
    - User name must be 1-10 alphanumeric characters
    - Password must be 1-10 alphanumeric characters
    - Two password fields must match
5. The user clicks the register button to request the account be created.
6. The user is redirected to the ClubUML login page.

Alternate Flow

User leaves registration fields blank

If the user does not complete all the fields, the registration page is refreshed but with an error message at the top of the page telling the user to complete all required fields.

User enters invalid entries

If the user enters too many characters into any field or unacceptable special characters, the registration page is refreshed but with an error message at the top of the page telling the user to enter the specified number of characters with no special characters.

Password fields don't match

If the two password fields don't match, the registration page is refreshed but with an error message at the top of the page telling the user to re-enter the same password in both fields.

User requests user name which is already taken

If the user enters a user name which already exists in the database, the registration page is refreshed but with an error message at the top of the page telling the user that the requested user name already exists.

Post-conditions

1. **Valid registration request**
   User arrives at ClubUML login page.
2. **Invalid registration request**
   User is returned to ClubUML registration page with an error message on the page.

Log in Use Case

Brief Description

This use case describes the log in process for a user to access the ClubUML system.

Actors

User: The user is the actor who logs in to his or her account.

Preconditions

1. The user has an account (see RegisterNewAccount for account creation process).
2. The ClubUML webapp is correctly deployed either on a server or on User's local machine
3. The user meets the Java Runtime Environment and browser requirements to run ClubUML.

Flow of events

1. The user navigates to the ClubUML login page.
2. The user enters his or her user name and password in the appropriate boxes.
3. The user clicks the log in button.
4. The user is redirected to the successful log in page.
5. The user clicks the button to continue to ClubUML.
6. The user is redirected to the main ClubUML webapp page.

Alternate Flow

User enters incorrect user name and/or password
If the user does not enter their correct account information, the ClubUML login page is refreshed with an invalid login error message now at the top of the page.

Post-conditions

1. **Correct login credentials entered**
   User arrives at main ClubUML webapp page.
2. **Incorrect login credentials entered**
   User is returned to ClubUML login page with an error message on the page.

UploadDiagram Use Case

Brief Description

This use case describes the process for uploading a UML diagram to ClubUML.

Actors

User: The user is the actor who is uploading a UML diagram.

Preconditions

1. The user follows Login use case flow to access ClubUML.
2. The diagram to be uploaded is either an Ecore diagram or the set of three Papyrus file types.
3. The diagram is either a class diagram (Ecore or Papyrus) or a sequence diagram (Papyrus only).

Flow of events

1. The user selects Ecore or Papyrus file type.
2. The user clicks on the “Choose file” button.
3. The file dialog box appears.
4. The user selects one file to upload from the file dialog box.
5. The user repeats steps 2-4 if there are other files to be uploaded.
6. The user clicks the “Upload” button after files have been selected.
7. After upload is successful, the ClubUML page shows the new diagram in the list of diagrams uploaded to the project.
8. The uploaded diagram image is shown next to the list of files in the project.

Alternate Flow

User selects an invalid file type
If the user selects a file that isn’t the appropriate type, or doesn’t select the three types of Papyrus file (.uml, .notation, .di), the system informs the user that the upload is not valid.
User selects a Papyrus XMI diagram which is not a Class or Sequence diagram
If the user selects an invalid diagram type, the system alerts the user with an error that lists the compatible diagram types (class or sequence).
There is an error parsing the file
If there is an error in the parser, implying a corrupt file, the system alerts the user that there was an error parsing that file and the file may be corrupt.

Post-conditions

1. **Successful Completion**
   The file is successfully added to the ClubUML file list.
2. **Failure Condition**
   No file is uploaded and an error is displayed.

CompareDiagram Use Case

Brief Description

This use case portrays the comparison of two diagrams selected by the user, with support for comments to aid in collaboration.

Actors

User: The user is the actor who invokes the comparison and reviews the report.

Preconditions

1. The user follows Login use case flow to access ClubUML.
2. The user has already uploaded two valid diagrams of the same type (following UploadDiagram flow of events).

Flow of events

1. The user follows Login use case flow to access ClubUML.
2. The user selects two diagrams from the list of diagrams in the project, which are the same type and have been output by the same piece of software to ensure compatibility.
3. The user clicks “Go to Compare” button for the selected diagram of the same type.
4. ClubUML generates a detailed comparison report in a display panel within the web page.
5. The user has the option to click the “save” button to download a PDF report of the comparison (the PDF report does not pop up automatically as in the previous version).
6. The user decides which diagram is preferred and adds a comment under the preferred diagram.
7. The user then clicks the Promote button next to the comment box.
8. The user is returned to the main ClubUML page and the comment can be seen under the promoted diagram.

Alternate Flow

User selects less than two diagrams
If the user selects less than two diagrams the system alerts with a pop-up saying, “*Please select at least two diagrams”*
User selects more than two diagrams
If the user selects more than two diagrams the system alerts with a pop-up saying, *“Please select at most two diagrams”*
No Response from Server
If in process of preparing the comparison report there is no response from the server, then:
1. ClubUML application shall display the message "Network unavailable – try again".
2. The use case ends with a failure condition.

Key Scenarios

If the button “save” has not been clicked, no file would be saved.

Post-conditions

1. **Successful Completion**
   A detailed comparison report will be generated.
2. **Failure Condition**
   No comparison report is generated.

MergeDiagram Use Case

Brief Description

This use case describes a manual merge functionality, in which the user selects which elements to keep from two related diagrams.

Actors

User: The user selects the diagrams to merge and makes decisions on merging.

Preconditions

1. The user follows Login use case flow to access ClubUML.
2. The user has already uploaded two valid diagrams of the same type (following UploadDiagram flow of events).
3. The uploaded diagrams to be merged are from Papyrus (no Ecore merge support).

Basic Flow of Events

1. The user selects two diagrams of the same type (class or sequence), both Papyrus files, to be merged.
2. The user clicks the “merge” button.
3. The webpage shows the two diagrams side by side. Under each diagram, there is a list of:
    a. Classes in diagram A only.
    b. Classes in diagram B only.
    c. Classes in common with diagram A and diagram B.
4. The user can select any pair of classes to merge if they are not already in common with both diagrams, e.g. class Bike in diagram A and class Bicycle in diagram B.
5. The user makes the selection of which elements to be kept in the merged class by checking a box (or some similar type of input) next to each desired element.
6. The user clicks another “merge” button to merge the class.
7. The user will be prompted to resolve any conflicts with associations in the newly merged class by picking which type of association is appropriate in each situation.
8. The user can repeat steps 4-7 to merge other classes which are not in common.
9. When the user is finished merging classes, the user clicks another Merge button to finalize the process and generate the overall merged diagram.
10. The UI displays a new merged diagram with the versions of each element combined.
11. The user can save the merged diagram.
12. The user can export Papyrus files of the merged diagram (see DownloadProject use case).
13. The user can comment on the merged diagram.

14. The user can return to the main ClubUML page after doing any combination of saving the drawing, exporting the merged diagram, or commenting.
15. The merged diagram will appear on the main ClubUML page along with the other diagrams that have been uploaded.

Alternative Flows

Incorrect type of diagrams

If user selects two different types of diagrams, then:

1. An error dialog will be prompted and ask the user to select two diagrams with the same type.
2. The use case returns to basic flow step 1.

Number of diagrams error

If user selects more or less than two diagrams, then:

1. An error dialog will be prompted and ask the user to select more or less diagrams to be merged.
2. The use case returns to basic flow step 1.

Two exact diagrams are selected

If user selects two diagrams that are exactly the same, then

1. A dialog will be prompted notifying user that the two diagrams are the same. Neither selections of merging nor saving of the diagram is allowed.
2. The use case returns to basic flow step 1.

Subflows

No merging

1. The user doesn't make the request to merge diagrams
2. The user can return to the main ClubUML page, and the use case ends.

No saving diagram

1. The user chooses not to save an image of the merged diagram.
2. No saving will be done.

No exporting XMI

1. The user chooses not to export an XMI file of the merged diagram.
2. No exporting will be done.

No commenting

1. The user chooses not to leave any comment.
2. No comment will be added to the diagram.

Key Scenarios

No diagrams will be saved unless the user makes the request.

Post-conditions

1. The merged diagram is saved with comments, if any.
2. No diagrams are lost.

Special Requirements

The merged diagram has the same type as the two original diagrams.

The merged diagram can be merged again with other diagrams, if the user makes the request.

DownloadProject

This use case is for improving the download functionality. The current download functionality gets only a png file of the diagram. User needs to download all uploaded diagrams and the XMI files to re-open the documents in Papyrus at the same time. In order to improve download functionality, it provides with 'download project' functionality that zip /uploads folder and generate .zip file.

DownloadProject Use Case

Brief Description

This use case describes the diagrams downloading process for ClubUML.

Actors

User: The user is the actor who needs to download previously uploaded UML diagrams.

Preconditions

1. The user has an account and has already logged in successfully.
2. Server has existing or uploaded diagram.

Flow of events

1. The user follows Login use case flow to access ClubUML.
2. The user clicks "Download Project" button.
3. ClubUML prompts the user for download location.
4. User enters the location directory.
5. .zip file is saved to the specified location.
6. Project page is redisplayed.
7. The user case ends successfully.

Post-conditions

1. **Successful Completion**
   Diagram downloaded and internal logs updated if any.
2. **Failure Condition**
   Logs updated if any.

System Architecture

The goal of the architecture is to provide an extensible design and design roadmap for the development of the ClubUML software according to the use cases and requirement specifications. The purpose of the tool is to a provide a platform for users to interact on UML Diagrams and give the user the capability for compare, merge and download UML diagrams.

Assumptions and dependencies

The architecture assumes the following:

1. A Linux server will be provided by Northeastern University.
2. Software packages may be remotely administered and installed on this server.
3. The team can program in Java.
4. The team can develop on a MySQL database.
5. The team has Java-based web service experience: JavaScript, JSP, etc.
6. The users will have Eclipse with the Eclipse Modeling Framework (or equivalent) to create UML class diagrams in .Ecore format.
7. The users will have Papyrus tool (an Eclipse plugin) for creation of different UML diagrams.

Decisions, Constraints, and Justifications

The following decisions were made to address the requirements:

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.

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. 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.

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. Besides having built-in Java support, it is also standard in the field and current infrastructure in the deployment environment. No alternatives were considered.

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 Papyrus plugin for Eclipse.

Papyrus is the tool of choice for ClubUML. This an open source tool, which can installed as an Eclipse Plugin. This tool supports UML 2.0 and also supports the following diagrams: Activity Diagram, Class Diagram, Communication Diagram, Component Diagram, Composite Structure Diagram, Deployment Diagram, Package Diagram, Sequence Diagram, State Machine Diagram, and Use Case Diagram. Only Class diagrams and Sequence diagrams will be supported this semester.

The diagrams will be uploaded in the XMI Format or Ecore format.

The Ecore format for class diagrams will be supported for backward compatibility. The UML diagrams generated by Papyrus are stored in the XMI format. Papyrus model generates three files and the User Interface shall add support for uploading multiple files. The files uploaded will saved in a folder with a unique date and time field.

Compare will be only performed on diagrams generated using the same tool.

The tool will support compare between ECORE class diagrams and support compare between class diagrams generated using the Papyrus and compare between sequence diagrams between sequence diagrams generated using Papyrus.

Merge functionality will be only performed on diagrams generated using the Papyrus UML tool

The merge functionality will only supported for Class and Sequence diagrams generated using the Papyrus UML tool. Merging ECORE Class diagrams is out of the scope of this project.

Download functionality will be supported

The User will be able to download a diagram from ClubUML and import it into Eclipse. The import mechanism is beyond the scope of this project.

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 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 Binary Large Object (BLOB) data like some other database applications.

Servlets will be used as the controller mechanism.

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.

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 Java Database Connectivity (JDBC) and did not want to introduce another learning curve by adding an additional library.

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. This decision is mainly based on the first milestone of the software and subject to change.

The architecture will be a layered architecture

The basic model of the layered architecture is shown in Figure 10:

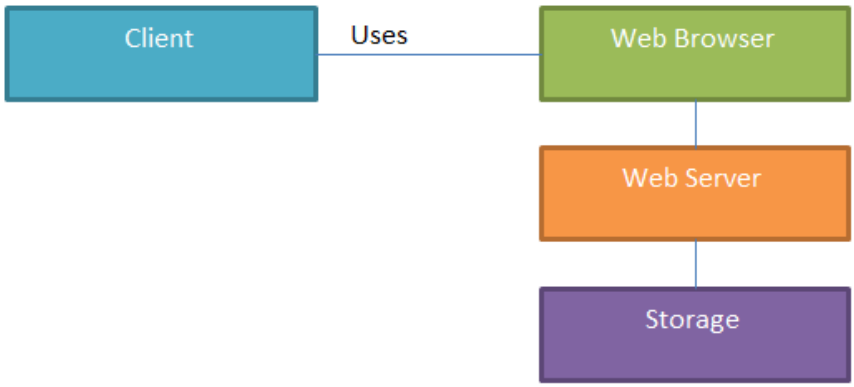


Figure 10: Layered Architecture Diagram

Architectural Mechanisms

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.

2 - Database

All users and project data will either be stored in the database, or have 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.

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.

4 – High-level Components

This consists of a series of high-level components as discussed in the next section. The main goal is to provide a level of abstraction for the developer's code to compare and possibly merge the UML diagrams. It also caters for facilitating any extension during the next iteration of the software. The other goal is to isolate the development teams' tasks and promote a low coupled design mindset.

5 – UML Diagram Parsing

The diagrams will be parsed using a standard SAX Parser. The Elements and its corresponding attributes will be saved in a list. The UML Parser will then place each element based on its UML attribute.

6 – Diagram Visualization

UMLGraph and GraphViz are responsible for class diagram visualization. The upload code will parse the .ecore/.uml and .notation file(s) and create a simple .java file which will be the input to UMLGraph. The output of UMLGraph (known as dot format) will be processed through GraphViz (a generic graph visualization tool, not UML specific), which will produce the diagram as a .png file.

For the extension to integrate sequence diagram in the existing project, the new system will continue with UMLGraph to define the sequence diagram in .pic file format and then visualize by pic2plot graphing tool, which will produce the diagram as a .png or .gif file.

The current limitation of pic2plot support is that it only runs on Windows system. Any extension of support on Linux systems could be implemented by installation of pic2plot for Linux.

7 – 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.

8 – 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 MVC architecture.

9 – 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.

10 – Parsing through Ecore Data Model

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

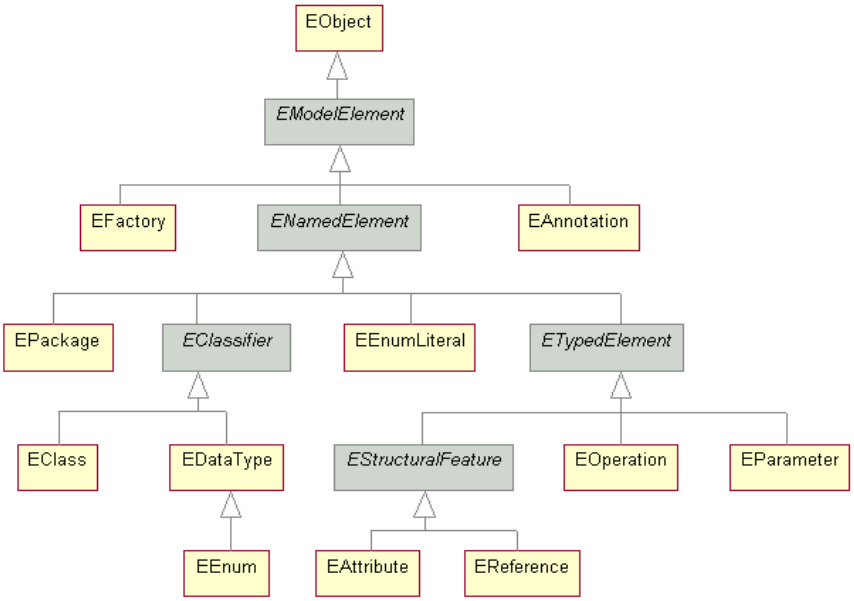


Figure 11: Ecore Data Model

11 – Parsing XMI Model

The SAX Parser will be used for parsing the UML diagram. The XMI Parser Model, as defined here, will be developed.

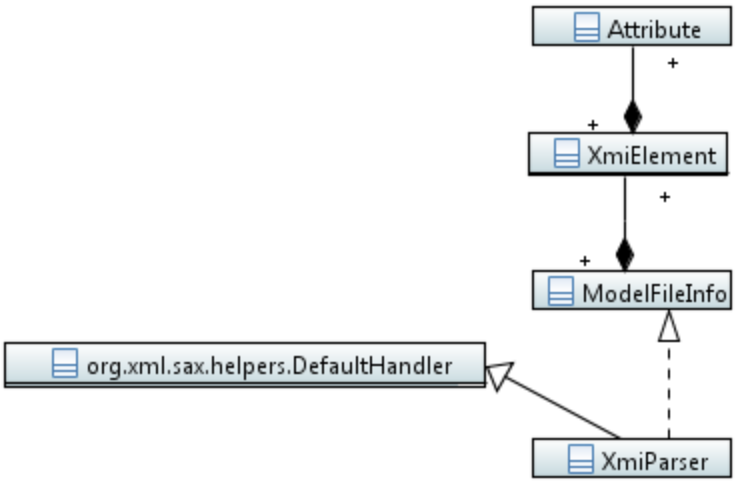


Figure 12: XMI Parser Model

12 – Upload Model

The upload model will implement Parameterized Factory Method Design Pattern. The model below will be developed.

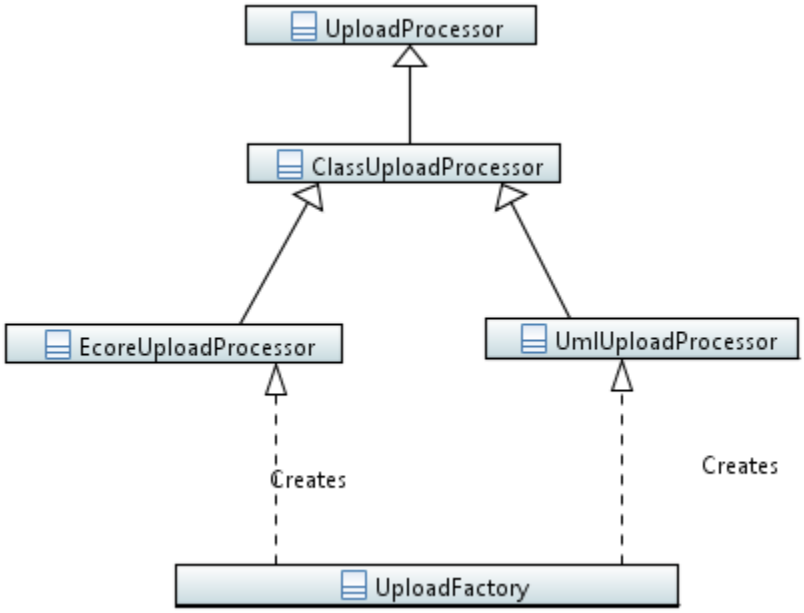


Figure 13: Upload Factory Model

13 – Compare and Merge Model

The compare model will implement the Factory Method design pattern. The model below will be developed.

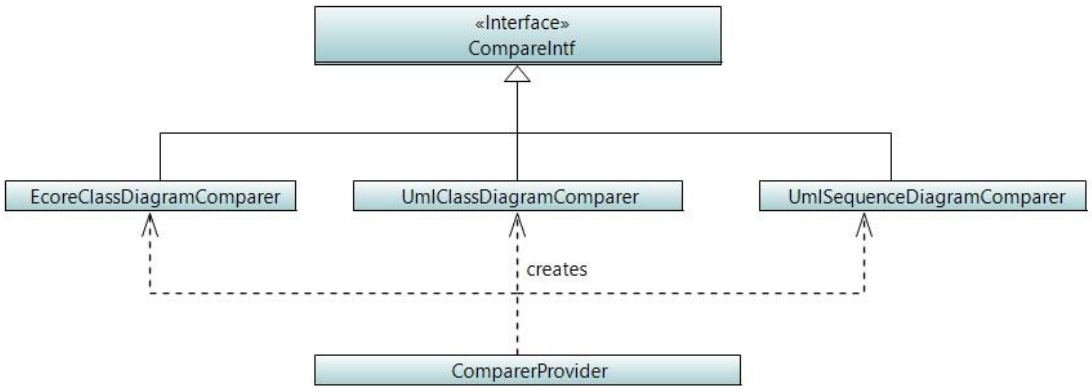


Figure 14: Compare Factory Model

Comparer will also include ClassElement, AttributeElement, OperationElement and RelationshipElement to store information for comparing operations.

14 – Class Diagram Meta Model

A Class Diagram meta model will be defined for use in the comparing and merging architecture. The meta model is a subset of the OMG UML super structure specification, which is too broad for use in this project.

Top level of Meta Model

Figure 15 shows the top level of Meta Model of Class diagram. We can see that Class is compromised by properties, operations, and sometimes NestedClass:

1. Property (covered in more detail in the next section) can be one of two types:
    - The property of an attribute
    - The property of an end of an association
2. Classes are associated by associations, and which may imply that these classes contain related properties.
3. Operation is covered in more detail in the next section.
4. NestedClass is a kind of special class which is nested in another class.

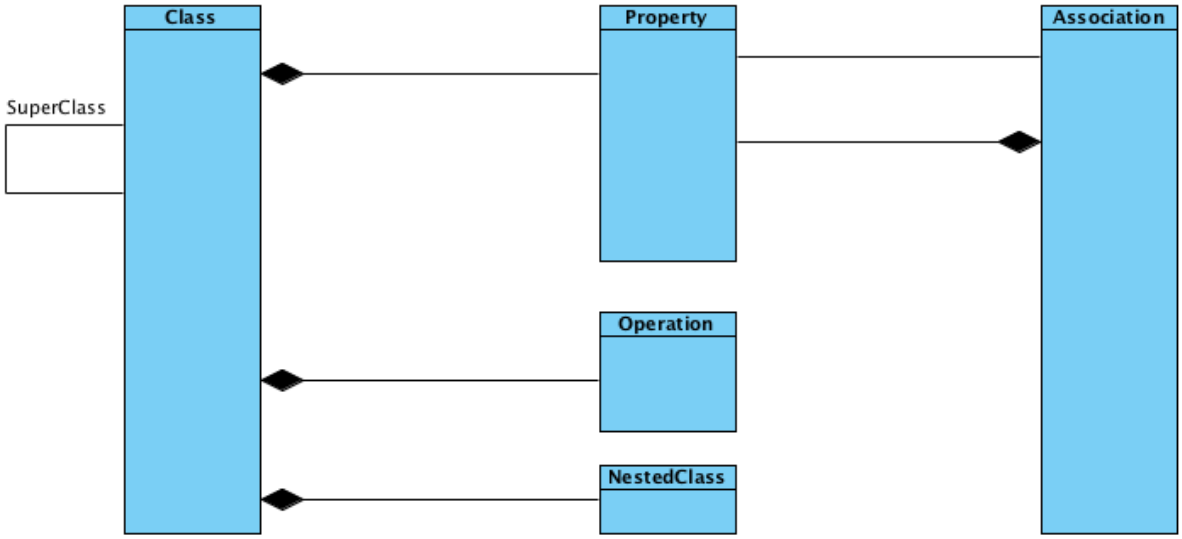


Figure 15: Top-Level Class Meta Model

Lower Level of Meta Model

Operation

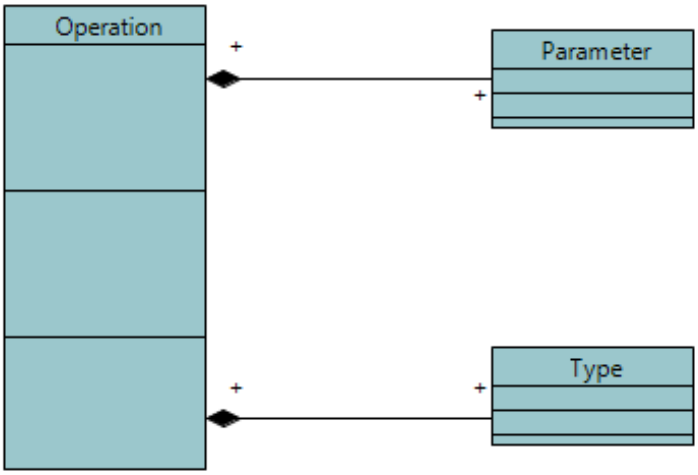


Figure 16: Operation Meta Model

Operation contains two types of elements:

1. Parameter, that is a specification of an argument used to pass information into or out of an invocation of a behavioral feature, that is, an operation.
2. Type, that constrains a type represented by a type element. In our project, the return type and the parameter type can impact one operation.

Association

An association describes a set of tuples whose values refer to typed instances. That means it describes a kind of relationship between at least two ends. Association also contain two types:

1. One ordered type, about this type, one end owns one property that represents this attribute. It is a kind of navigable association.
2. One normal type, about this type, no ends owns property. The association itself owns two ends. This is kind of association represented by association itself, classifier is not owner of this association.

Either type contains a name and multiplicity.

A multiplicity can be separated as a upper value and lower value, as shown in Figure 17.

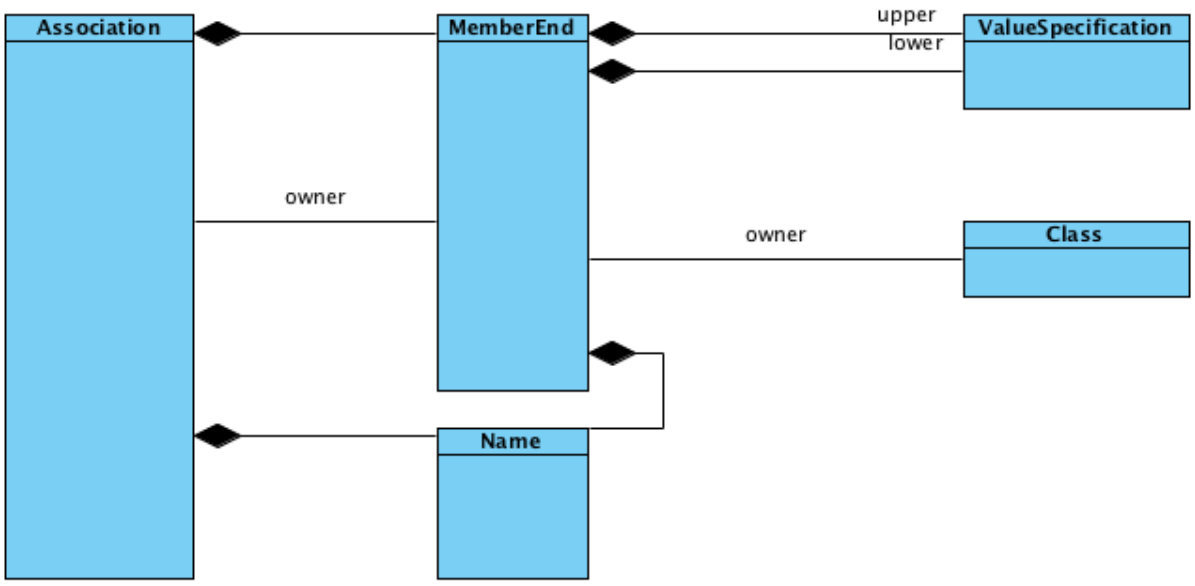


Figure 17: Association Meta Model

Property

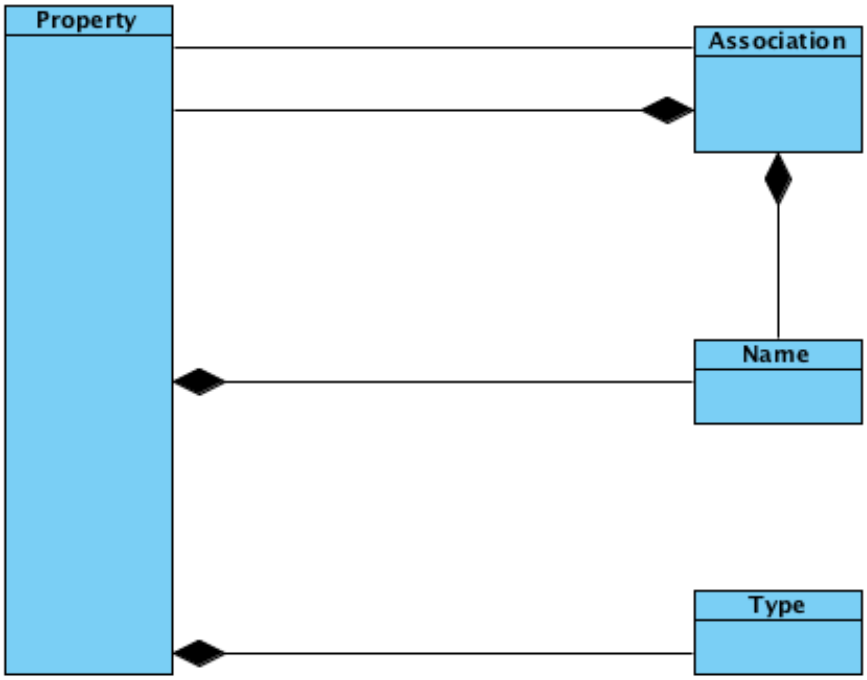


Figure 18: Property Meta Model

A property related to a classifier by ownedAttribute represents an attribute, and it may also represent an association end. It relates an instance of the class to a value (or collection of values) of the type of the attribute.

A property related to an Association by memberEnd or its specializations represents an end of the association. The type of property is the type of the end of the association.

Layers of Architectural Framework

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

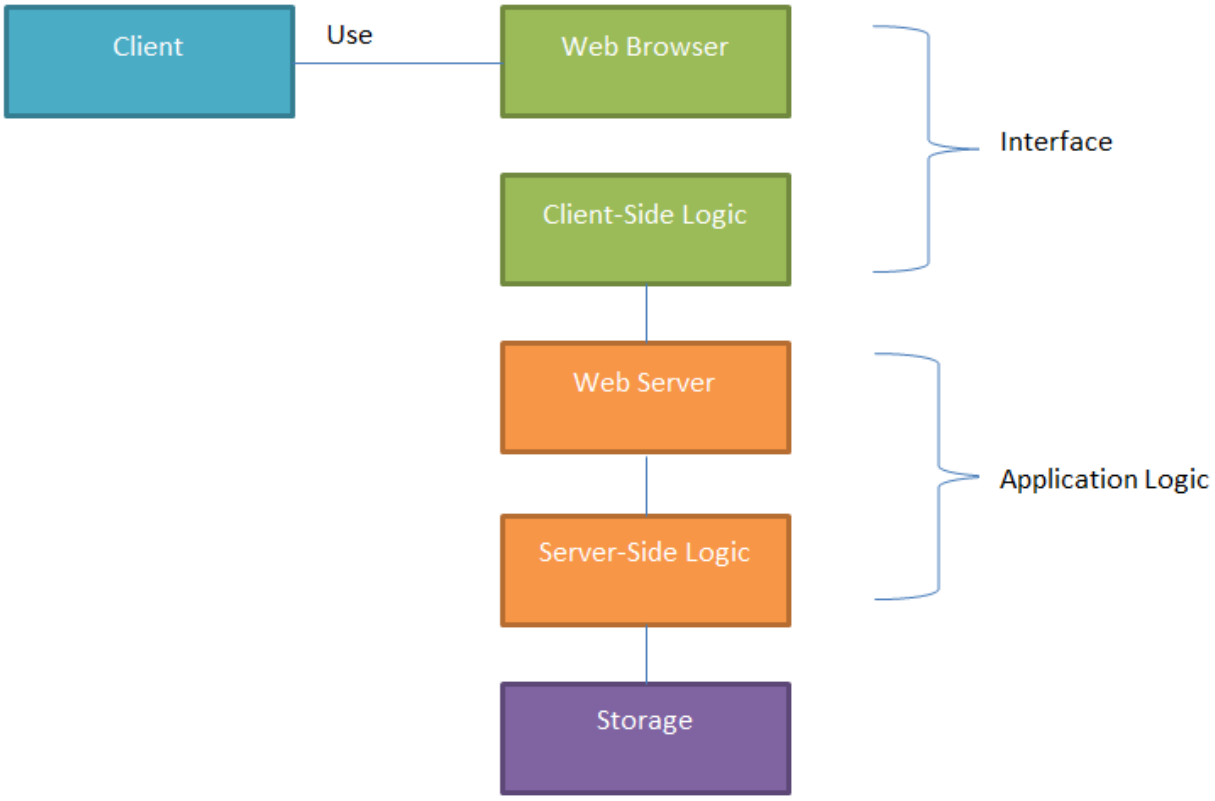


Figure 19: Layer Framework

Design and Implementation

Client-Side

Most of the enhancements to the previous semester's project made use of the existing GUI. However, the merge functionality required a completely new user interface, so this section focuses on the client-side merge development.

Merge Flow from User's Perspective

To elaborate on the flow of the merge use case, the merge process from the user's perspective is broken down into the following steps. Screenshots of the GUI design are shown to illustrate the steps.

Note that steps 3-5 and steps 6-8 can be conducted multiple times, and the user can go through steps 6-8 before steps 3-5, or choose to alternate between those sets of steps, as desired.

1- The user selects the two diagrams for the merge and clicks a "Go to merge" button on the GUI.

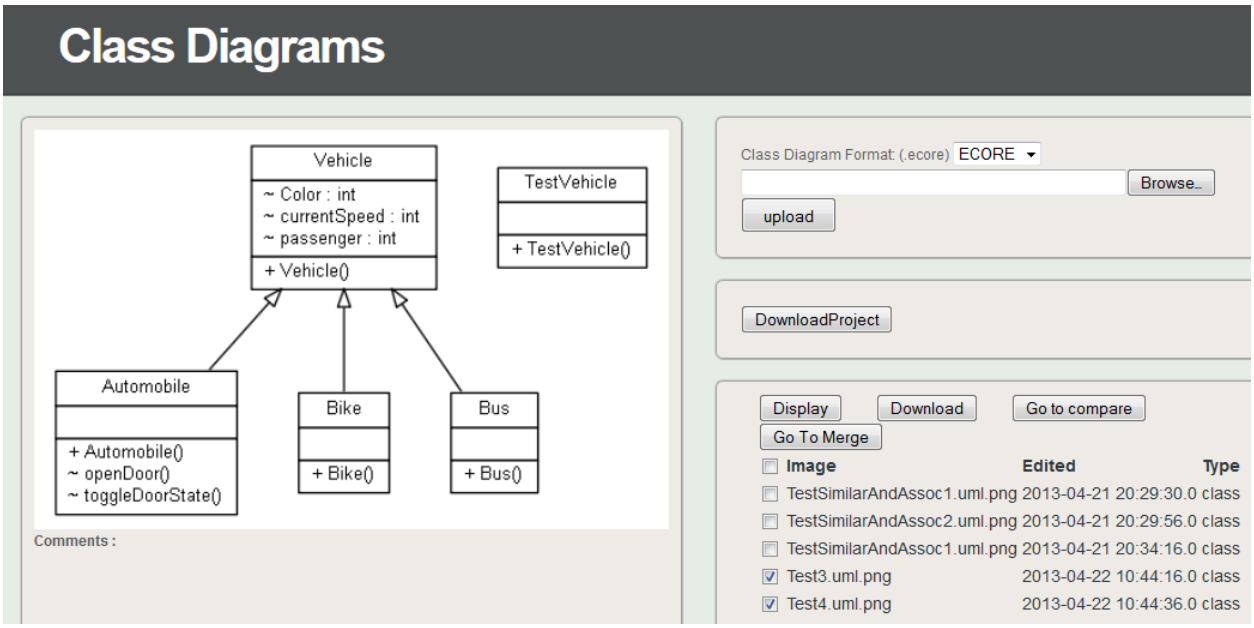


Figure 20: GUI - Selecting Diagrams to Merge

2- The user is taken to the top-level merge page, which shows the two selected diagrams, along with lists for classes unique to diagram 1, classes unique to diagram 2, and classes that the two diagrams have in common. Also, any similarly named classes are shown.

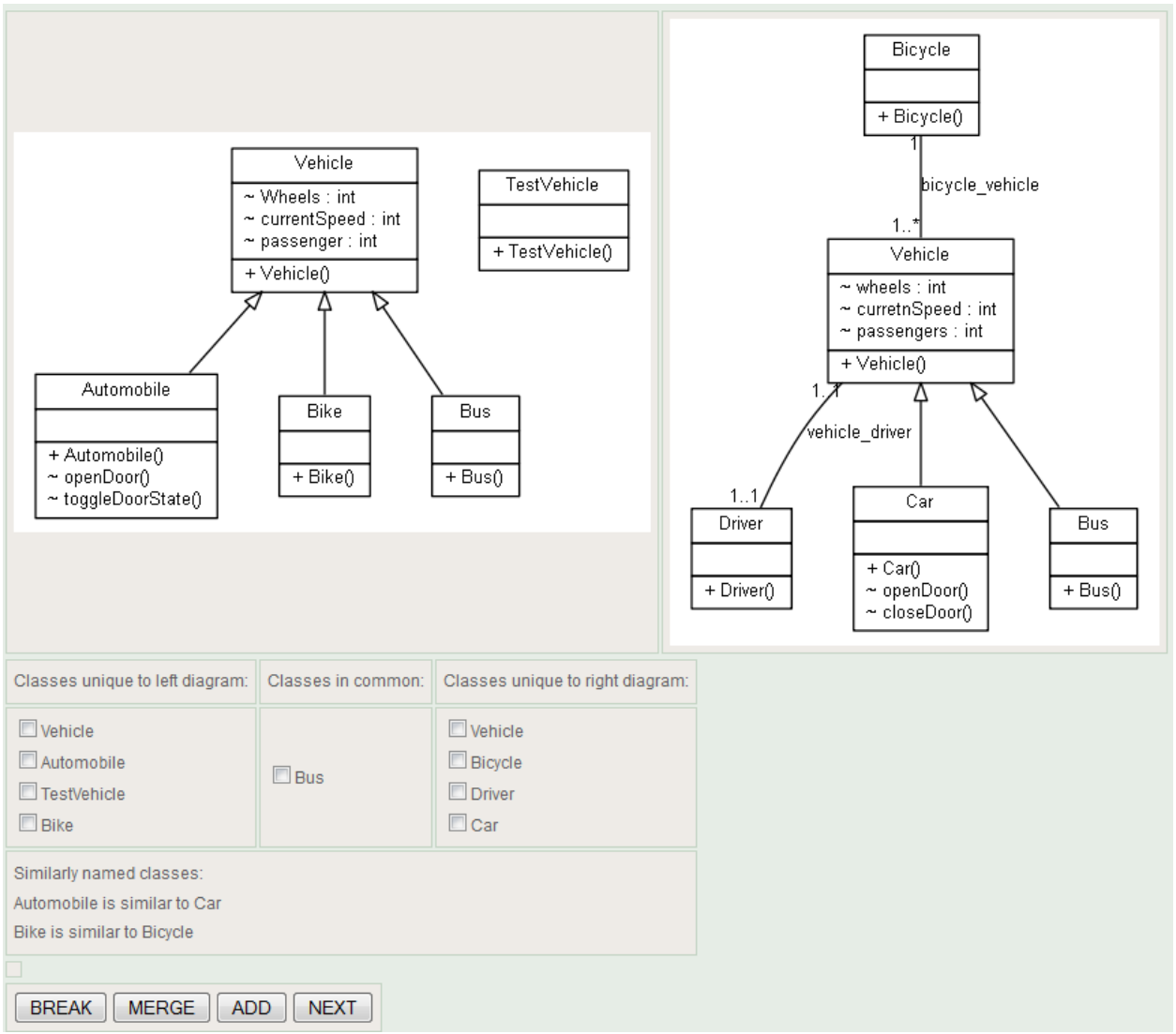


Figure 21: GUI - Select Classes Page

3- If desired, the user picks a set of two classes "unique" to each diagram to consolidate into one. In other words, saying that these two classes that have some differences should actually be the same class. After selecting two classes, the user clicks MERGE.

4- The user is presented with lists of the elements unique to class 1, elements unique to class 2, and elements common to the two classes.

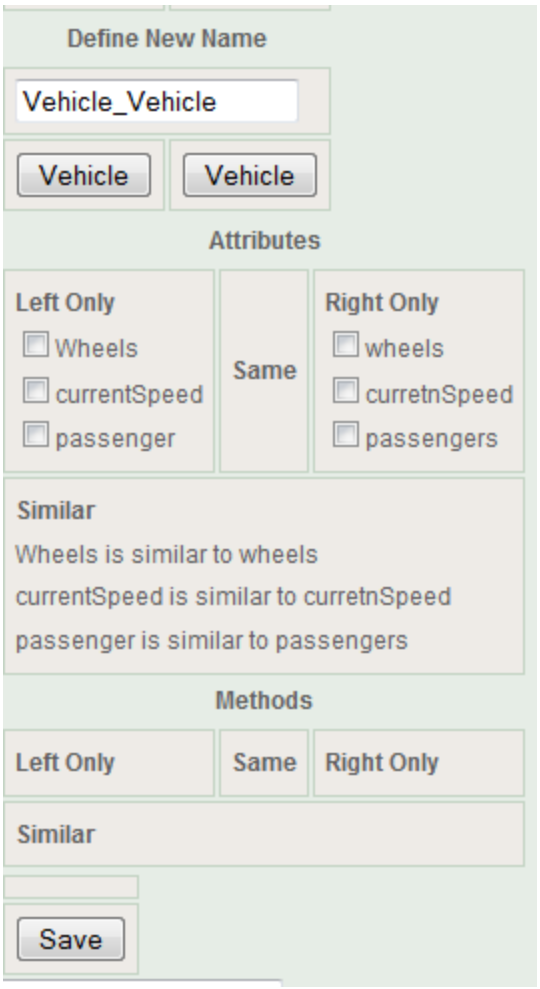


Figure 22: GUI - Merge Classes Page

5- After choosing which class elements to keep in the new merged class, the user submits the selection by clicking Save. The user can repeat steps 3-5 as needed for other classes.
6- If desired, the user chooses to keep a class unique to one diagram, to be included in the merged diagram, by clicking ADD on the select classes page.
7- The UI shows the elements of that class, so the user can choose what to keep out of the selected class.

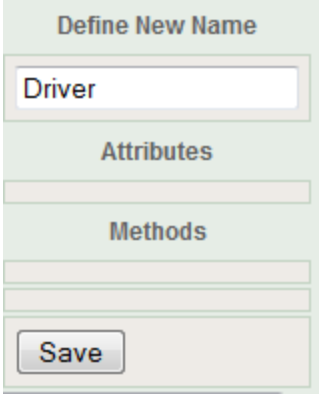


Figure 23: GUI - Add Class Page

8- The user submits selections of what to keep in the added class. The user can repeat steps 6-8 as needed for other classes.
9- The user clicks NEXT on the select classes page, after finishing selecting what classes to merge or add.
10- The UI shows associations that were in place for all the classes which are going to be included in the new merged diagram.

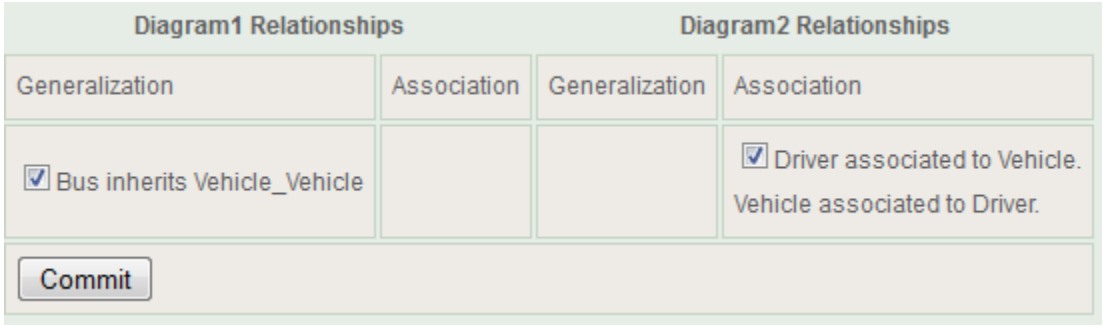


Figure 24: GUI - Select Associations Page

11- After selecting which associations to keep and which to discard, the user submits the selections by clicking Commit.
12- The merge process is now complete, and the newly merged diagram is shown to the user.

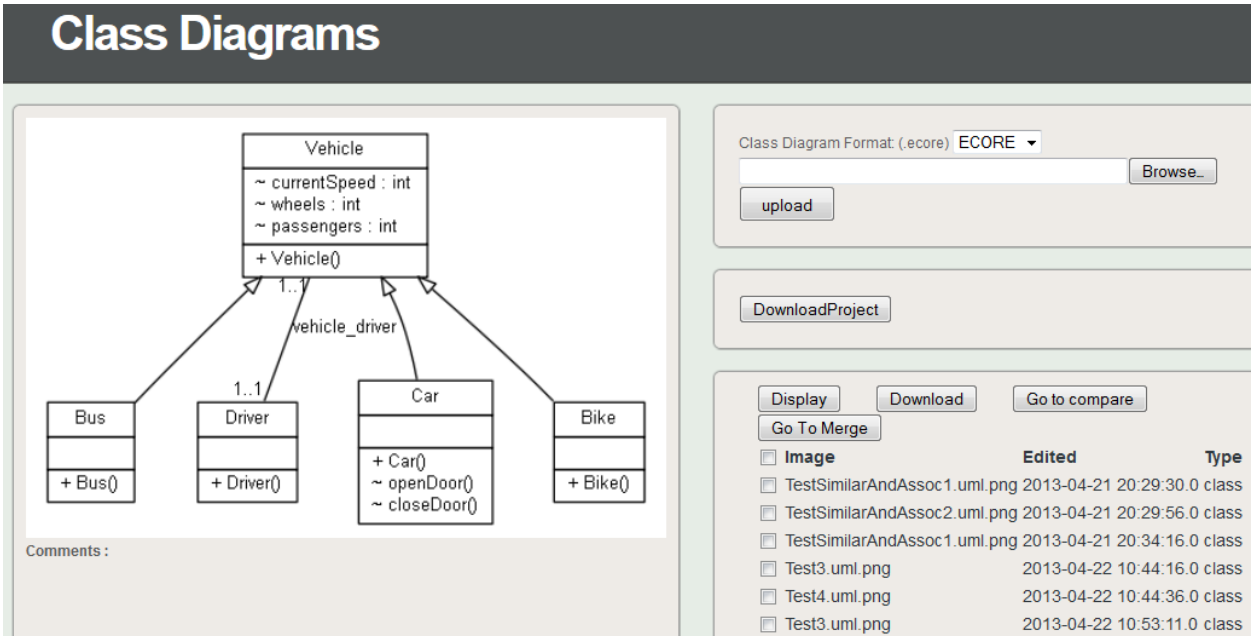


Figure 25: GUI - Displaying Merged Diagram Result

Server-Side

Upload Factory

When we upload a file, we need to do the following

1. Determine the type of the file ECORE or Papyrus. This can be as simple as determining the extension of the files.
2. If the file type is Papyrus, make sure we have uploaded all the necessary files to make the comparison (file_name.uml , file_name.notation and file_name.di)
3. If the file type is Papyrus, we need to determine the models that are currently present in the files.
4. If the file type is ECORE, we can start processing the file as ECORE is only used for class diagrams and does not require any other files.

A parameterized factory method is used to determine which upload processor object to instantiate and return depending on the extension type of the file(s) passed into the argument. It currently handles Ecore and XMI file types.

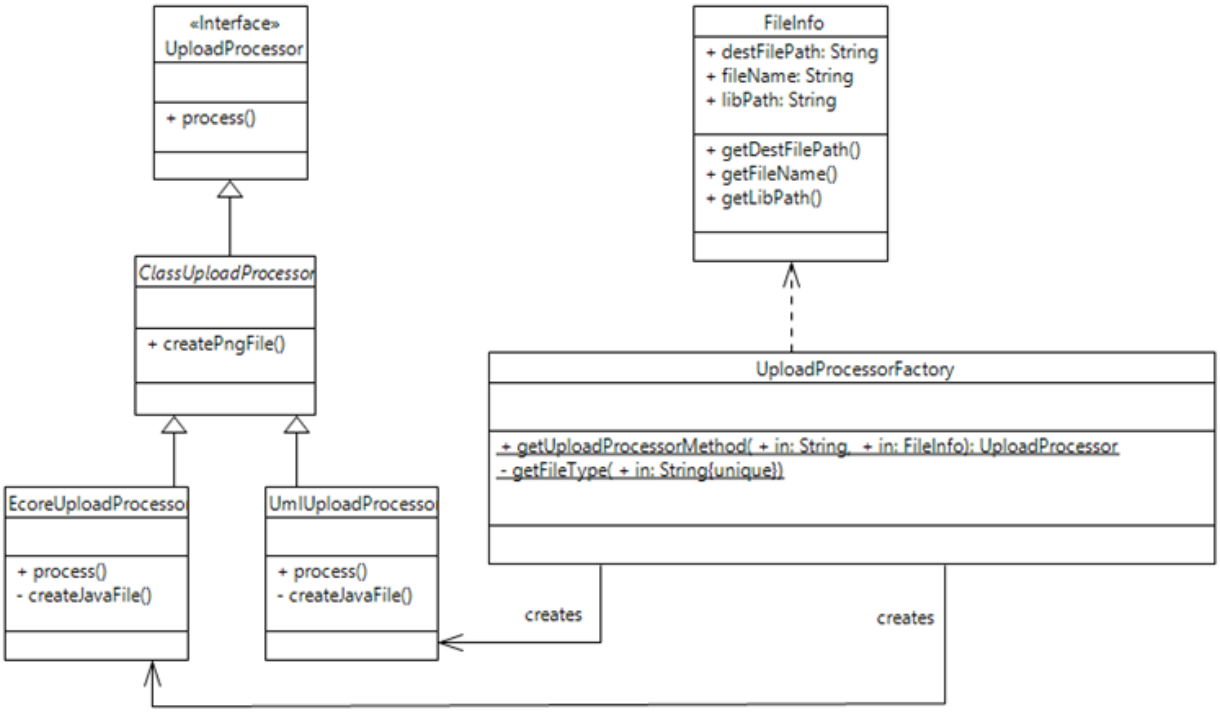


Figure 26: Upload Factory Class Diagram

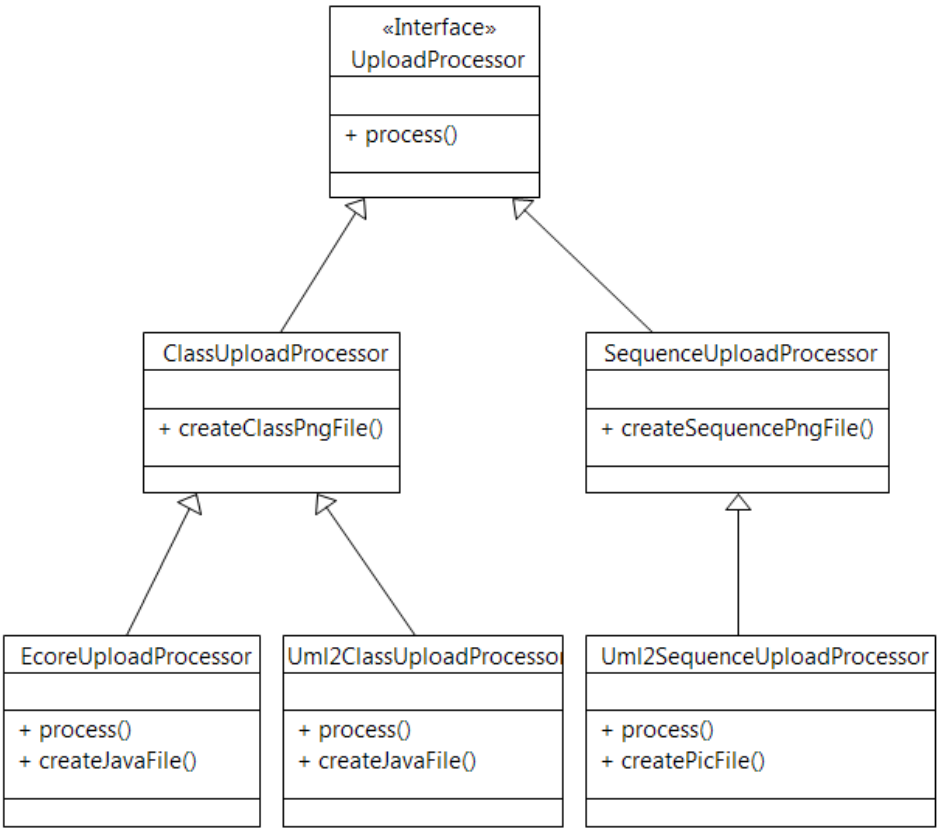


Figure 27: Upload Design Updated to Process Sequence Diagrams

Parameterized factory method is used to determine which upload processor object to instantiate and return depending on the extension type of the file(s) passed into the argument. It currently handles class diagrams and sequence diagram.

Parsing an ECORE File

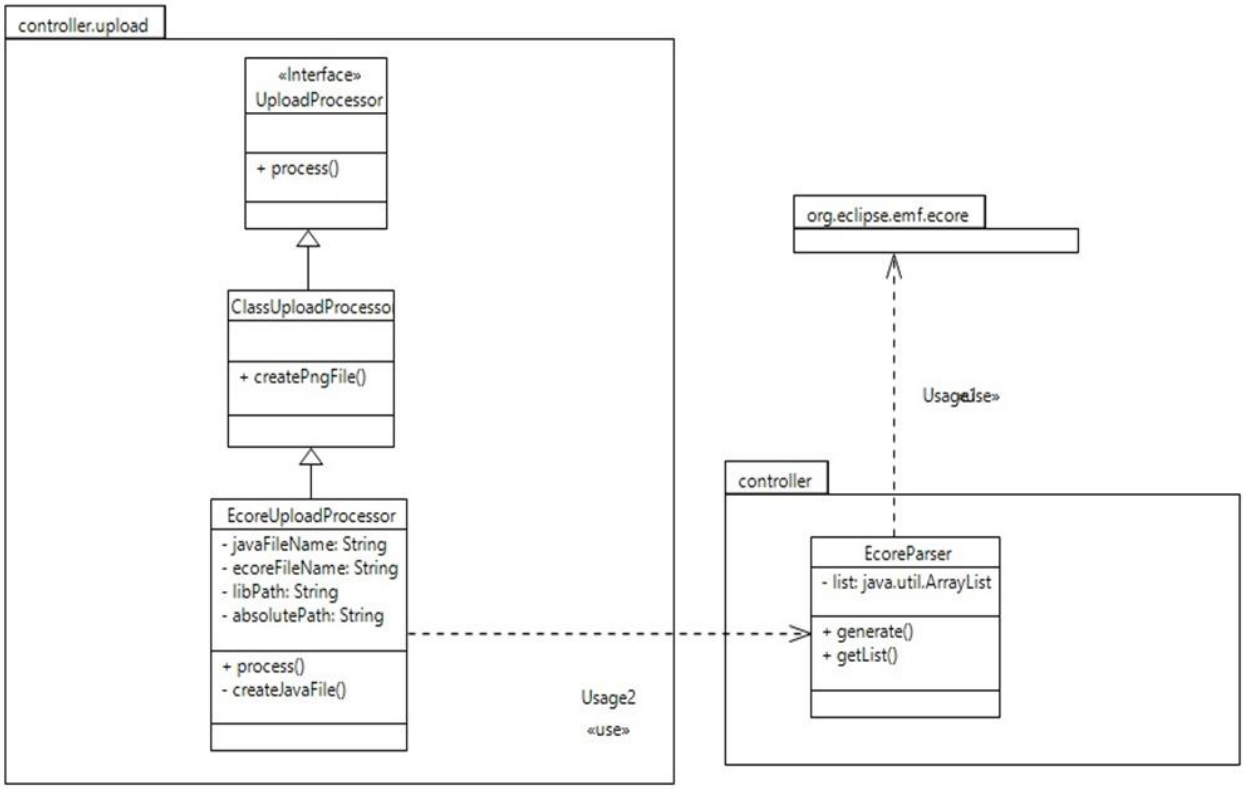


Figure 28: Class Diagram for the Ecore Parser

The implementation for uploading an Ecore file is basically the same (parse ecore file, create java file, create dot file and generate PNG file), but it is now migrated to the EcoreUploadProcess class in controller.upload package. This new class is used by the new Upload Factory Method previous presented.

Parsing a Papyrus Model

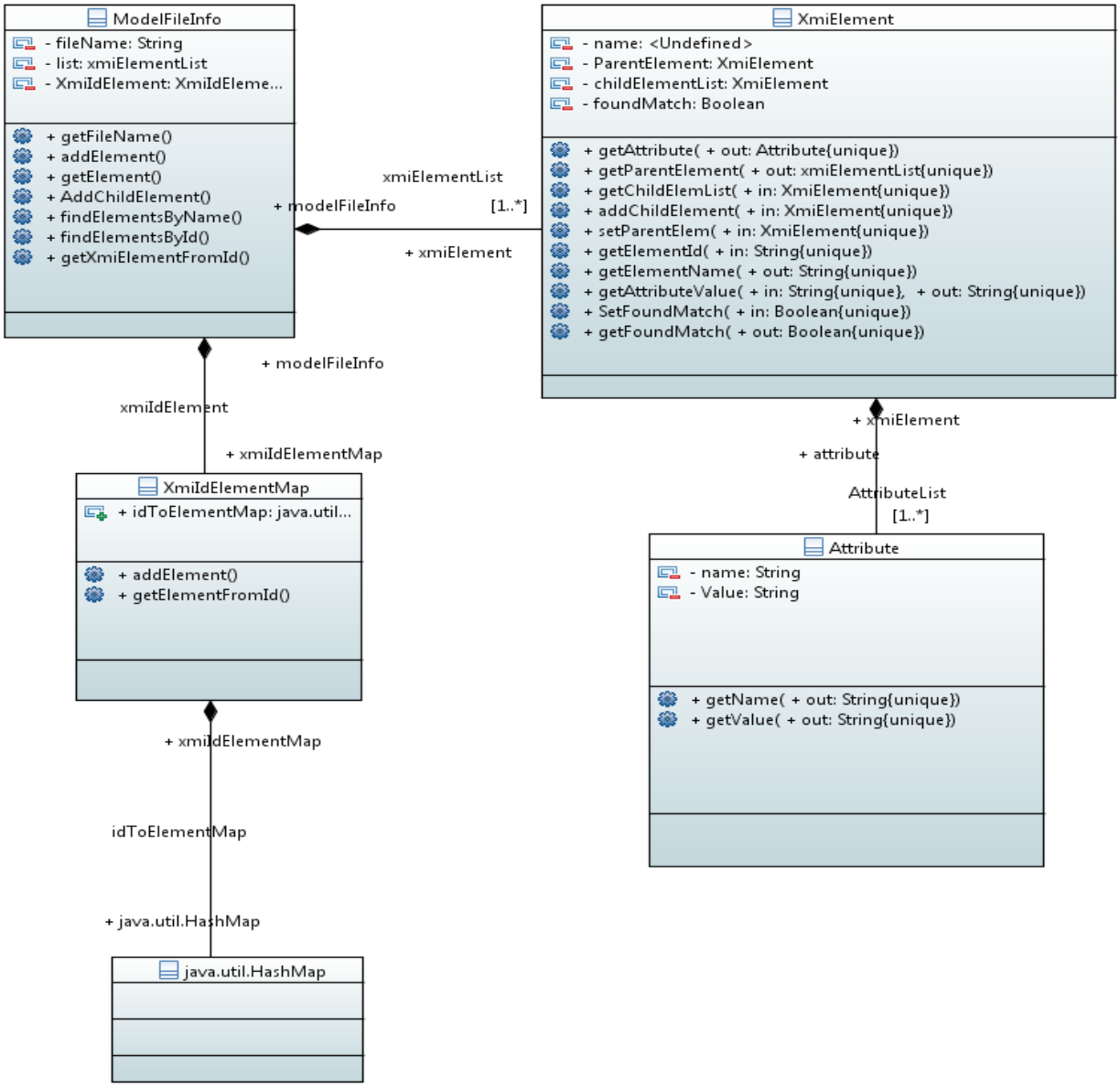


Figure 29: Class Diagram for the XML Parser

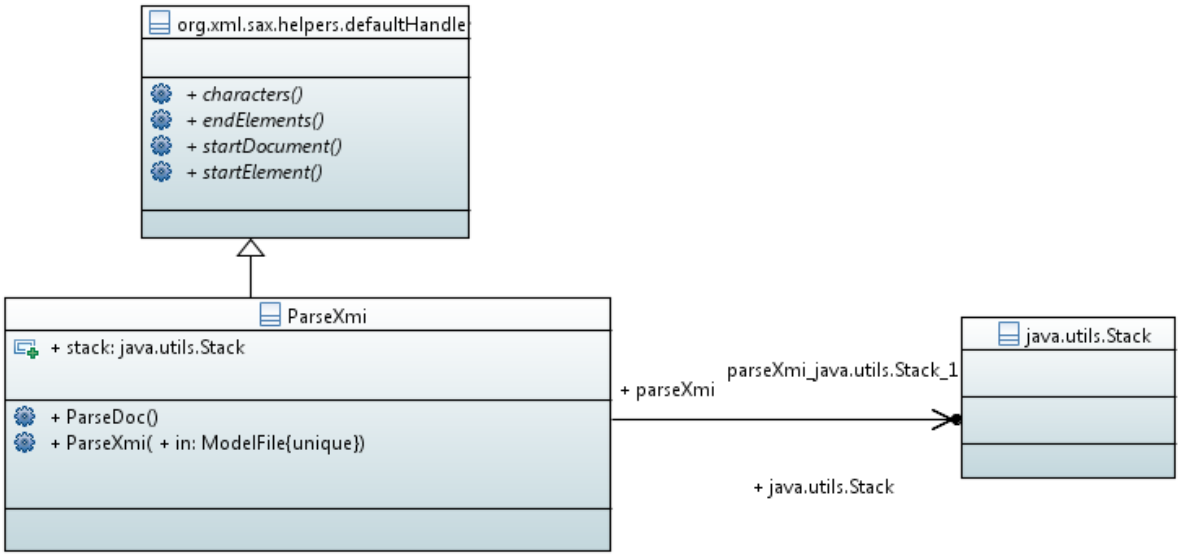


Figure 30: Class Diagram for the XMI Parser.

When a user uploads a Papyrus project (A papyrus project includes 3 files *.uml , *.notation and *.di file).We create an instance of the UmlUploadProcessor (shown in figure 1). We first have to parse the files. The parse mechanism can be found under the uml2parser package of the project. We have divided the package into 5 classes.

1. ParseXmi : This class implements a SAX parser. Please see the documentation on SAXParser for more details. This class opens a XMI files breaks the file into elements and attributes.
2. ModelFileInfo : This class contains a list of all the Elements present in the XMI file. The name of the file the elements are associated with.
3. XmiElement : This class contains a list of all the child elements present in the Element. A list of all the attributes. And parent Element.
4. XmiIdToElement : So most Elements are referenced using Id. So we created a Hashmap of all the ID to elements.
5. Attribute : An Attribute has a name and value.

Above is a class diagram indicating how we can parse a XMI file that is generated by the Papyrus. Once we parsed the XMI file, based on the information from it we can perform any operation e.g., Comparison, Merging, Models supported etc.

Comparing Framework

The class diagram for the initial compare design is shown in Figure 31. This design was developed to support the comparison of elements in Ecore, XMI class diagrams, or XMI sequence diagrams.

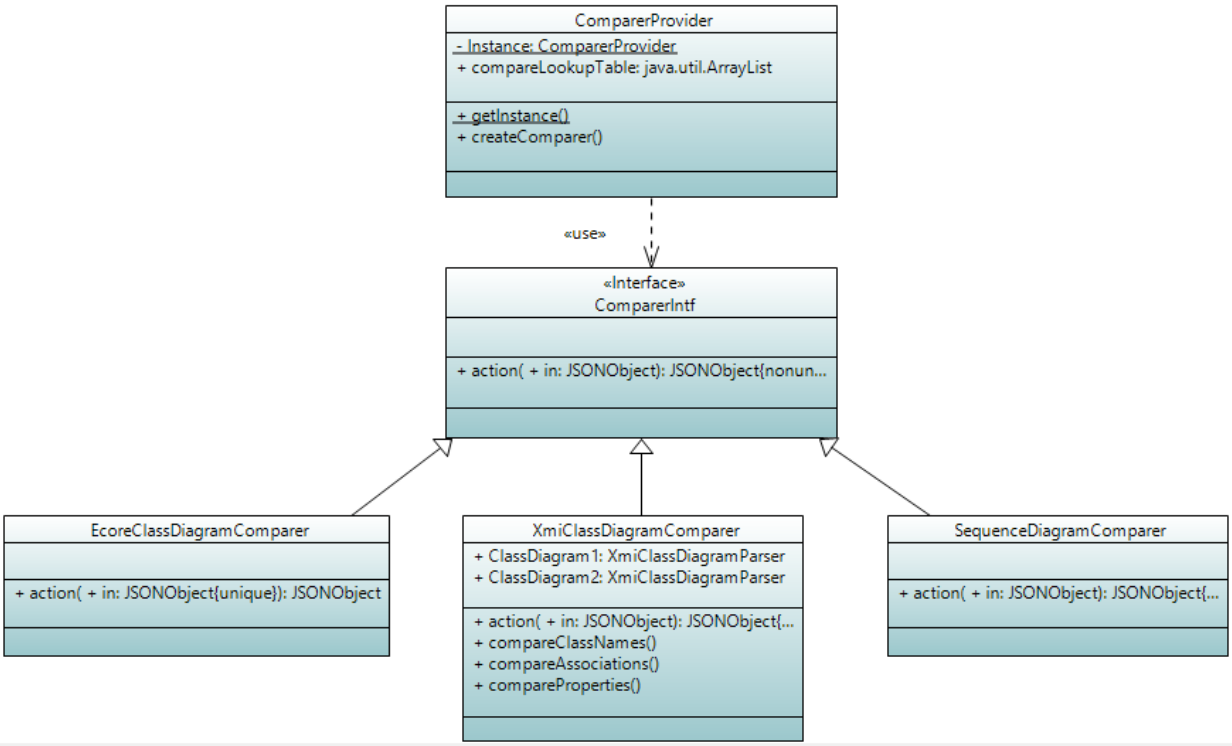


Figure 31: Class Diagram for the Initial Compare Design

CreateComparer method is used to determine which comparer object to instantiate and return depending on diagram model type passed into the argument. It is set up to handle Ecore Class Diagrams, XMI Class Diagrams and XMI Sequence Diagrams. Ultimately, given the time constraints of the project, we focused on only XMI class diagram comparison. EcoreClassDiagramComparer and SequenceDiagramComparer were not implemented, but could be in a future version of the software.

Comparing Papyrus Models

XmiClassDiagramComparer class contains parsers to locate the XmiElements for the compare algorithms, such as Class, Attribute and Association compares. The merge UI will invoke the action() method with a JSON object as the argument for which layer to compare (layer: Class, Attribute and Association), and the action() method will invoke the specific compare method. For example, invoking "Class" request will invoke the compareClassNames() method and return a JSON object that encapsulates information about differences and similarities between the class diagrams' names.

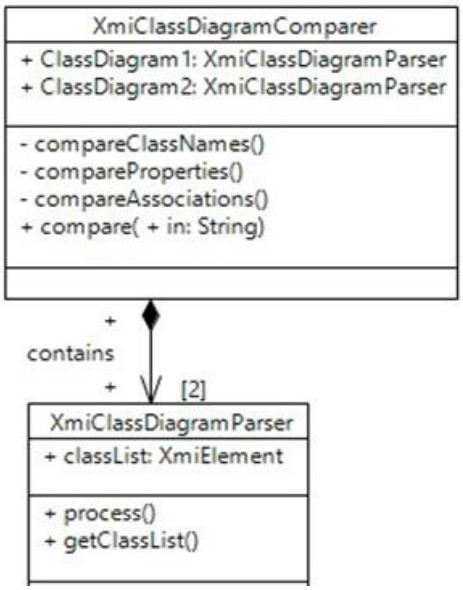


Figure 32: Implementation of the Compare Function

Merging Papyrus Class Diagram Files

When the merging process is completed by the user, new Papyrus XMI files need to be created in order to have a new merged diagram that can be viewed and download. This merge is accomplished by parsing the original XMI files and then constructing a new XMI file using the tags from the original files, changing the ordering or naming of elements as necessary. The merge functionality is within the controller.merge package.

Displaying Papyrus Sequence Diagrams

The Spring2013 version of ClubUML supports uploading, parsing, and displaying images of sequence diagrams. The tools required are:

- UML diagram generation: Papyrus. This is the same tool used to generate class diagrams in the XMI format. A tutorial on creating sequence diagrams with Papyrus can be found here: http://www.eclipse.org/papyrus/usersTutorials/resources/PapyrusTutorial_OnSequenceDiagrams_v0.1_d2010100.pdf
- PNG image deneration:
    - UMLGraph: http://www.umlgraph.org/
    - Pic2Plot: http://www.gnu.org/software/plotutils/manual/en/html_node/pic2plot-Introduction.html
        - For setup instructions see Appendix C: Pic2Plot Setup.

Algorithm Structure Diagram

Once you upload the papyrus files, the system parses it to generate the .pic file and .pic statement, then converts the .pic file into a .png file through the pic2plot program.

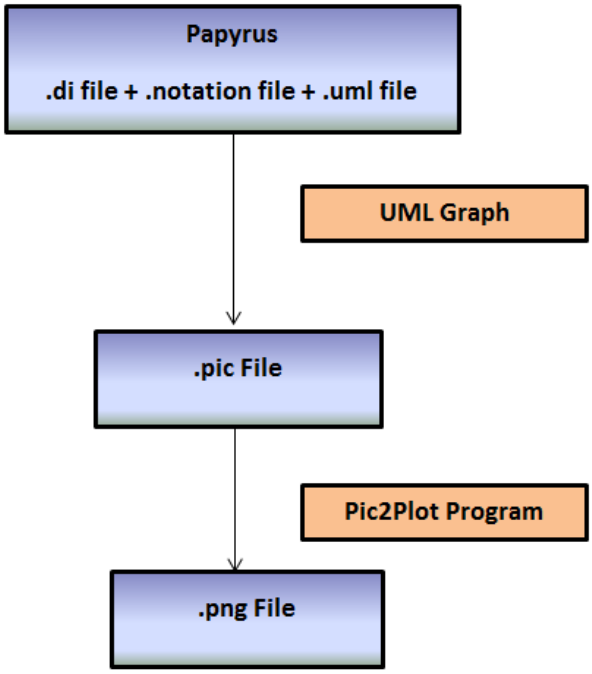


Figure 33: Sequence Diagram Generation Flow

Algorithm Details

Once we have the original papyrus sequence files, we can create a .pic file which defines the sequence diagram in textual format.

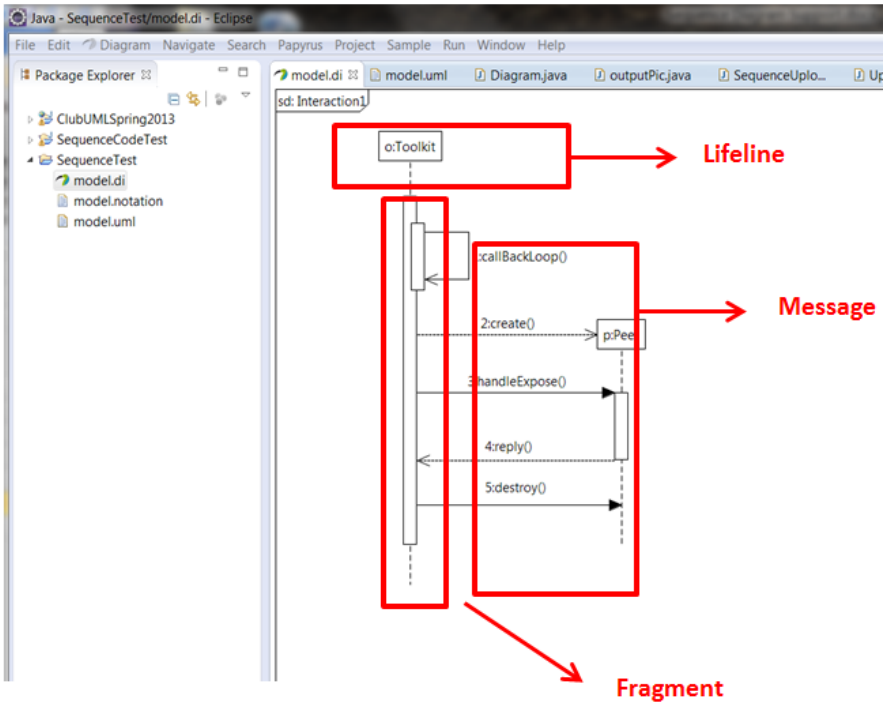
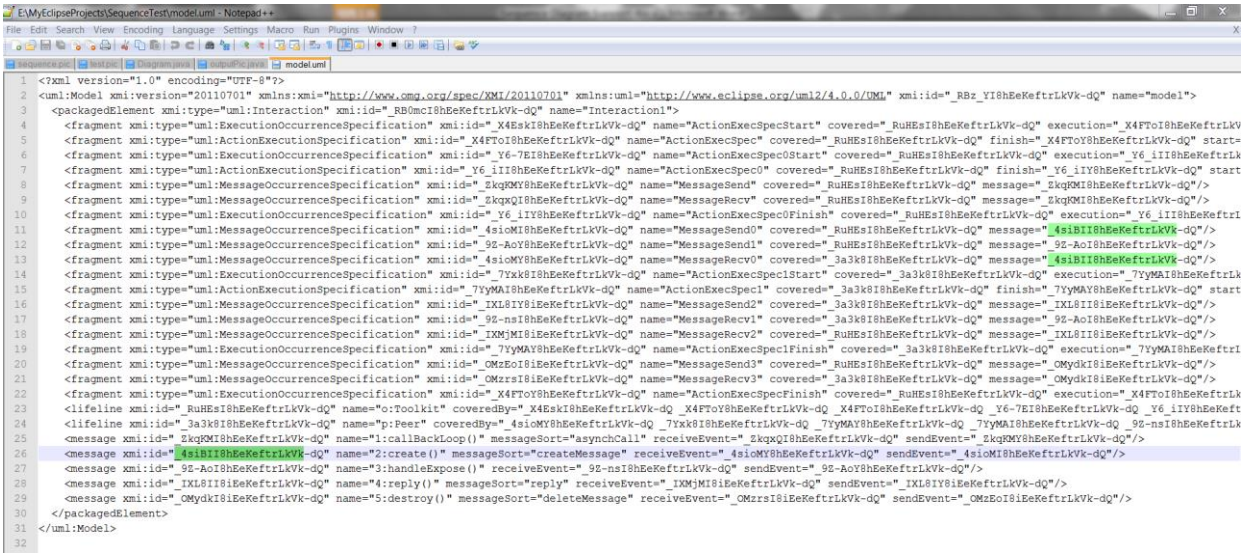


Figure 34: Elements of a Papyrus Sequence Diagram




```
<?xml version="1.0" encoding="UTF-8"?>
<uml:Model xmi:version="20110701" xmlns:xmi="http://www.omg.org/spec/XMI/20110701" xmlns:uml="http://www.eclipse.org/uml2/4.0.0/UML" xmi:id="_RBz_YI8hEeKeftrLkVk-dQ" name="model">
  <packagedElement xmi:type="uml:Interaction" xmi:id="_RB0mcI8hEeKeftrLkVk-dQ" name="Interaction1">
    <fragment xmi:type="uml:ExecutionOccurrenceSpecification" xmi:id="_X4EskI8hEeKeftrLkVk-dQ" name="ActionExecSpecStart" covered="_RuHEsI8hEeKeftrLkVk-dQ" execution="_X4FToI8hEeKeftrLkV
    <fragment xmi:type="uml:ActionExecutionSpecification" xmi:id="_X4FToI8hEeKeftrLkVk-dQ" name="ActionExecSpec" covered="_RuHEsI8hEeKeftrLkVk-dQ" finish="_X4FToY8hEeKeftrLkVk-dQ" start=
    <fragment xmi:type="uml:ExecutionOccurrenceSpecification" xmi:id="_Y6-7EI8hEeKeftrLkVk-dQ" name="ActionExecSpec0Start" covered="_RuHEsI8hEeKeftrLkVk-dQ" execution="_Y6_iII8hEeKeftrLk
    <fragment xmi:type="uml:ActionExecutionSpecification" xmi:id="_Y6_iII8hEeKeftrLkVk-dQ" name="ActionExecSpec0" covered="_RuHEsI8hEeKeftrLkVk-dQ" finish="_Y6_iIY8hEeKeftrLkVk-dQ" start
    <fragment xmi:type="uml:MessageOccurrenceSpecification" xmi:id="_ZkqKMY8hEeKeftrLkVk-dQ" name="MessageSend" covered="_RuHEsI8hEeKeftrLkVk-dQ" message="_ZkqKMI8hEeKeftrLkVk-dQ"/>
    <fragment xmi:type="uml:MessageOccurrenceSpecification" xmi:id="_ZkqxQI8hEeKeftrLkVk-dQ" name="MessageRecv" covered="_RuHEsI8hEeKeftrLkVk-dQ" message="_ZkqKMI8hEeKeftrLkVk-dQ"/>
    <fragment xmi:type="uml:ExecutionOccurrenceSpecification" xmi:id="_Y6_iIY8hEeKeftrLkVk-dQ" name="ActionExecSpec0Finish" covered="_RuHEsI8hEeKeftrLkVk-dQ" execution="_Y6_iII8hEeKeftrI
    <fragment xmi:type="uml:MessageOccurrenceSpecification" xmi:id="_4sioMI8hEeKeftrLkVk-dQ" name="MessageSend0" covered="_RuHEsI8hEeKeftrLkVk-dQ" message="_4siBII8hEeKeftrLkVk-dQ"/>
    <fragment xmi:type="uml:MessageOccurrenceSpecification" xmi:id="_9Z-AoY8hEeKeftrLkVk-dQ" name="MessageSend1" covered="_RuHEsI8hEeKeftrLkVk-dQ" message="_9Z-AoI8hEeKeftrLkVk-dQ"/>
    <fragment xmi:type="uml:MessageOccurrenceSpecification" xmi:id="_4sioMY8hEeKeftrLkVk-dQ" name="MessageRecv0" covered="_3a3k8I8hEeKeftrLkVk-dQ" message="_4siBII8hEeKeftrLkVk-dQ"/>
    <fragment xmi:type="uml:ExecutionOccurrenceSpecification" xmi:id="_7Yxk8I8hEeKeftrLkVk-dQ" name="ActionExecSpec1Start" covered="_3a3k8I8hEeKeftrLkVk-dQ" execution="_7YyMAI8hEeKeftrLk
    <fragment xmi:type="uml:ActionExecutionSpecification" xmi:id="_7YyMAI8hEeKeftrLkVk-dQ" name="ActionExecSpec1" covered="_3a3k8I8hEeKeftrLkVk-dQ" finish="_7YyMAY8hEeKeftrLkVk-dQ" start
    <fragment xmi:type="uml:MessageOccurrenceSpecification" xmi:id="_IXL8IY8iEeKeftrLkVk-dQ" name="MessageSend2" covered="_3a3k8I8hEeKeftrLkVk-dQ" message="_IXL8II8iEeKeftrLkVk-dQ"/>
    <fragment xmi:type="uml:MessageOccurrenceSpecification" xmi:id="_9Z-nsI8hEeKeftrLkVk-dQ" name="MessageRecv1" covered="_3a3k8I8hEeKeftrLkVk-dQ" message="_9Z-AoI8hEeKeftrLkVk-dQ"/>
    <fragment xmi:type="uml:MessageOccurrenceSpecification" xmi:id="_IXMjMI8iEeKeftrLkVk-dQ" name="MessageRecv2" covered="_RuHEsI8hEeKeftrLkVk-dQ" message="_IXL8II8iEeKeftrLkVk-dQ"/>
    <fragment xmi:type="uml:ExecutionOccurrenceSpecification" xmi:id="_7YyMAY8hEeKeftrLkVk-dQ" name="ActionExecSpec1Finish" covered="_3a3k8I8hEeKeftrLkVk-dQ" execution="_7YyMAI8hEeKeftrI
    <fragment xmi:type="uml:MessageOccurrenceSpecification" xmi:id="_OMzEoI8iEeKeftrLkVk-dQ" name="MessageSend3" covered="_RuHEsI8hEeKeftrLkVk-dQ" message="_OMydkI8iEeKeftrLkVk-dQ"/>
    <fragment xmi:type="uml:MessageOccurrenceSpecification" xmi:id="_OMzrsI8iEeKeftrLkVk-dQ" name="MessageRecv3" covered="_3a3k8I8hEeKeftrLkVk-dQ" message="_OMydkI8iEeKeftrLkVk-dQ"/>
    <fragment xmi:type="uml:ExecutionOccurrenceSpecification" xmi:id="_X4FToY8hEeKeftrLkVk-dQ" name="ActionExecSpecFinish" covered="_RuHEsI8hEeKeftrLkVk-dQ" execution="_X4FToI8hEeKeftrLk
    <lifeline xmi:id="_RuHEsI8hEeKeftrLkVk-dQ" name="o:Toolkit" coveredBy="_X4EskI8hEeKeftrLkVk-dQ _X4FToY8hEeKeftrLkVk-dQ _X4FToI8hEeKeftrLkVk-dQ _Y6-7EI8hEeKeftrLkVk-dQ _Y6_iIY8hEeKeft
    <lifeline xmi:id="_3a3k8I8hEeKeftrLkVk-dQ" name="p:Peer" coveredBy="_4sioMY8hEeKeftrLkVk-dQ _7Yxk8I8hEeKeftrLkVk-dQ _7YyMAY8hEeKeftrLkVk-dQ _7YyMAI8hEeKeftrLkVk-dQ _9Z-nsI8hEeKeftrLk
    <message xmi:id="_ZkqKMI8hEeKeftrLkVk-dQ" name="1:callBackLoop()" messageSort="asynchCall" receiveEvent="_ZkqxQI8hEeKeftrLkVk-dQ" sendEvent="_ZkqKMY8hEeKeftrLkVk-dQ"/>
    <message xmi:id="_4siBII8hEeKeftrLkVk-dQ" name="2:create()" messageSort="createMessage" receiveEvent="_4sioMY8hEeKeftrLkVk-dQ" sendEvent="_4sioMI8hEeKeftrLkVk-dQ"/>
    <message xmi:id="_9Z-AoI8hEeKeftrLkVk-dQ" name="3:handleExpose()" receiveEvent="_9Z-nsI8hEeKeftrLkVk-dQ" sendEvent="_9Z-AoY8hEeKeftrLkVk-dQ"/>
    <message xmi:id="_IXL8II8iEeKeftrLkVk-dQ" name="4:reply()" messageSort="reply" receiveEvent="_IXMjMI8iEeKeftrLkVk-dQ" sendEvent="_IXL8IY8iEeKeftrLkVk-dQ"/>
    <message xmi:id="_OMydkI8iEeKeftrLkVk-dQ" name="5:destroy()" messageSort="deleteMessage" receiveEvent="_OMzrsI8iEeKeftrLkVk-dQ" sendEvent="_OMzEoI8iEeKeftrLkVk-dQ"/>
  </packagedElement>
</uml:Model>
```

Figure 35: Sequence Diagram .uml File

Figure 36 highlights ways in which the Messages, Fragments, and Lifelines of a Papyrus sequence diagram are related within its XMI .uml file.

```
<?xml version="1.0" encoding="UTF-8"?>

<uml:Model xmi:version="20110701" xmlns:xmi="http://www.omg.org/spec/XMI/20110701"
xmlns:uml="http://www.eclipse.org/uml2/4.0.0/UML" xmi:id="_RBz_YI8hEeKeftrLkVk-dQ" name="model">

  <packagedElement xmi:type="uml:Interaction" xmi:id="_RB0mcI8hEeKeftrLkVk-dQ"
name="Interaction1">

    <fragment xmi:type="uml:MessageOccurrenceSpecification" xmi:id="_4sioMI8hEeKeftrLkVk-dQ"
name="MessageSend0" covered="_RuHEsI8hEeKeftrLkVk-dQ" message="_4siBII8hEeKeftrLkVk-dQ"/>

    <fragment xmi:type="uml:MessageOccurrenceSpecification" xmi:id="_9Z-AoY8hEeKeftrLkVk-dQ"
name="MessageSend1" covered="_RuHEsI8hEeKeftrLkVk-dQ" message="_9Z-AoI8hEeKeftrLkVk-dQ"/>

    <fragment xmi:type="uml:MessageOccurrenceSpecification" xmi:id="_4sioMY8hEeKeftrLkVk-dQ"
name="MessageRecv0" covered="_3a3k8I8hEeKeftrLkVk-dQ" message="_4siBII8hEeKeftrLkVk-dQ"/>

    <fragment xmi:type="uml:ExecutionOccurrenceSpecification" xmi:id="_7Yxk8I8hEeKeftrLkVk-dQ"
name="ActionExecSpec1Start" covered="_3a3k8I8hEeKeftrLkVk-dQ" execution="_7YyMAI8hEeKeftrLkVk-
dQ"/>

    <lifeline xmi:id="_RuHEsI8hEeKeftrLkVk-dQ" name="p:Toolkit" coveredBy="_X4EskI8hEeKeftrLkVk-
dQ _X4FToY8hEeKeftrLkVk-dQ _X4FToI8hEeKeftrLkVk-dQ _Y6-7EI8hEeKeftrLkVk-dQ _Y6_iIY8hEeKeftrLkVk-
dQ _Y6_iII8hEeKeftrLkVk-dQ _ZkqKMY8hEeKeftrLkVk-dQ _ZkqxQI8hEeKeftrLkVk-dQ _4sioMI8hEeKeftrLkVk-
dQ _9Z-AoY8hEeKeftrLkVk-dQ _IXMjMI8iEeKeftrLkVk-dQ _OMzEoI8iEeKeftrLkVk-dQ"/>

    <lifeline xmi:id="_3a3k8I8hEeKeftrLkVk-dQ" name="p:Peer" coveredBy="_4sioMY8hEeKeftrLkVk-dQ
_7Yxk8I8hEeKeftrLkVk-dQ _7YyMAY8hEeKeftrLkVk-dQ _7YyMAI8hEeKeftrLkVk-dQ _9Z-nsI8hEeKeftrLkVk-dQ
_IXL8IY8iEeKeftrLkVk-dQ _OMzrsI8iEeKeftrLkVk-dQ"/>

    <message xmi:id="_ZkqKMI8hEeKeftrLkVk-dQ" name="1:callBackLoop()" messageSort="asynchCall"
receiveEvent="_ZkqxQI8hEeKeftrLkVk-dQ" sendEvent="_ZkqKMY8hEeKeftrLkVk-dQ"/>

    <message xmi:id="_4siBII8hEeKeftrLkVk-dQ" name="2:create()" messageSort="createMessage"
receiveEvent="_4sioMY8hEeKeftrLkVk-dQ" sendEvent="_4sioMI8hEeKeftrLkVk-dQ"/>

  </packagedElement>

</uml:Model>
```

Figure 36: Relationships Between Message, Fragment, Lifeline

Steps to convert from Papyrus files to .pic file:

1. **Define Messages:**

```
<message xmi:id="_ZkqKMI8hEeKeftrLkVk-dQ" name="1:callBackLoop()" messa
<message xmi:id="_4siBII8hEeKeftrLkVk-dQ" name="2:create()" messageSort
<message xmi:id="_9Z-AoI8hEeKeftrLkVk-dQ" name="3:handleExpose()" recei
<message xmi:id="_IXL8II8iEeKeftrLkVk-dQ" name="4:reply()" messageSort=
<message xmi:id="_OMydkI8iEeKeftrLkVk-dQ" name="5:destroy()" messageSor
```

Figure 37: Messages in XMI File

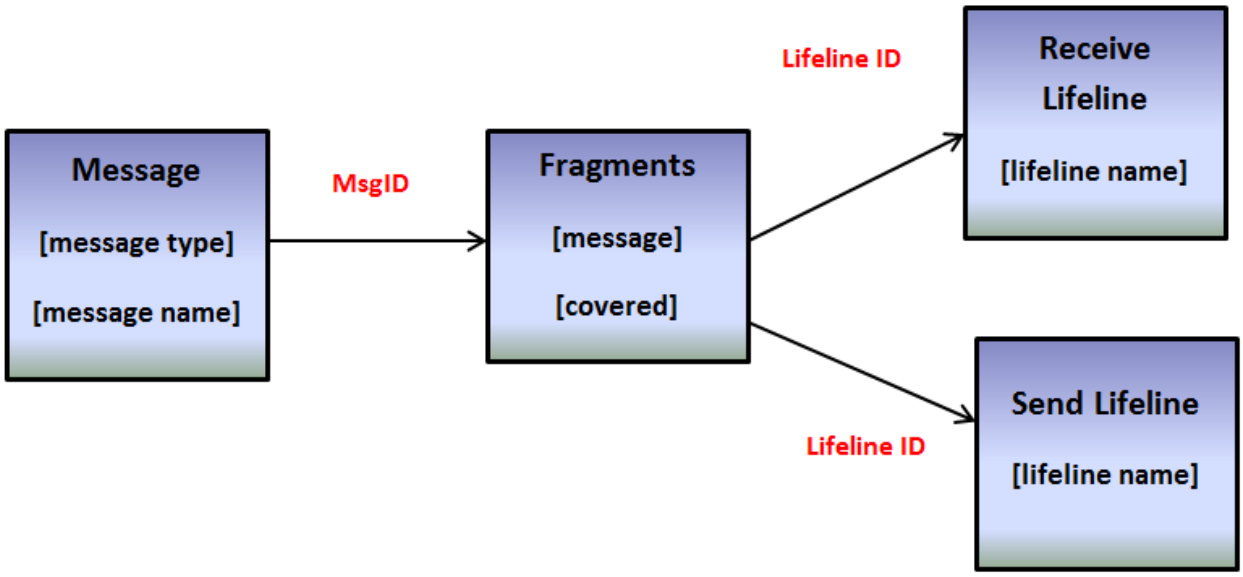


Figure 38: Message Sequence

```
# Message sequences
active(O);
message(O,O,"callbackLoop()");
create_message(O,P,"p:Peer");
message(O,P,"handleExpose()");
active(P);
return_message(P,O,"");
inactive(P);
destroy_message(O,P);
inactive(O);
```

Figure 39: Message Definition in .pic File

2. **Define Objects and PlaceHolder_Objects:**

All the lifelines are defined in .pic files as objects or placeholder_objects.

```
<lifeline xmi:id="_RuHEsI8hEeKeftrLkVk-dQ" name="o:Toolkit"
<lifeline xmi:id="_3a3k8I8hEeKeftrLkVk-dQ" name="p:Peer" cov
```

Figure 40: Lifeline Definition in XMI File

```
# Define the objects
object(O,"o:Toolkit");
placeholder_object(P);
step();
```

Figure 41: Object Definition in .pic File

In order to define one lifeline to be an object or placeholder_object, we have to check the messageSort value. We will use the 2:create() message in Figure 42 as an example.

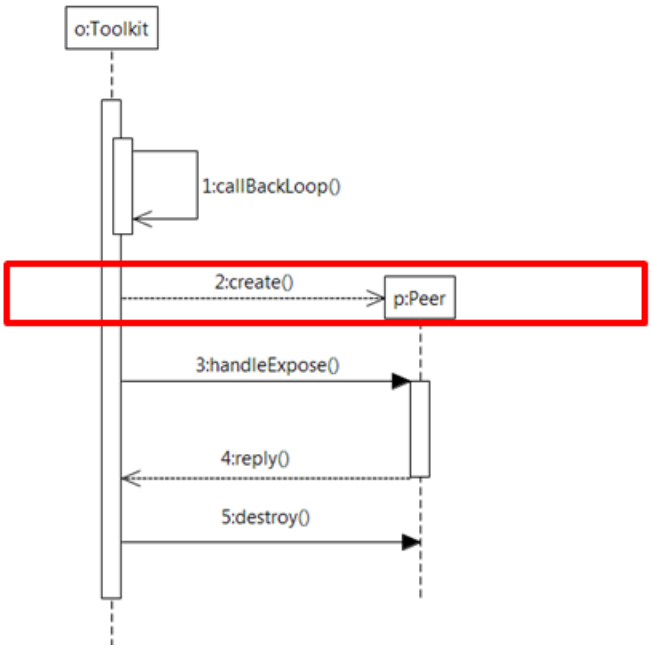


Figure 42: Example create() Message in Papyrus Diagram

```
<message xmi:id="_ZkqKMI8hEeKeftrLkVk-dQ" name="1:callBackLoop()" messageSort="asynchCall" receiveEvent="_ZkqxQI8hEeKeftrLkVk-dQ
<message xmi:id="_4siBII8hEeKeftrLkVk-dQ" name="2:create()" messageSort="createMessage" receiveEvent="_4sioMY8hEeKeftrLkVk-dQ"
```

Figure 43: messageSort value for create()

There is one create() message between o:Toolkit and p:Peer. The message type is createMessage, so the message receiver (p:Peer) should be a placeholder_object instead of an object. The o:Toolkit, on the other hand, is an example of an object.

Implementation Details

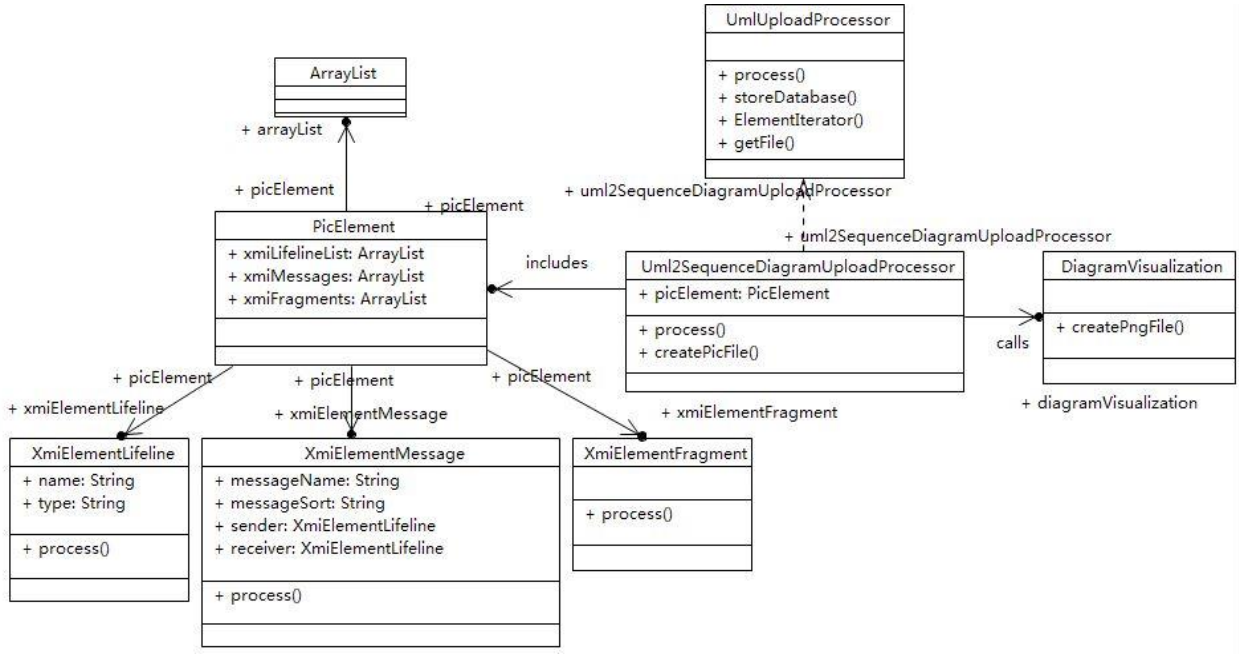


Figure 44: Class Diagram for Sequence Diagram Support

1. Add sequence diagram support in process() method of UmlUploadProcessor

```
if (isSeqDiag) {
    new Uml2SequenceDiagramUploadProcessor().process();
    String image_file_name = "class_diagram_" + umlInfo.getFileName()
            + ".png";
    String folder = diagramPath + "/" + umlInfo.getFileName();
    this.storeDatabase(folder, image_file_name, id);
}
```

2. Call Uml2SequenceDiagramUploadProcessor process() method:
   - Find Active Element List
     Create Fragment xmiElement list
     Create Lifeline xmiElement list
     Create Message xmiElement list
   - Create xmiElementLifeline and add into picElement instance
   - Create xmiElementMessage, set sender lifeline and receiver lifeline and sort message into correct order then add into picElement instance
   - Create xmiElementFragment and add into picElement instance
3. Calls createPicFile() method:
   Pass picElement instance into the method, read it and create .pic statement
4. Calls createPngFile() method of SequencePngFile.java to generate the .png file

Results

Figure 45 shows one of the diagrams used to test sequence diagram support. The diagram is shown as it appears as a Papyrus model opened in Eclipse.

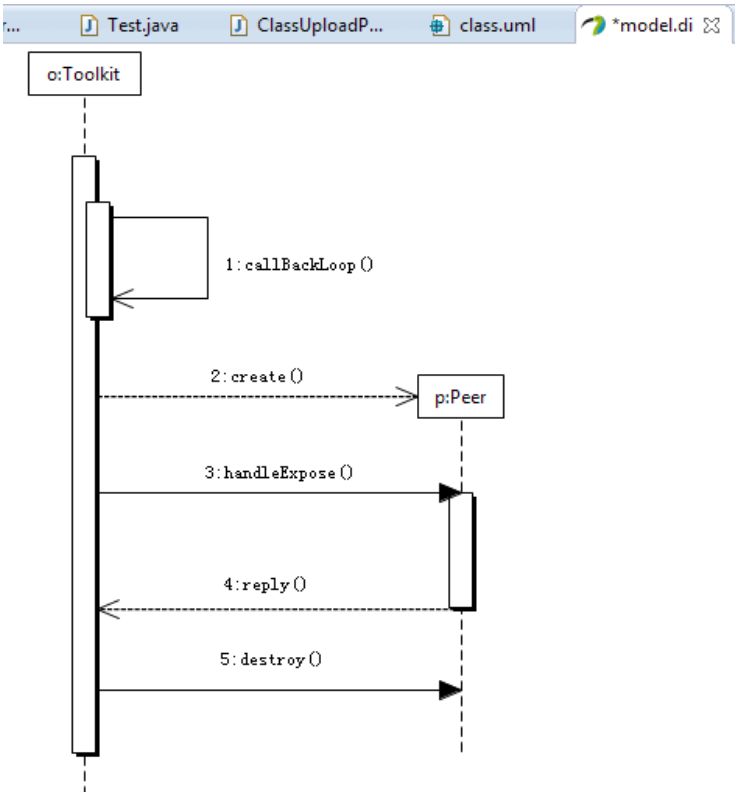


Figure 45: Sequence Diagram Test File in Papyrus

Once uploaded and displayed in ClubUML, the image appears as shown in Figure 46.

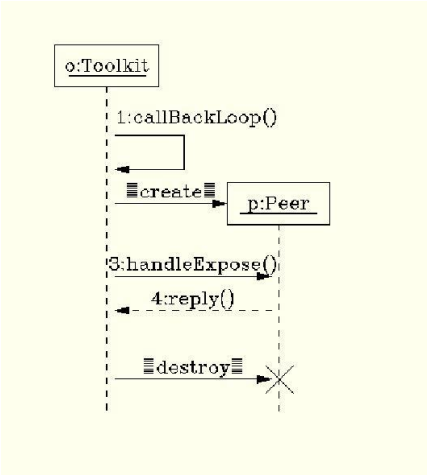


Figure 46: Sequence Diagram rendered in ClubUML

A screenshot of the GUI, with red boxes showing the uploaded diagram and the listing of the uploaded file, is shown in Figure 47.

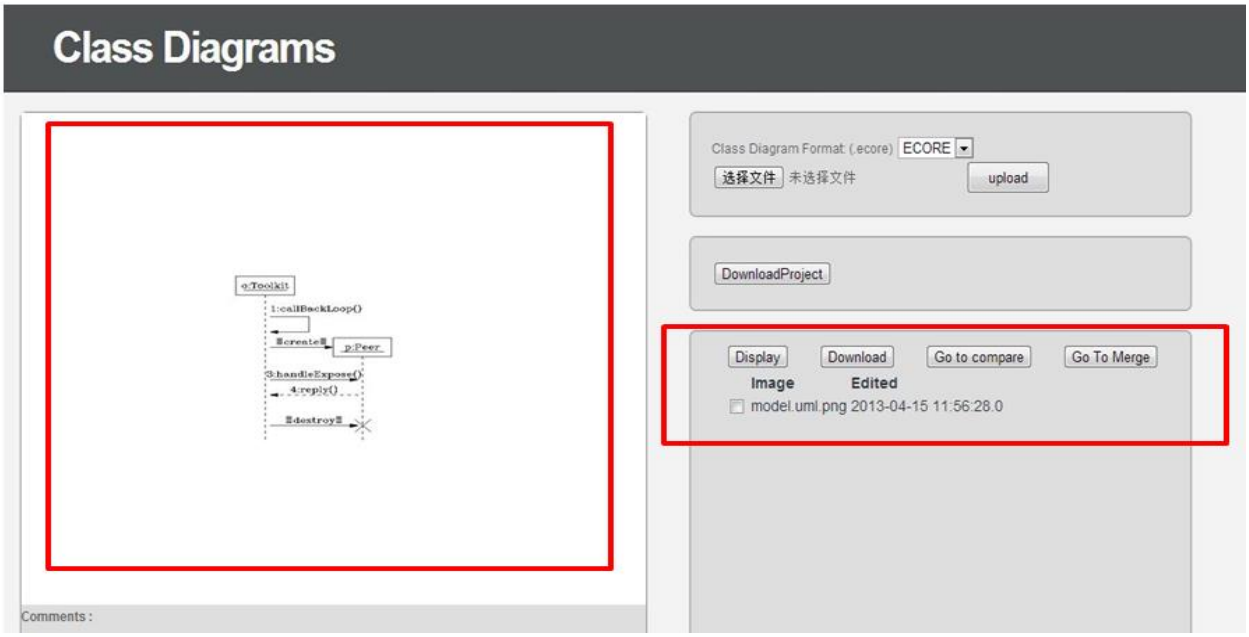


Figure 47: ClubUML GUI with Sequence Diagram

Checking for Similar Names

The old comparing functions from last semester would only find exact matches between sets of Strings. Similarly named classes would now be acknowledged by the comparing functionality. For example, class "Bicycle" would not match "Bike".

We developed functionality to find similar names which can be presented to the user as suggestions for classes or class elements that could possibly be the same or at least related.

The classes developed to check for similar names are in the controller.similaritycheck package and consist of the following four classes, which return true if the check results in finding the two input Strings to be similar.

1. **LowerUpperCheck.java**
   Compares case sensitivity of two names. For example, "className" and "Classname" would be considered similar.
2. **SpellCheck.java**
   Checks for potential spelling mistakes in a couple of different ways. The first check is to see if there is just one character different between two strings. The second check is for two letters transposed, e.g. "bicycle" vs. "bicycel".
3. **PluralCheck.java**
   Compares two strings to see if they are the same except for one being singular and the other being plural. Since this is not the core of our project and there are many special cases, now it just works for most words. Support for less common plural forms of words would be useful as a future update but is not considered a high priority for this semester.
4. **SynonymCheck.java**

Uses a public API from bighugelabs.com to get a list of the synonyms of one string, to compare to the other input string. To use the API, we need:
An API key which is: eb23eace35633b1274abdfeabc1b4753
The account username: guo.do@husky.neu.edu
The account password: umlclub2013

The top-level API for the similar names check is SimilarityCheck.doSimilarityCheck() with two Strings as input arguments. It will return true if any of the checks return true, false otherwise. Figure 48 shows the class diagram for the entire similarity check package. Figure 49 shows the sequence diagram of a user calling the public method SimilarityCheck.doSimilarityCheck() and the sequence of messages as it conducts the various tests.

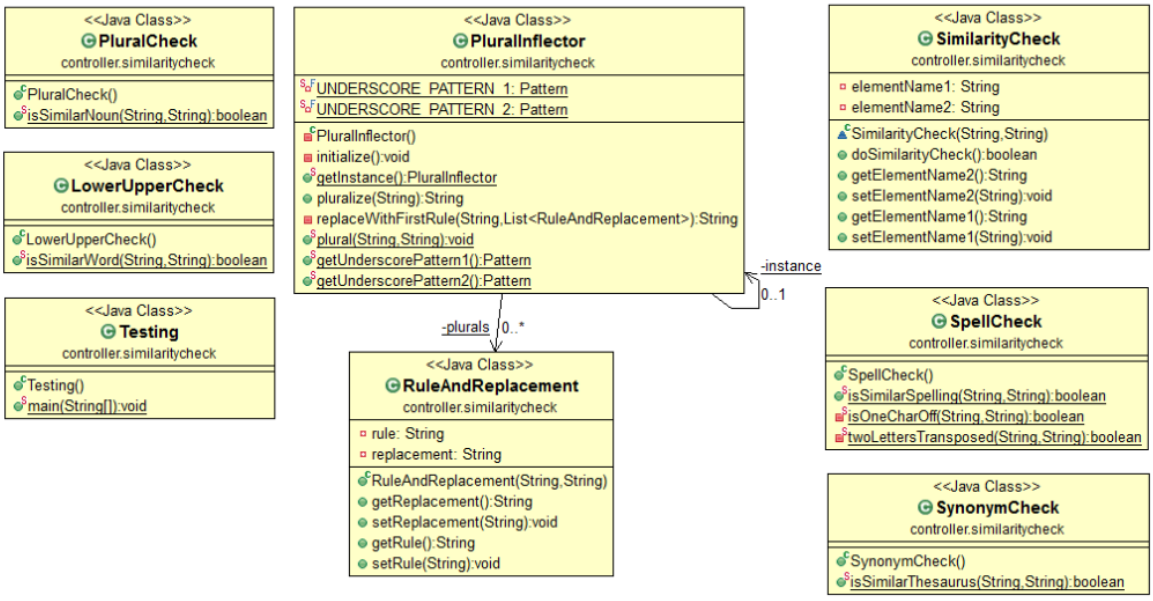


Figure 48: Similarity Package Class Diagram

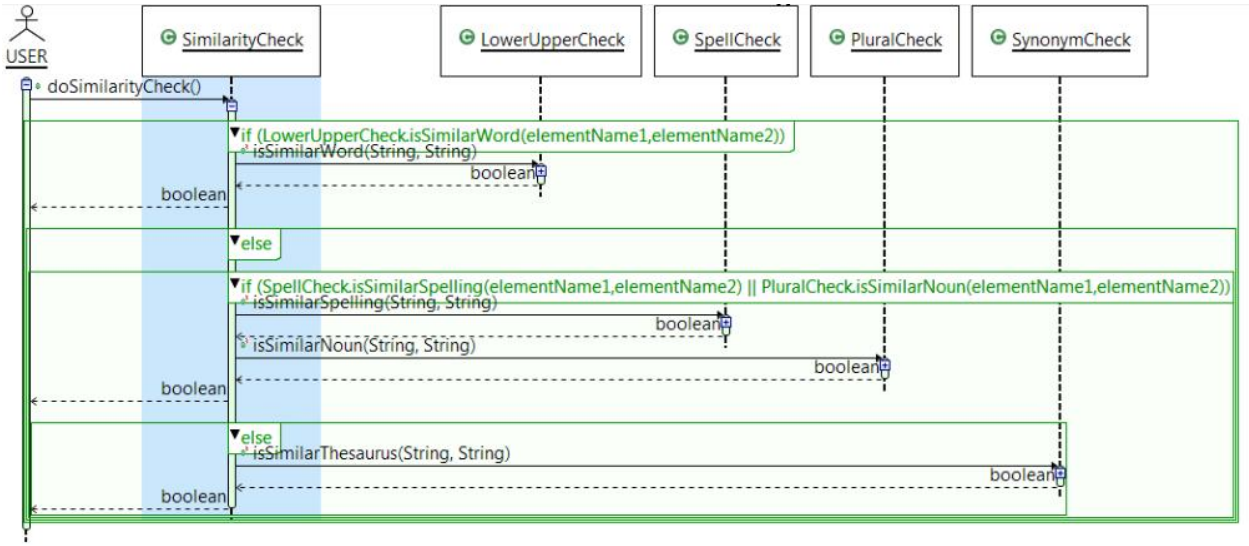


Figure 49: Similarity Package Sequence Diagram

Downloading

All files are uploaded and generated by the software are saved in the same "uploads" folder, under a sub-folder named based on the original creation date and time plus the name of the uploaded file. The previous semester's software only permitted downloading of PNG image files, but the Spring 2013 design allows for downloading of diagram source files, as well as intermediate files used to generate the PNG images. All requested files are put into a ZIP file so that the user only has to download the one compressed ZIP file. The user can choose to download a single diagram and its associated files, or to download all files in the project. Figure 50 shows the sequence diagram used for the design of the download project functionality.

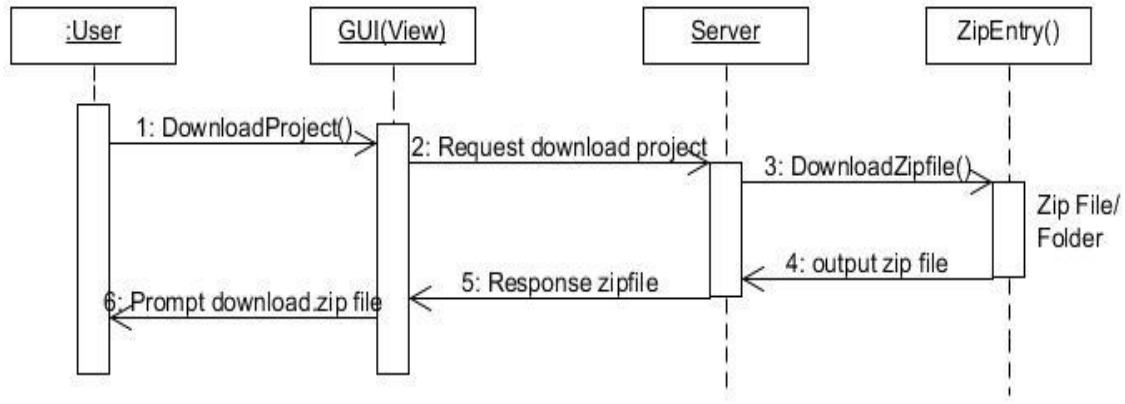


Figure 50: Download Project Sequence Diagram

Client-Server Interface

Merge Client-Server Communication

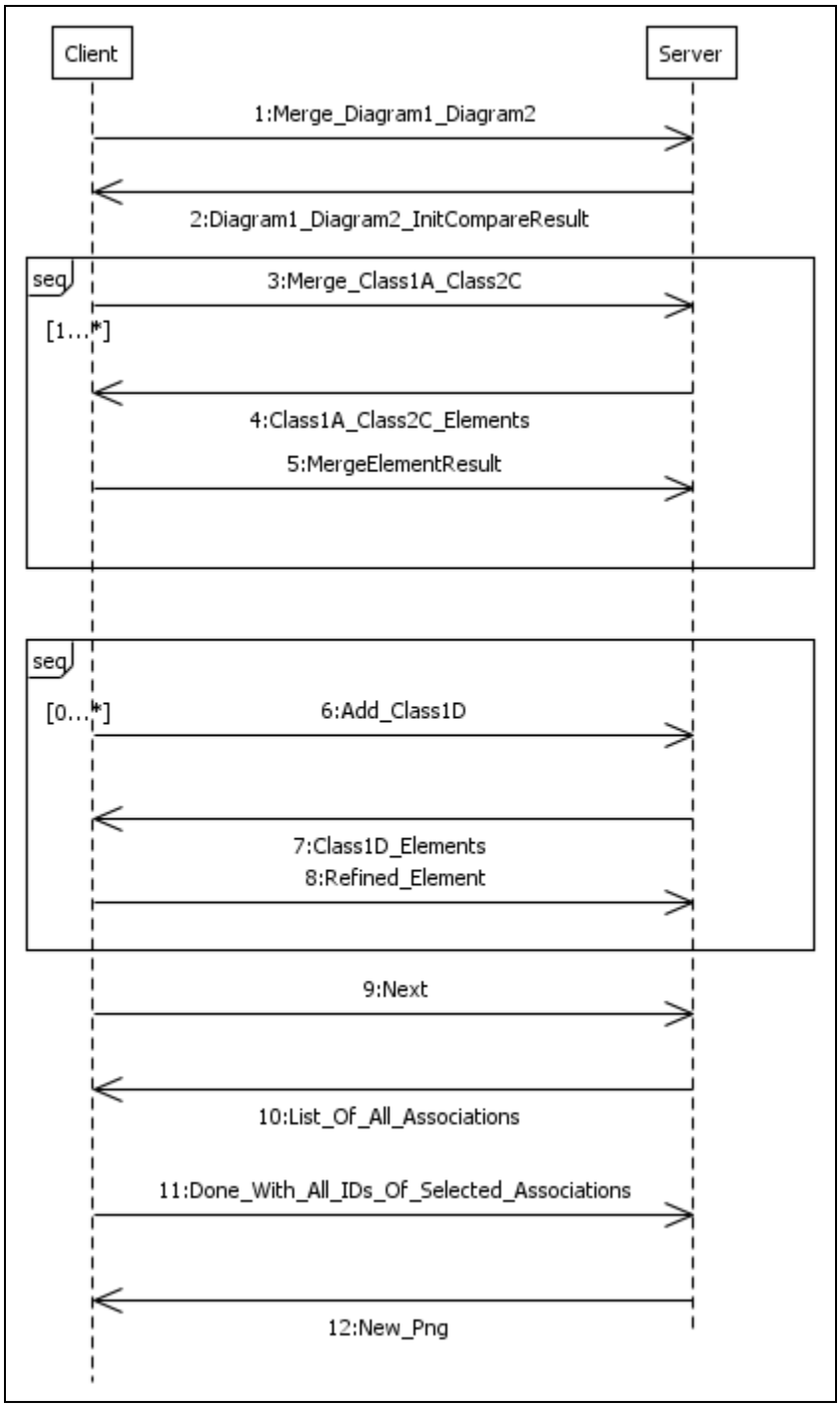


Figure 51: Client-Server Merge Sequence Diagram

Figure 51 shows the sequence diagram for the merge process. The steps are numbered in the same way as the steps describe in the Client-Side section. The steps are as follows:

1- The client requests to start the merge process for two diagrams.
2- The server returns the list of classes unique to diagram 1, classes unique to diagram 2, and classes that the two diagrams have in common.
3- The client sends the set of two classes "unique" to each diagram to consolidate into one.
4- The server returns lists of the elements unique to class 1, elements unique to class 2, and elements common to the two classes.
5- After choosing which class elements to keep in the new merged class, the selection is sent to the server. Steps 3-5 can be repeated as needed for other classes.
6- An "Add" request to keep a class unique to one diagram is sent to the server.
7- The server returns all the elements of that class to the client.
8- The selection of elements to keep in that class is sent to the server. Steps 6-8 can be repeated as needed for other classes.
9- The user clicks Next after finishing selecting what classes to merge or add.
10- The server sends the client information for associations that were in place for all the classes which are going to be included in the new merged diagram.
11- The selection of which associations to keep and which to discard is sent to the server.
12- The merge process is now complete, and the server can create a new merged diagram file, followed by generating an image for the new file. This image URL is then sent to the client to show to the user.

This flow was used as the basis of our client-server requests, described in the following sections.

Request Interface

In order to implement the various client-server requests and responses needed for the merge process, we created a Request interface. The Request logic was initially intended to be coded inside the XmiClassDiagramComparer class, which caused tight coupling between Request logic and XmiClassDiagramComparer logic. To help decouple the logics, we created a Request interface. Each type of request was implemented independently by several members of the development team.

The *Request* interface is under package *controller.comparer.xmi.request*.

```
public interface Request {

	public JSONObject request(JSONObject, XmiClassDiagramComparer);

}
```

All requests implement this Request interface. The set of request classes in the package *controller.comparer.xmi.request* are:

1- Refresh
2- Compare
3- Consolidate
4- Add
5- Break

6- Next
7- Done

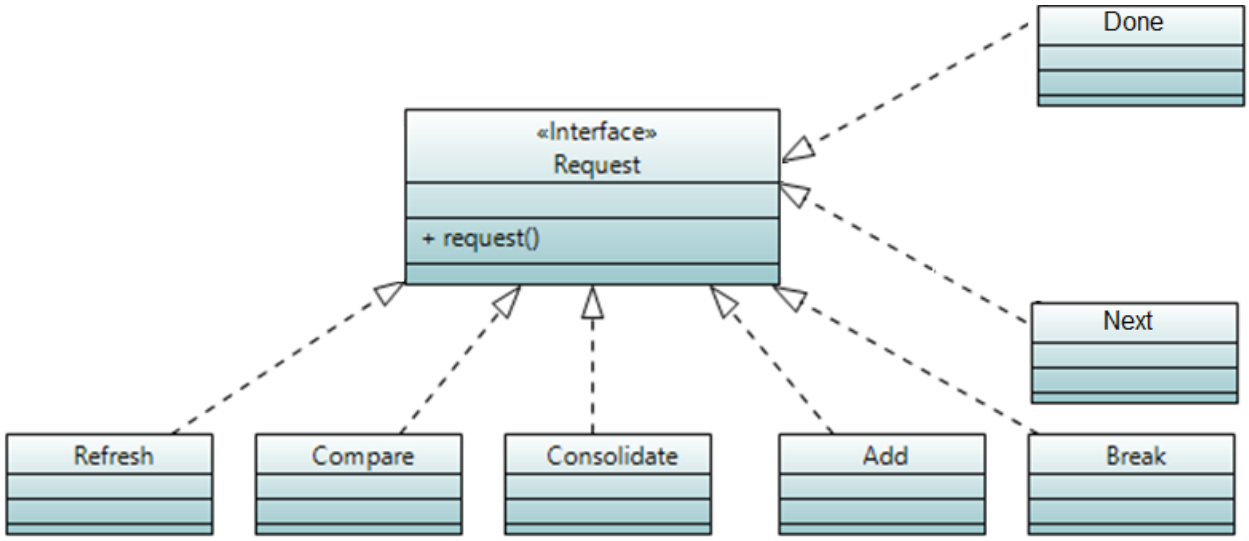


Figure 52: Request Class Diagram

The client does not need to know about this request structure; it just needs to send the JSON string defined in the JSON structure document. The servlet that communicates between the client and server will invoke XmiClassComparerDiagram.action() method. This action() method will use reflection to create an instance of one of the *Request* classes and invoke the request() method for that *Request* object. Request logic will have access to both class diagrams data and client's JSON object. The request class will perform the necessary logic and return a JSON object.

Initial Merge Request (Refresh Request)

Client will request to merge 2 diagrams. The client-server interface will instantiate an XmiClassComparer which has 2 arguments to pass in diagrams. The client-server interface will then request a refresh shown below to send back to the client.

The state of the merging process is stored on the server, and the **Refresh** request will return the class level information. This **Refresh** request is used at any point the client wants the state of the classes.

1. Client request - Merge_Diagram1_Diagram2

{
    Request: Refresh
}

2. Server response – (Diagram_Diagram2_InitCompareResult and other operations)

Diagram1 and *Diagram2* contain a list of classes that are unique to each diagram. *Same* contains a list of classes that were merged/consolidated by user or by server during initialization.

{
    Response: Success,
    Diagram1: [ClassA1, etc…],

```
        Diagram2: [ClassA2, etc…],
        Same: [ClassS1, ClassS2, etc…]
        Similar: [ClassS1 is similar to ClassS2, etc…]
}
```

Merge Sequence (Compare & Consolidate Requests)

Client request to merge 2 classes.

3. Client request - Merge_Class1A_Class2C

```
{
        Request: Compare,
        Class1: Class1Name,
        Class2: Class2Name
}
```

4. Server response – Class1A_Class2C_Elements

Client interface can display this information and should be straight forward for attribute and operations.

```
{
        Response: Success,
        Class1: Class1Name,
        Class2: Class2Name,
        Attributes: {
                Class1: [],
                Class2: [],
                Same: []
                Similar: []
        },
        Operations: {
                Class1: [],
                Class2: [],
                Same: []
                Similar: []
        }
}
```

5. Client request - MergeElementResult

Client sends the user's choices for each option to keep consolidate into a final class.

```
{
        Request: Consolidate,
        Class1: {
                Class: Class1Name,
                Attributes: [],
                Operations: []
        },
        Class2: {
                Class: Class2Name,
                Attributes: [],
```

Operations: []
},
Name: NewNameForClass, (user entered or default from one of the original classes)
}
After consolidation is completed for the 2 classes, the **Refresh** server response (2) JSON is returned.

Add Sequence (Add & Consolidate Requests)

Adding a class is basically the same as a merge/consolidate in step (3), but only 1 class is passed in and the other class field should not exist. Class 1 key and Class 2 key are mutually exclusive.

6. Client request - Add_Class1D

{
Request: Add,
Class1 or Class2: ClassName
}

7. Server response – Class1D_Elements.

List of elements for the class that is being added.
{
Response: Success,
Class1 or Class2: ClassName,
Attributes: [],
Operations: []
}

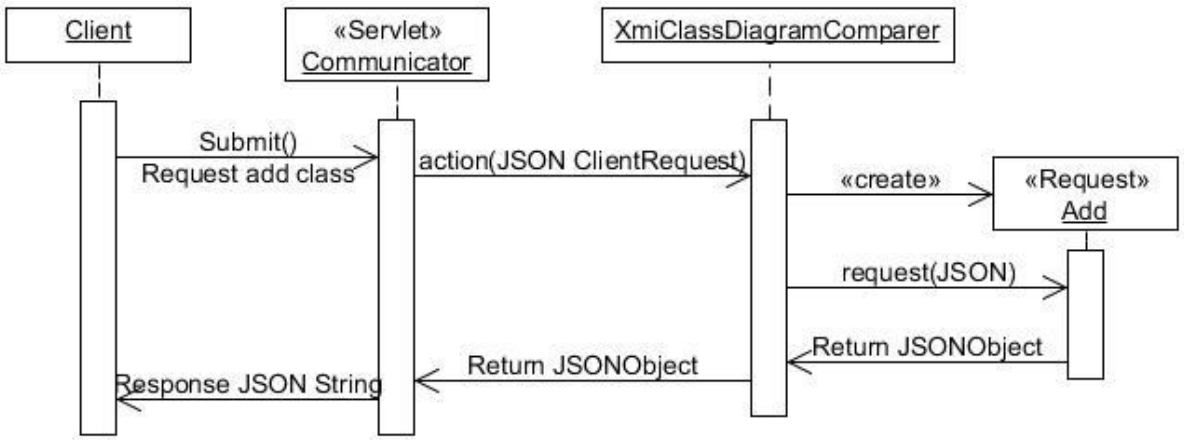


Figure 53 : Sequence Diagram of Building the JSON Add Request

8. Client request - Refined_Element (this is a **Consolidate** request for only 1 class)

Only send elements the user wants to keep. (Class 1 key and Class 2 key are mutually exclusive)
{
Request: Consolidate,
Class1 or Class2: {
Class: ClassName,
Attributes: [],

Operations: []
},
Name: NewNameForClass, (user entered or default original name)
}
After adding is completed, the **Refresh** server response (2) JSON is returned.

Associations Sequence (Next & Done Requests)

In order to enter the relationship page, the **Next** request will return the list of Associations and Generalizations.

9. Client request - Next

{
Request: Next
}

10. Server response – List of all Associations & Generalizations

Client interface can display the associations and generalizations for all classes that were merged.

{
Response: Success,
Diagram1: {
Associations: [{id:, text:},…],
Generalizations: [{id:, text:},…]
},
Diagram2: {
Associations: [[{id:, text:},…],
Generalizations: [{id:, text:},…]
}
}

11-1. Done_with_All_IDs_of_Selected_Associations

Client sends the selection the user wants to keep

{
Request: Done,
Diagram1: {
Associations: [id,…],
Generalizations: [id,…]
},
Diagram2: {
Associations: [id,…],
Generalizations: [id,…]
}
}

Undo Merge (Break Request)

Client request to break apart a Merged element into Class1 and/or Class2 elements.

Client request - Break

{
Request: Break,
Class: NameOfMergedClass
}

Server response

{
Response: Success,
}

Complete Merge

11-2. Done_with_All_IDs_of_Selected_Associations

Client presses complete and the merge data is passed into a merge processor for XMI.

12. New_PNG

After the server merge process completes, a PNG is generated and the path to the PNG is passed back to the client to view.

Testing Requests

A TestRequest.java file was developed for the purpose of testing different request methods. Refresh, Compare, Consolidate, Add, and Break requests were tested for the two test diagrams shown in

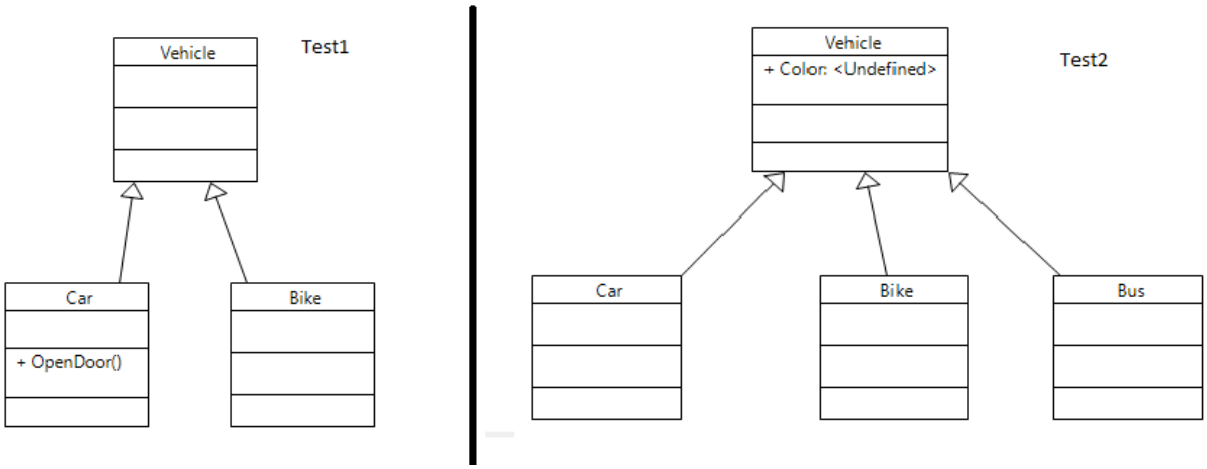


Figure 54: Test Diagrams for Requests

The request classes were successfully tested as shown in the test results table below.

Test case:	Objective:	Result:
Refresh	Returns similar and different class names	Successful
Compare	Returns similar and different elements for two classes	Successful
Consolidate	Merges elements from two classes and then does a Refresh step	Successful
Add	Adds desired elements from class, returns success/fail	Successful
Break	Breaks a previously merged class into its two original classes, returns success/fail	Successful
Next	Returns associations for the two diagrams	Successful
Done	Completes final merge of the two diagrams	Successful

Database

The enhancements made to the project required several changes to the MySQL database. This section details the new database. Shows the new database structure, generated using Toad Data Modeler MySQL 5.0. Figure 55 shows the new database structure.

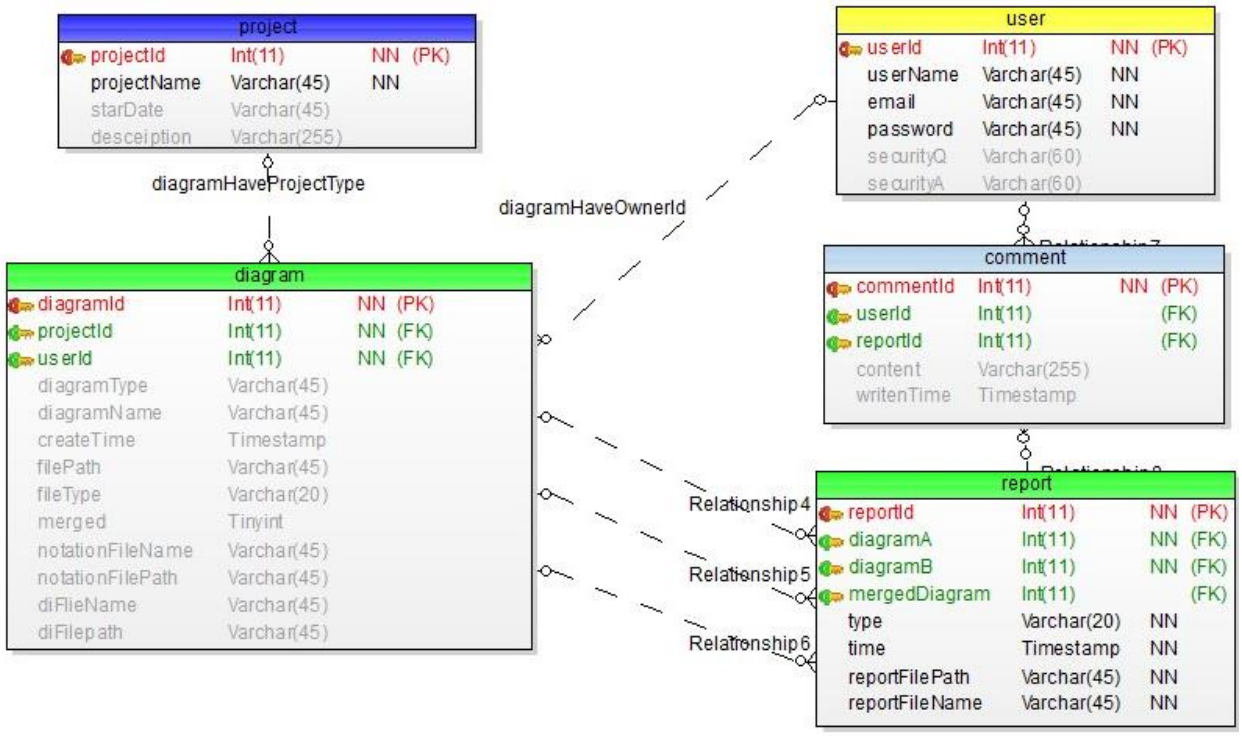


Figure 55: New Database Structure

Modification for User - Project - Diagram

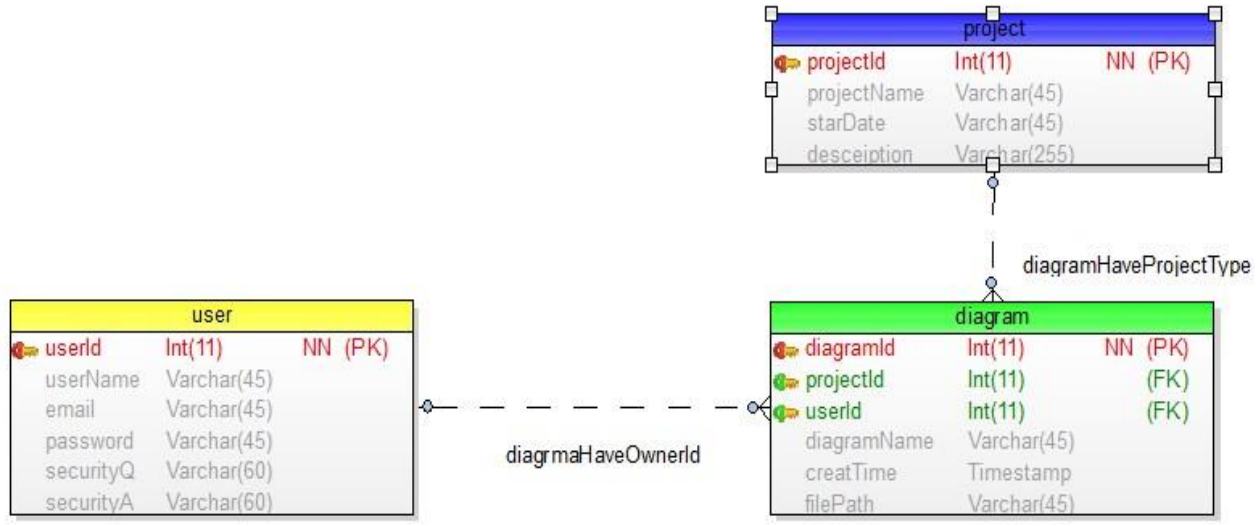


Figure 56: User-Project-Diagram Data Model

Changed:

1. ProjectId used to be put in the user table. Now we put the diagram table in the middle of user and project, so the user table and project table do not have a direct relationship any more. Diagram table contains both projectId and userId.
2. Add start date in the project table.

Benefit:

1. Improved the old version where one user could only join in one project. Now one user could join as many projects as he/she wants.
2. Diagrams can be grouped by projects. So the user could choose diagrams from the same project to compare or merge.

Modification for Diagram Table

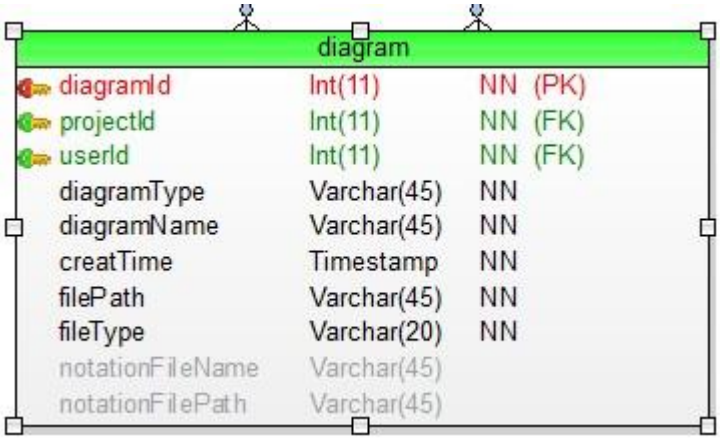


Figure 57: Diagram data model

Changed:

3. Add diagram type attribute. (class diagram or sequence diagram)
4. Add file type attribute. (Ecore file or UML file)
5. Add notation file name and file path

Benefit:

1. Make the diagram table support new diagram and data types.

Modification for Diagram - Report

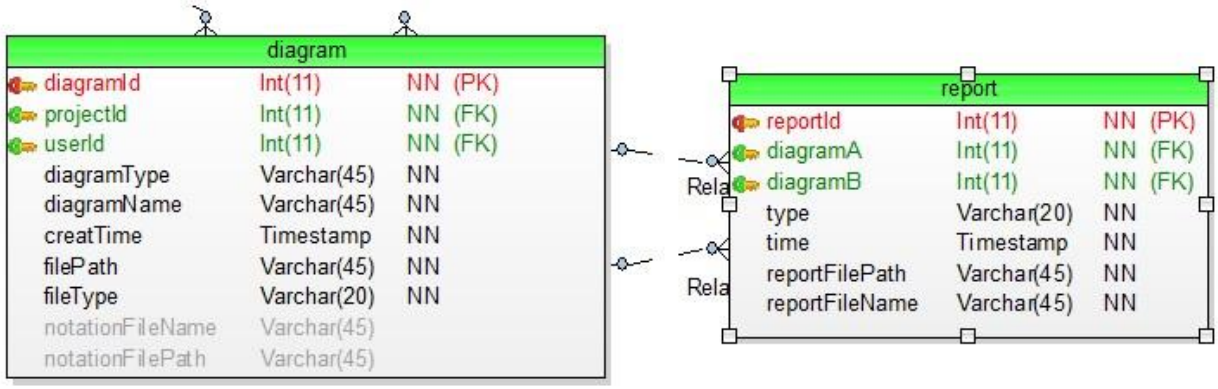


Figure 58: Diagram - Report Data Model

Changed:

2. Add type attribute to report table. (compare or merge)
3. Add reportFileName attribute to report table.
4. Switch comparedTime attribute to time.

Benefit:

5. Make the report table support new functions and data types.

Modification for User - Comment

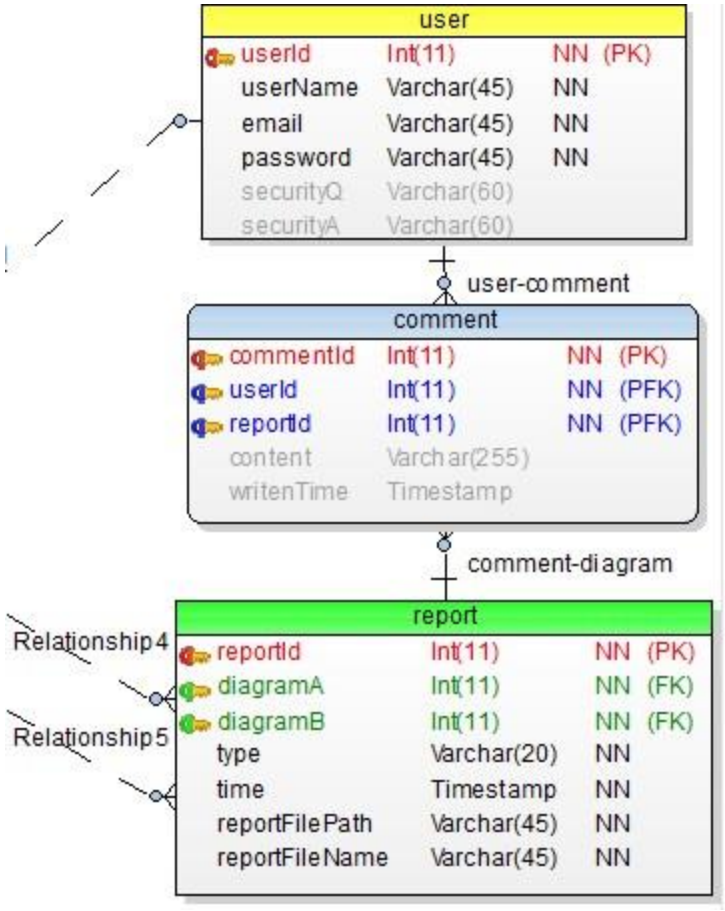


Figure 59: User - Comment Data Model

Changed:

6. Comment table connects to the report table instead of diagram table.
7. Comment table contains reportId instead of diagramId.

Benefit:

1. User could leave his/her comment under the compare or merge result. Old version database could only let user leave the comments under one diagram.

Example SQL Insert Queries

See Appendix B: Database Query for the query needed to set up the MySQL tables used by the Spring 2013 version of ClubUML. This section shows some queries that can be used to insert new values into the MySQL tables and then check the results.

Insert Project

Insert into project VALUES (000001, 'projectA', 'Jan2013', 'projectDesceiption')

select * from project

projectId	projectName	starDate	desceiption
1	projectA	Jan2013	projectDesceiption
NULL	NULL	NULL	NULL

Figure 60: Project Table Result

Insert User

insert into user VALUES (000001, 'admin', 'admin@clubuml.com', 'adminpd','','')

select * from user

userId	userName	email	password	securityQ	securityA
1	admin	admin@clubuml.com	adminpd		
NULL	NULL	NULL	NULL	NULL	NULL

Figure 61: User Table Result

Insert Diagram

insert into diagram (diagramId, projectId, userId, diagramType, diagramName, filePath, fileType, notationFileName, notationFilePath)

VALUES (000001, 000001, 000001, 'use case','2013 use case','../xxx/xxx.xmi','xmi', 'natationA' ,'../xxx/xxx')

select * from diagram

Filter: Edit: File: Autosize:

diagramId	projectId	userId	diagramType	diagramName	createTime	filePath	fileType	notationFileName	notationFilePath
1	1	1	use case	2013 use case	2013-02-22 17:04:03	.../xxx/xxx.xmi	xmi	natationA	.../xxx/xxx
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Figure 62: Diagram Table Result

Insert Report

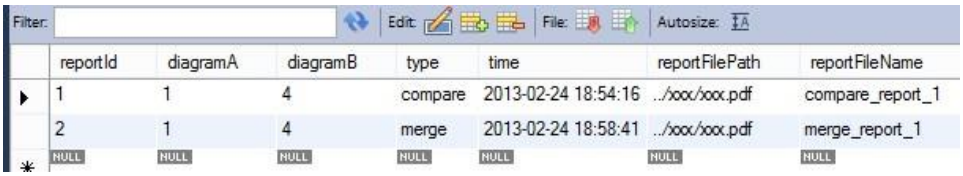
insert into report (reportId, diagramA, diagramB, type, reportFilePath, reportFileName)

VALUES (000001, 000001, 000004, 'compare','../xxx/xxx.pdf','compare_report_1')

insert into report (reportId, diagramA, diagramB, type, reportFilePath, reportFileName)

VALUES (000002, 000001, 000004, 'merge','../xxx/xxx.pdf','merge_report_1')

select * from report



reportId	diagramA	diagramB	type	time	reportFilePath	reportFileName
1	1	4	compare	2013-02-24 18:54:16	.../xxx/xxx.pdf	compare_report_1
2	1	4	merge	2013-02-24 18:58:41	.../xxx/xxx.pdf	merge_report_1
NULL	NULL	NULL	NULL	NULL	NULL	NULL

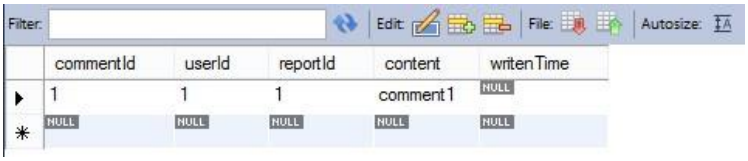
Figure 63: Report Table Result

Insert Comment

insert into comment(commentId, userId, reportId, content)

VALUES (000001, 000001, 000001, 'comment1')

select * from comment



commentId	userId	reportId	content	writenTime
1	1	1	comment1	NULL
NULL	NULL	NULL	NULL	NULL

Figure 64: Comment Table Result

System Performance and Test

Testing Methodology

Due to the agile nature of the methodology adopted (Scrum), the testing methodology has been incremental. The focus of the test in the initial stages of software development has been more white box tests that were performed by the developers, and as the project reached final stages, the focus shifted to black box testing by other members of the team using test cases.

Test case development has been closely aligned with the use cases. There are mainly two types of use cases: Those testing the normal behavior of the software based on expected user inputs and those testing the unexpected or complex scenarios.

Integration testing (testing after integration of the components developed by different developers both on the server and client side) was performed in the last phase of the software development. It is performed by the developers, before or while the test cases are being executed by members of the test team.

System testing of the final version of the software was performed by all members of the team who were available to participate in testing as the final check for software usability.

Static testing (testing without running the software, such as code reviews) were not conducted due to the aggressive schedule of the project, in favor of the dynamic testing described above.

Test Cases

RegisterNewAccount

Registration with valid data

Description:
Test the process of registration in ClubUML.

User should be able to find and click the register button in login page.
User need to be able to fill in the
username: test with multiple alphanumeric usernames, 1 to 10 chars.
password: test with multiple alphanumeric values, 1 to 10 chars.
password confirmation: type in exactly the same password.
email: enter a valid email. (better to test with multiple emails with different extensions)
Security question: enter a security question, with an answer.
clicking the register button.

Expected result: Registration success, user can login within the username and password.

Pre-conditions:
User goes to ClubUML homepage,

Post-conditions:
Providing a valid input, the user can register and land in the login in page with their account created.

Data required:
a username that does not already exist.

Registration with invalid data

Description:
Test the process of registration in ClubUML.

User should be able to find and click the register button in login page.
User need to be able to fill in the
username
password,
password confirmation
email
Security question
Following inputs should be entered to break the input validation:
Empty username
Username longer than 10
Use of non-alphanumeric characters in username
a combination of the above
Empty password
Password longer than 10 characters.
Password containing alphanumeric characters should be invalidated (according to the usecase)
Empty password confirmation.
Mismatching password confirmation
Empty email address
Email address with wrong format
Empty answer to the security question
clicking the register button.

Expected result: Registration fails, user is provided with appropriate message and should be given the chance to correct the errors and continue. the Error messages should be disappeared after continuing.

Pre-conditions:
User goes to ClubUML homepage,

Post-conditions:

If the input is invalid, user gets user friendly message according to the input validation criteria (denoted in use case and tested in step two of this testcase).

Data required:

No data is necessary to perform this test.

Register using existing username

Description:

Tests the ability of the system to detect that the username that is being entered during registration is not available and act per the Alternate flow of RegisterNewAccount usecase.

1- Register a user with a specific username according to RegisterNewAccount Usecase.
2- Try to register another user with the same username.

Expected result: User should get a message at the top of the page that the username already exists and should be given the chance to correct and continue.

Pre-conditions:

The Registration testcase should succeed before this testcase is performed.

Post-conditions:

The user should fail to create the account and should stay on the registration page.

Data required:

No data is necessary to perform this test.

UploadDiagram

Upload valid file

Description:

Tests the ability of the system accept valid files from the user

- Login, and the landing page should have a mechanism to select upload file type.
- Select Ecore file format.
- Click choose file and in the dialog box, select the Ecore class diagram file.
- Click upload button.
- Perform the same steps for Papyrus file format. User should be able to select multiple files if necessary in step 4.
- Repeat the 1-4 steps for sequence diagrams.

Expected result: The new diagram should be added to the list of uploaded diagrams. the image should be visible next to the list of files.

Pre-conditions:

The login should be successful.

Post-conditions:

Website can successfully accept Ecore and Papyrus formats. for class diagrams and sequence diagrams.

Data required:

Valid Ecore and Papyrus files for class diagram and sequence diagram.

Upload invalid file

Description:

Tests the ability of the system to reject invalid files, as denoted in the alternate flow of Upload Usecase

1. Login, and the landing page should have a mechanism to select upload file type.
2. Select Ecore file format.
3. Click choose file and in the dialog box, select the Papyrus class diagram file.
4. Click upload button.
5. Perform the same steps for other random file formats except Ecore. Also try sending multiple files that only some of them are invalid or not Ecore format.
6. Perform the above steps, but select Papyrus in step 2 and only send Ecore or other file formats. Try mixing multiple formats here too by selecting multiple files to upload.
7. Perform the same steps for Sequence diagram in Papyrus format.

Expected result: The website should reject the file and provide user friendly message and allow the user to upload the correct files right away.

Pre-conditions:

The login should be successful.

Post-conditions:

Test succeeds, meaning that website behaves in a way that no invalid file can be uploaded.

Data required:

Valid and invalid Ecore and Papyrus files for class diagram and sequence diagram.

Papyrus files for other diagram types.

Other random file types, such as image etc.

Login

Login using valid password

Description:

Tests the ability of the system accept valid username and password in the login page.This test case is to verify the login process of ClubUML web application. The user will type in the username and password to login the application,

1. Navigate to the login page.
2. Enter the username and password in the appropriate textboxes.
3. Click login; The successful login page should appear.
4. Click continue.

Expected result: The Login button and continue button should succeed and user should see the ClubUML webapp page.

Pre-conditions:

No pre-conditions.

Post-conditions:

if test succeeds, users who already have an account can successfully log in to the website.

Data required:

A username and password pair of a registered user.

Login using invalid username

Description:

Tests the ability of the system reject a username that does not exist, in the login page.

1. Navigate to the login page.
2. Enter the invalid username and an arbitrary password in the appropriate textboxes.
3. Click login;

Expected result: The Login page should refresh with a message that either username or password is incorrect.

Pre-conditions:

No pre-conditions.

Post-conditions:

if test succeeds, the website is able to reject invalid usernames with a correct message. the server denies the attempt and redirects the user to the login failed page.

Data required:
A username that does not exist.

Login using invalid password

Description:
Tests the ability of the system reject a password that is wrong, although the user is valid.

1. Navigate to the login page.
2. Enter the username and an arbitrary wrong password in the appropriate textboxes.
3. Click login;

Expected result: The Login page should refresh with a message that either username or password is incorrect.

Pre-conditions:
No pre-conditions.

Post-conditions:
if test succeeds, the website is able to reject invalid passwords with a correct message. The user typed in the wrong username or password, and the server denies the attempt and redirects the user to the login failed page.

Data required:
A username that exist and a password that is wrong

Compare Diagrams

Compare Uploaded Diagrams

Description:
Tests the ability of the system to compare two valid diagrams (Ecore) that are already uploaded.

1. Upload two diagrams for the purpose of test.
2. Select the diagrams and click Go to Compare button..

Expected result: The user should be able to see the expected differences in the detailed report.

Pre-conditions:
The user should have already logged in.

The Upload file testcase should succeed.

Post-conditions:
The system is able to compare two diagrams and output the report.

Data required:
Two diagrams with a known set of differences to fit the test purposes (will be elaborated in next versions). The diagrams should be in Ecore format

Save Comparison Report

Description:
Tests the ability of the system save the comparison report.

1. Upload two diagrams for the purpose of test or skip if there are already files to select.
2. Select the diagrams and click Go to Compare button..
3. On the report page, click on save button.

Expected result: The user should be able to save the PDF version of the file on the disk. no pop ups should open.

Pre-conditions:
The user should have already logged in.

The Upload file and Compare Uploaded Diagrams testcases should succeed.

Post-conditions:
The system is able export the report to PDF

Data required:
Two diagrams to upload for test. The diagrams should be in Ecore format

Merge Diagram

Merge of simple diagram with class differences

Description:

Test the process of simple class diagram merge. No conflicts, only classes that exist in one and not the other.

a) Upload the test papyrus files and select them. Refer to Data Required section.
b) Click Go to Merge button
c) Website should display two diagrams side by side.
d) Three lists should be displayed:
    a) Classes in Diagram A only: Bike, Passenger, Vehicle
    b) Classes in Diagram B only: Bicycle, Human, Vehicle
    c) Common Classes: Bus
e) Select the elements that you want to keep. I.e. add all elements from diagram A plus Human diagram from B.
f) Click on the merge diagram to finalize. select and commit all the generalizations.

Expected result: The Merged diagram should be displayed. It should be diagram A with the inclusion of Human class.

Pre-conditions:

User goes to ClubUML homepage,

Post-conditions:

Merged diagram functionality is satisfies the inclusion of the classes that are not in one diagram in the final result.

Data required:

The Papyrus files for the diagrams shown in Figure 65.

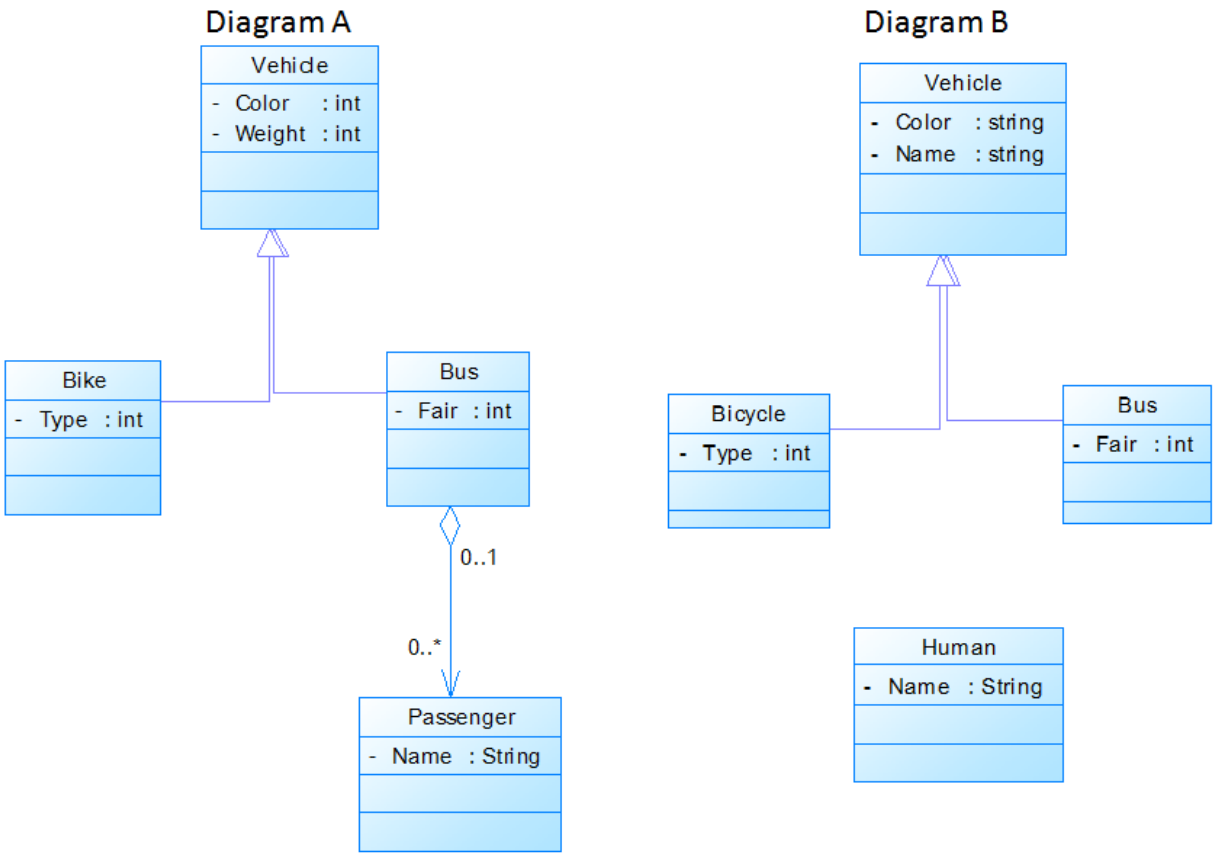


Figure 65: Class Diagrams Used for Test

Merge classes with Attribute differences

Description:

Test the process of simple class diagram merge. Testing the ability of the website to handle property differences as a conflict between the classes being merged.

1. Upload the test papyrus files and select them. Refer to Data Required section.
2. Click Go to Merge button
3. Website should display two diagrams side by side.
4. Three lists should be displayed. Showing differences and commonalities (Ref to prev. testcase)
5. Select the elements that you want to Merge. Select class Vehicle From diagram A and class Vehicle from diagram B.
6. Click on the merge button that is associated with merging the selected elements.
7. The software should show a dialog showing the attribute conflict. Select the desired properties. Change the name to VehicleMerged. Click save.
8. The VehicleMerged class should be added to the center and two vehicle classes should be removed from both sides.
9. Click on the next button diagram to finalize. Select and commit the generalization.

Expected result: The Merged diagram should be displayed. It should have two classes with the Name Vehicle and Bus including the appropriate properties that were selected in step 7.

Pre-conditions:
User goes to ClubUML homepage,

Post-conditions:
Merged diagram functionality is satisfied. The software is able to handle attribute differences between two classes.

Data required:
The Papyrus files for the diagrams shown in Figure 65.

Break classes with Attribute differences that were once merged

Description:
Test the break functionality of merge UI. If two classes that are merged in the merge UI, user must be able to roll back the change.

1. Upload the test papyrus files and select them. Refer to Data Required section.
2. Click Go to Merge button
3. Website should display two diagrams side by side.
4. Three lists should be displayed. Showing differences and commonalities (Ref to prev. testcase)
5. Select the elements that you want to Merge. Select class Vehicle From diagram A and class Vehicle from diagram B.
6. Click on the merge button that is associated with merging the selected elements.
7. The software should show a dialog showing the attribute conflict. Select the desired properties. Change the name to VehicleMerged. Click save.
8. The VehicleMerged class should be added to the center and two vehicle classes should be removed from both sides.
9. Select the VehicleMerged. Click on the break button. Expected result: The Merged diagram

   Expected result: The merged class should be removed and the changes should be reverted back to the initial stage and you should be able to see two vehicle classes on both sides.

Pre-conditions:
User goes to ClubUML homepage,

Post-conditions:
Merged diagram functionality is satisfied. The software is able to break merged classes.

Data required:
The Papyrus files for the diagrams shown in Figure 65.

Merge of simple diagram with Name differences

Description:
Test the process of simple class diagram merge. Testing the ability of the website to handle name differences as a conflict between the classes being merged. In this test, the Passenger class in diagram A will be merged with the Human class in diagram B, in a way that we will have diagram A, but only the Passenger class is changed to Human class and the association is preserved.

1- Upload the test papyrus files and select them. Refer to Data Required section.
2- Click merge button
3- Website should display two diagrams side by side.
4- Three lists should be displayed. Showing differences and commonalities (Ref to prev. testcase)
5- Select the elements that you want to Merge. Select class Passenger From diagram A and class Human from diagram B.
6- Click on the merge button that is associated with merging the selected elements.
7- The software should show a dialog showing the name conflict. User should be able to select the desired name, in this case, Human (you should be able to click on Human button too). Also select the name attribute to be kept.
8- Click on Save.
9- Add all elements on diagram A (on the left), meaning we want to keep them.
10- Click on the merge diagram to finalize.

Expected result: The Merged diagram should be displayed. It should be the diagram A, but the name of the Passenger class should be changed to Human.

Pre-conditions:
User goes to ClubUML homepage,

Post-conditions:
Merged diagram functionality is satisfied. The software is able to handle name differences between two classes.

Data required:
The Papyrus files for the diagrams shown in Figure 65.

Merge of simple diagram with Association differences

Description:

Test the process of simple class diagram merge. Testing the ability of the website to handle association conflicts between the classes being merged. The system should be able to detect association cardinality differences and provide mechanism to resolve that.

1- Upload the test papyrus files and select them. Refer to Data Required section.
2- Click merge button
3- Website should display two diagrams side by side.
4- Three lists should be displayed. Showing differences and commonalities
5- The classes Bicycle and Vehicle must be seen in the section dedicated to items existing in both left and right.
6- Select the elements that you want to keep should be already viewed in the middle.
7- Click on the next button.
8- The software should show a dialog (web page) showing Association conflict. User should be able to select the desired association type and cardinality from available Association choices.
9- By accepting that dialog, the desired cardinality between two classes should be seen in the final result.

Expected result: The Merged diagram should be displayed. It should be the diagram with two classes, Bicycle and Vehicle with the desired association type.

Pre-conditions:

User goes to ClubUML homepage,

Post-conditions:

Merged diagram functionality is satisfied. The software is able to handle name differences between two associations.

Data required:

The Papyrus files for the diagrams shown in Figure 66.

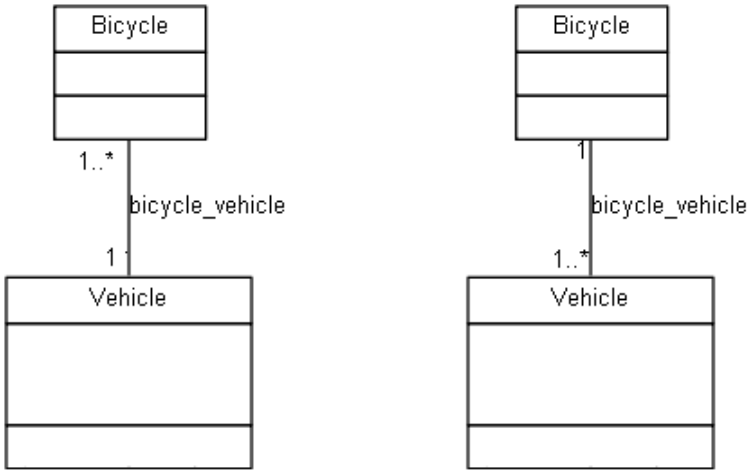


Figure 66: Class Diagrams for Association Test

Select different diagram types

Description:
This test case checks what happens if user select different types diagrams to merge

Pre-conditions:
User has login ClubUML.

User has uploaded more than two UML diagrams of the same project.

Post-conditions:
Error message window prompts up.

User click “OK” and goes back to select diagram step.

No new diagram will be generated.

Data required:
None

Select more than two diagrams to merge

Description:
This test case checks what happen if user select more than two diagrams to merge

Pre-conditions:
User has login ClubUML.

User has uploaded more than two UML diagrams of the same project.

Post-conditions:
Error message window prompts up.

User click “OK” and goes back to select diagram step.

No new diagram will be generated.

Data required:
None

Selects less than two diagrams to merge

Description:
This test case checks if the user selects less than two diagrams, the error message will be shown.

Pre-conditions:
The user is logged into the ClubUML application.

At least one diagram has been uploaded before proceeding with comparison.

Post-conditions:
Shows the error message “*Please select at least two diagrams”.*

Data required:
None, or one diagram.

Download Project

Download all diagrams in a project

Description:

Test the download project functionality of ClubUML.

1- If the project content is empty, upload multiple arbitrary diagrams for the purpose of test.
2- There should be a download project button in the landing page of the ClubUML website.
3- Click "Download Project" button
4- Select a path to save the file.
5- Project page should be displayed again.

Expected result: A zip file should be created in the specified path containing all the diagrams in the project (the same files uploaded in step 1)

Pre-conditions:

User goes to ClubUML homepage and is already logged in.

Post-conditions:

The file is saved and the project page will be displayed.

Data required:

Some arbitrary Ecore and papyrus diagrams to be uploaded to the website for the purpose of test.

Remove Diagram

Deleting an uploaded diagram

Description:

Tests if the website performs the deletion of a diagram from a project.

1- If the project content is empty, upload one or multiple arbitrary diagrams for the purpose of test.
2- Select the diagrams to delete.
3- Click "Remove" button

Expected result: The diagram should be deleted from users' diagram list.

Pre-conditions:

User goes to ClubUML homepage and is already logged in.

Post-conditions:

The selected diagrams are deleted.

Data required:

Some arbitrary Ecore and Papyrus diagrams to be uploaded to the website for the purpose of test.

Miscellaneous

Show file diagram

Description:

Users choose one of the file on file list. And click display button.

Expected result: the chosen file show on the left area of the same pages.

Pre-conditions:

User chooses one file and click display button.

Post-conditions:

A file displayed

Data required:

File, file path from database, file upload time.

Test Results

Test Case Generation [Login page]:

As per the test cases in the previous section, and few additional ones added, the following are the test cases generated

- **Precondition**: Login page with blank fields

Test case	Check Item	Test case Objective	Steps to Execute	Test data / Input	Excepted Results	Actual Result
TC001	Login Page	Leave all fields as blank and click Log-in button	Click Login	Username: NULL Password: NULL	By leaving all fields as blank and on click Login button then mandatory symbol (*) should appear in front of Username and Password fields.	PASS
TC002	Username	Enter Invalid Username	NA	Username: xyz [not in DB]	By entering invalid Username then an error message should appear as " Please Enter Valid Username.	PASS
TC003	Username	Enter Valid username	NA	Username: ClubTester	Allow user to proceed.	PASS
TC004	Password	Enter Invalid password	NA	Password: *********	By entering invalid password, an error message should appear as "Please Enter Valid password".	PASS
TC005	Password	Enter valid password	NA	Password: **********	Allow user to proceed.	PASS
TC006	Log-in button	Correct credentials	Click Login button		Take user to respective page.	PASS

- **Post Condition**: Continue with access to the respective page after Logging.

Test Case Generation [Registration page]:

- **Precondition**: Registration page with blank fields

Test case	Check Item	Test case Objective	Steps to Execute	Test data / Input	Excepted Results	Actual Result
TC001	Registration Page	Leave all fields as blank and click Register button	Click Create account button	Username: NULL Password: NULL Email id: NULL	By leaving all fields as blank and on click Login button then mandatory symbol (*) should appear in front of Username, Password and email id fields.	PASS
TC002	Username	Enter Valid username	NA	Username: Sarvesh	Message “Registration completed” appear.	PASS
TC003	Password	Enter valid password	NA	Password: **********	Message “Registration completed” appear	PASS
TC004	Email id	Enter valid email id	NA	Email: Svic@gmail.com	Message “Registration completed” appear	PASS
TC006	Create account button	Correct credentials	Click Create account button		Message “Registration completed” appear	PASS

- **Post Condition**: Continue with access to the settings page after registration.

Test Case Generation [Upload page]:

- **Precondition**: The login should be successful.

Test case	Check Item	Test case Objective	Steps to Execute	Test data / Input	Excepted Results	Actual Result
TC001	Upload file	Leave all fields as blank and click Upload button	Click Upload	NA	Shouldn't proceed to compare and merge functionality.	PASS
TC002	Upload file	Select Ecore drop down selection and try selecting XMI files	Select ECORE Select XMI file Click Upload	NA	Shouldn't proceed to compare and merge functionality.	PASS
TC003	Upload file	Selecting Ecore and further proper ecore files selection	Click Upload	NA	Allow user to proceed.	PASS
TC004	Upload file	Select XMI in drop down selection and try selecting ecore files	Click Upload	NA	Shouldn't proceed to compare and merge functionality.	PASS
TC005	Upload file	Selecting XMI and further proper XMI files selection	Click Upload	NA	Allow user to proceed.	PASS

- **Post Condition**: Continue with Compare and Merge functionality.

Test Case Generation [Compare Upload page]:

- **Precondition:** The user should have already logged-in and the Upload file test case should succeed.

Test case	Check Item	Test case Objective	Steps to Execute	Test data / Input	Excepted Results	Actual Result
TC001	Compare diagram	Selecting only one diagram	Click Compare	NA	Shouldn't allow to proceed and generating the report.	PASS
TC002	Compare diagram	Selecting more than one diagram	Click Compare	NA	Shouldn't allow to proceed for comparing	PASS
TC003	Compare diagram	Selecting exactly two diagrams	Click Compare	NA	Compare and generate report	PASS

- **Post Condition**: The system is able to compare two diagrams and output the report.

Test Case Generation [Display diagrams]:

- **Precondition**: User already uploaded the files

Test case	Check Item	Test case Objective	Steps to Execute	Test data / Input	Excepted Results	Actual Result
TC001	Test1.ecore	Display the UML diagram	Choose a diagram Click on display	Ex: Test1.ecore	Successfully displays the UML diagram	PASS
TC002	Nothing. Blind display click	To throw null exception	Click on display	Nothing	Successful display of exception	PASS

- **Post Condition**: nothing

Test Case Generation [Download Project]:

- **Precondition**: Project files present in server with the user name.

Test case	Check Item	Test case Objective	Steps to Execute	Test data / Input	Excepted Results	Actual Result
TC001	Download.zip	Download the files to local system	Click on download project	N/A	Successfully displays download popup. The file zip has some size	PASS
TC002	Download.zip	Empty zip file. As without uploading or doing anything in server.	Click on download project	Nothing	Successfully displays download popup. The file will be default size as it is empty	PASS

- **Post Condition**: nothing

Test Case Generation [Merge page]:

- **Precondition:** The user should have already logged-in and the Upload file test case should succeed.

Test case	Check Item	Test case Objective	Steps to Execute	Test data / Input	Expected Results	Actual Result
TC001	Merge- simple diagram with class differences		According to Testcase	Diagrams in Testcase	The Merged diagram should be displayed. It should be diagram A with the inclusion of Human class.	PASS
TC002	Merge classes with Attribute differences		According to Testcase	Same as TC001	The merged diagram should have two classes with the Name Vehicle and bus including the appropriate properties	PASS
TC003	Merge - Adding classes		Included in TC001	Same as TC001	User should be able to add a class from one side to the final result	PASS
TC004	Merge - Break		According to Testcase		User should be able to undo merging two classes	PASS
TC005	Merge - Association		According to Testcase		Result should be the diagram with two classes, Bicycle and Vehicle with the desired association type	PASS

- **Post Condition**: Continue with access to the respective page.

Test Case Generation [merge xmi files] (duplicate):

Precondition: User already uploaded xmi files to server

Test case	Check Item	Test case Objective	Steps to Execute	Test data / Input	Excepted Results	Actual Result
TC001	Test1.uml, null	Compare one xmi file	Select one .uml file Click on compare	Test1.uml	Message “select 2 xmi files to compare”	PASS
TC002	Test1.uml, Test1.ecore	Compare two different files	Choose one file as .ecore and another file as .xmi file	Test1.ecore, Test1.uml	Message “ select 2 files of same type”	PASS
TC003	Test1.uml, Test2.uml	Compare two uml files	Select 2 .uml files Click on compare	Test1.uml, Test2.uml	Display class merge communicator page for next steps	PASS
TC004	Add	Add a class	Choose any one class from either diagram 1 or diagram 2	Ex: Radar	A new named or the same name Radar appears under common classes	PASS
TC005	Break	Break a class	Choose a class under common classes Click on break	Ex: Radar	The class will be removed from the common class	PASS
TC006	Next	Shows relations between classes from both the diagrams	Click on next	N/A	Relations must be shown	Yet to be done
TC007	Merge	Merge 2 classes. It can be either from both the diagrams	Select one class from diagram 1 and another class from diagram 2 Click on merge	Ex: Rider (diagram 1) and Bus (diagram 2)	Shows a new name for the merged class. Enter a name or click on one button. Then click on save. Now a new class name will be shown	PASS

TC008	Merge	Choose 2 same classes and click on merge	Select 2 same classes and click on merge	Ex: Vehicle(diagram 1) and Vehicle (diagram 2)	Shows a new name for the merged class, Enter a name or click on one button. Then click on save. Now the vehicle class will be shown.	PASS
TC009	Merge	After the previous step checking for relations	After the previous step. Click on merge The table will now show the relationships identified in the classes. Choose relations and click on commit	N/A	Commit must be successful	FAIL
TC009	Merge	Few of the merge tests must be done				

Test Case Generation [Comment Authorization]:

- **Pre Condition**: User was logged-in, At least one diagram has been uploaded, and At least one comment has been post.

Test case	Check Item	Test case Objective	Steps to Execute	Test data / Input	Excepted Results	Actual Result
TC001	Comment Authorization	Trying to reply on comment without being logged-in.	Click reply	Enter Comment	Without having proper authorization shouldn't be allowed to comment.	

- **Post Condition**: Show the "Replay Button" beside the comment.

[Comment Length]

- **Pre Condition**: User was logged-in; At least one diagram has been uploaded.

Test case	Check Item	Test case Objective	Steps to Execute	Test data / Input	Excepted Results	Actual Result
TC001	Comment length	Try inserting larger comments for the diagram.	Click Comment.	Enter Comment	Try inputting more than 250 characters. Shows the error message “Comments cannot exceed 250 characters.”	

- **Post Condition**: Show the error message “Comments cannot exceed 250 characters.”.

Conclusion:

1. Started with the “Integration Testing” Story and ***all the test cases were successful.***
2. Others which were stated in the use case but do to time constrains couldn’t be implemented are kept as “open issues”
3. Few of the merge test cases must be done.

Open Issues

The following items are known issues or limitations in the current version of the software that should be resolved in future versions. Some are unexpected bugs that came up, while others are functionality we intended to implement but lacked the time and resources to accomplish before the end of the semester.

- We were not able to successfully deploy the application on the COE web server (rho.coe.neu.edu). Early on in the term, we saw that the Fall 2012 version of the software was not capable of displaying images of uploaded diagrams. We later found out that the software hosted on the Rho server was never able to display the images, and we did not have the proper permissions to setup the required GraphViz software on the server, so we instead focused on implementing everything for use on our local machines.
- We intended to include the ability to remove diagrams, but since it was lower priority than other features and not addressed until later in the semester, it was not completed.
- Currently, there is no working comment functionality. We set up the new database to allow a comment to be added to either a compare or merge result, whereas the previous semester set up comments to be added to individual diagrams. If we had time to include a comment when completing a merge, the comment would be linked to the two diagrams going in to the merge as well as the resulting merged diagram. In this way, the user could see the comment when looking at any of the diagrams involved.
- We now support setting a value to show if a diagram has been merged or not. This would be useful to show the user on the main display page, perhaps as an icon, so the user can instantly ascertain which diagrams were uploaded draft documents and which are (potentially final version) merge results.
- Merging a diagram that has been merged before results in unexpected problems, such as missing elements when trying to merge the previously merged diagram. This is due to reuse of XMI element IDs in the merged diagram. We would need to develop a way to generate new IDs for each new diagram.
- There may be issues if we try to merge two Papyrus diagrams from the same Papyrus model. We only focused on one diagram per model and did not test merging of multiple diagrams from the same model.
- When a merged element keeps both an association and generalization, the association isn't drawn in the resulting diagram's image. However, when opening the merged diagram in Papyrus, both the association and generalization are still shown.
- Sometimes associations or generalizations show up twice in the Association merge step.
- If adding two classes with the same name (for example add Car from the left diagram and then add Car from the right diagram without merging them), one is dropped. This is normally okay since the user wouldn't have two of the same class name in one diagram, but the user isn't warned that one of the classes will be dropped.

Conclusion

Analysis of the Process

Scrum definitely helped the team achieve our goals. It could have been better but overall for being many team member's first time around with Scrum it went well. Scrum empowers the team to lead the project development of tasks. The problem we had was that everyone was new to the project and process. The team did not take ownership at first but as the weeks went by the team took more ownership and helped with the tasks. One thing that could have helped the team do better was the story creation. If we had more time in class to write up the stories fully, the team would understand the story criteria better and complete it better during the week.

Subversion also helped the team achieve our goals. One problem we had was that some team members were new to the tool and took some time to get used to it. This is normal for any new tool but we had to go through it. As part of understand how to use the tool there was some work that got overwritten by another commit but we were able to retrieve back those changes so nothing was lost. This however caused some confusion and frustration. As we learned the tool, these problems went away. Overall, the tool helped keep the code and artifacts organized and controlled.

Assessment of the Work

Scrum Metrics

Scrum results show that there was not a real up or down swing in work. It looks pretty constant but towards the end the team really picked it up a bit and completed open tasks. Ideally we would have liked a ramp up and then a ramp down of work as we get to the completion; we were just not able to achieve that. We really got productive later than we would have liked and that led to the work being bunched towards the end. The spike in week 13 work was due to some last minute work that was not foreseen and also the fact that we really broke down the stories to be smaller with the hope to focus the team better and have completed stories at the end. It did help since we completed the most stories of any other week.

The following diagram shows the number of stories created per week (in blue). It also shows the number of stories completed from that week (in phase, in red) and also stories completed from a previous week (out of phase, in orange). There were 15 weeks this semester and the first few weeks we did not have stories because we were still getting organized with the process. Week 7 was counted with week 6 work.

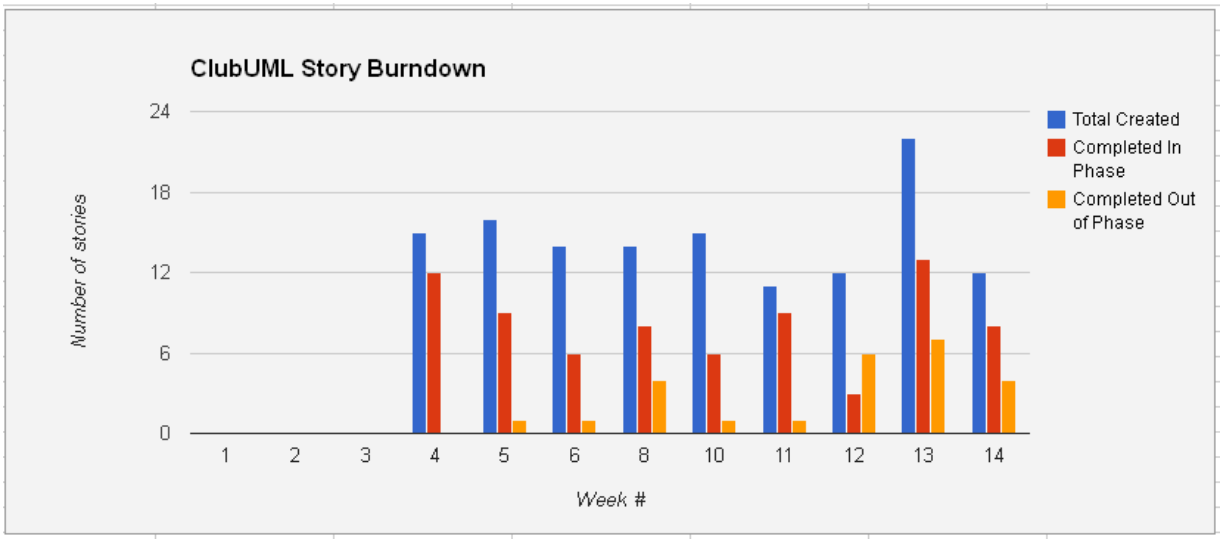


Figure 67: Burndown Chart Diagram

Code Metrics

The team worked hard to get the necessary changes into the baseline. We had over 200 commits into svn and produced over close to 5000 new source lines of code (sloc). The following diagrams were produced from the Tortoise SVN statistics feature.

From the semester, we committed files into the Spring2013 code baseline around mid-February (week 7) and continued making commits until the end of the semester (week 16). A total of 202 commits were made over that span with most of them coming towards the second half. The team generated just under 5000 new sloc and put the total at 7610 sloc. Sloc count was computed by an Eclipse plugin called Metrics2.

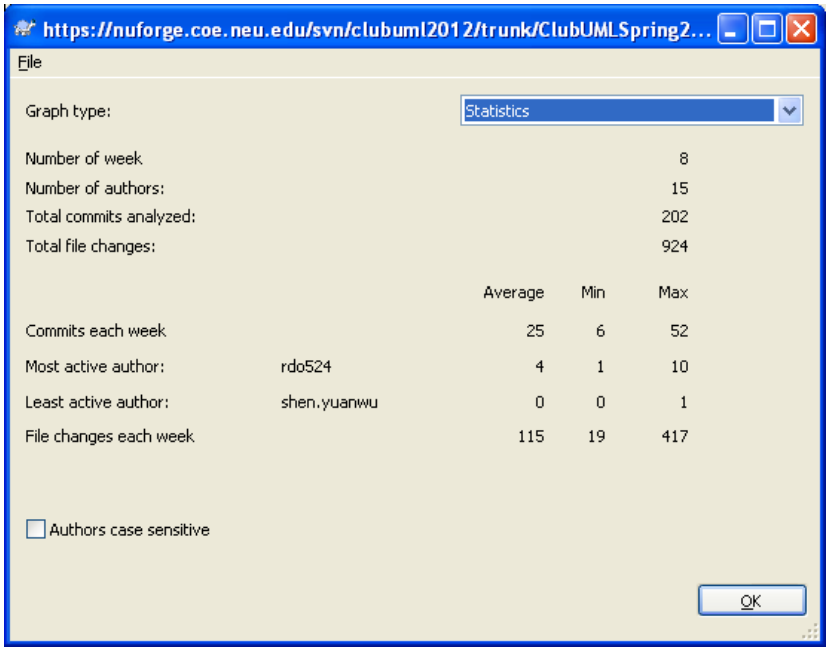


Figure 68: Spring 2013 SVN Statistics Diagram

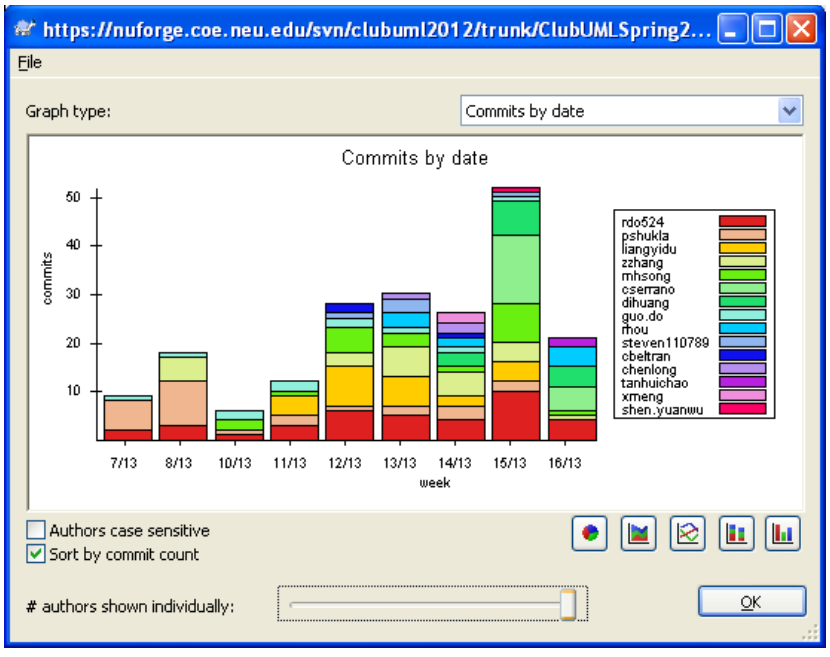


Figure 69: Spring 2013 SVN Commits Diagram

Total Lines of Code	7610
src	7610
controller.comparer.xmi	1511
controller	1045
controller.upload	987
controller.merge.xmi.xclass	931
compareAlgorithm	713
controller.comparer.xmi.request	619
repository	617
domain	334
uml2parser	307
controller.similaritycheck	267
controller.download	135
controller.compare	80
logging	43
controller.util	21

Figure 70: Spring 2013 SLOC Diagram

For comparison to last semesters work we computed their statistics and SLOC as well. They had a shorter span of commits with a range of 5 weeks compared to our 8 weeks. We started coding sooner which allowed us to see the problems sooner and adjust. Over the 5 week span, last year's group had 38 commits while we had 202 commits over the 8 weeks. For SLOC they had 2748 compared to our 7610.

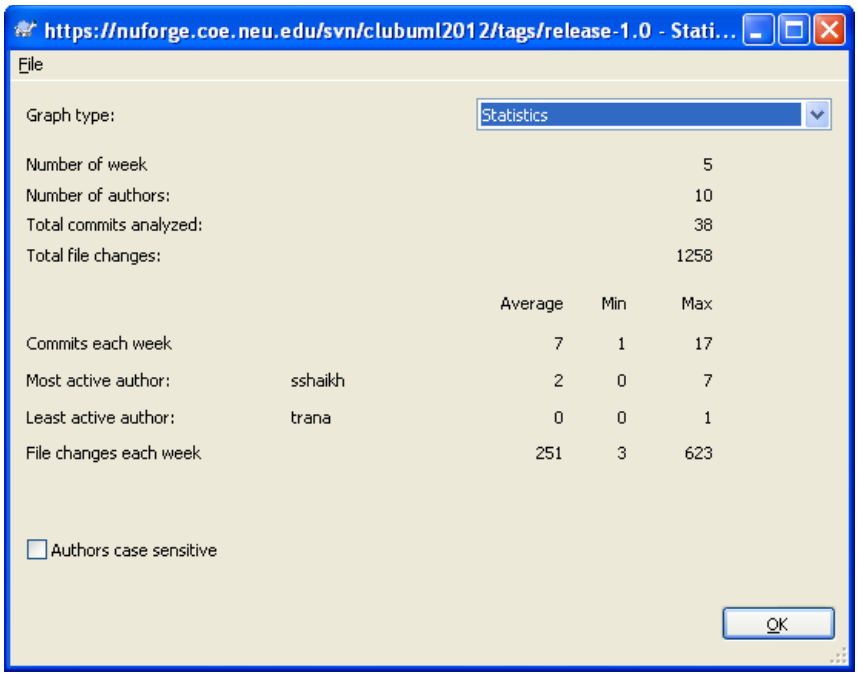


Figure 71: Fall 2012 Statistics Diagram

Total Lines of Code	2748
src	2748
controller	813
compareAlgorithm	712
repository	660
domain	320
controller.upload	243

Figure 72: Fall 2012 SLOC Diagram

Lessons Learned

We learned a lot going through and working together on this project. Some things we were able to identify and address correctly during the semester while others we did not till late but learned from it anyways.

As a Scrum master role, getting the team organized and focused was challenging as in any project. The team was spread out and everyone had different work schedules that made working together hard. Using the Blackboard and email helped. We also used TeamViewer software which allowed us to talk and show our work in real time with others through a computer. Meeting face to face outside of class also helped. We were able to work through problems and keep the team focused on the goals for that week.

Another problem we had was the direction early on. The team investigated and shared their numerous ideas but we were not able to put everything together until later. This made it seem like we were not making much progress at first. Work from one week was not always carried over to the next week and at times duplication would occur. It was not until the middle of the semester that we started making progress and coding started towards the goals.

Starting on an existing project is a lot of times harder than starting from scratch. For one, we needed to understand the existing project. The documentation and code from last semester did not always match. There were areas where the design document would say xyz existed but it did not. It was either out of date or never implemented. This made it harder for the team to pick up what the software was doing and contributed to the team delays in identifying what we wanted to achieve early on.

Suggestions for Future Work

There are several ways in which the ClubUML software could be improved in the future. Potential improvements include:

- Ability to run the software completely on a COE server
- Support Sequence Diagram merging
- Support Upload/Display/Merge of other UML diagram types
- Support Ecore Diagram merging
- Ability to generate a comparison report for two XMI diagrams
- Support a mid-merge save session for when the user may want to stop but continue later on
- Show the mid-merge diagram as it currently stands, updating after each change
- Improve commenting
- Display some revision history of merged diagrams to the user

Appendix A: ClubUML Local Machine Deployment

This appendix outlines the steps needed to install and run the ClubUML software locally.

Prerequisite Software

1. Java JRE 1.7
2. GraphViz (http://www.graphviz.org/Download..php)
3. MySQL Community Server (http://dev.mysql.com/downloads/mysql/)
4. MySQL Workbench (http://dev.mysql.com/downloads/workbench/)
5. TomCat 7 (http://tomcat.apache.org/download-70.cgi)
6. Tortoise SVN (http://tortoisesvn.net/downloads.html)
7. Eclipse IDE for Java EE Developers (http://www.eclipse.org/downloads/)

Install GraphViz

1. Install GraphViz to 'C:\Graphviz'. (The code will reference this directory when it attempts to generate UML class diagram images.)

2. After installing GraphViz, copy ‘cgraph.dll’ to ‘graph.dll’ under C:\Graphviz\bin folder. Don’t remove ‘cgraph.dll’ file. Both files need to create a png file.

Install MySQL Community Server

Once you install the Community Server, it'll take you through a MySQL Server Instance Configuration Wizard to setup a MySQL server. If this is the first time installing MySql Server then:

Use Standard Configuration.
Install As Windows Service
Set root password: 1234

> ***(Note: ClubUML source code for these instructions uses ‘1234’ as the password as of 4/22/13. This can be updated later for better security. If you decide to use a different password, the password can be changed in DbManager.java file.)***

Setup MySQL Workbench

1. Install MySQL Workbench.
2. Launch MySQL workbench.
3. Under the “Open Connection to Start Querying” section, select 'New Connection' at the bottom.

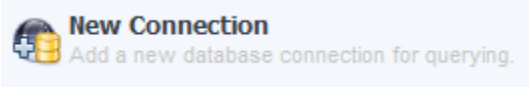


Figure 73: MySQL Workbench New Connection Button

4. Give the new connection a name (ex: clubuml) and press **OK**. Enter ‘1234’ when the password prompt appears. (This password was set in **Installing MySQL Community Server**)

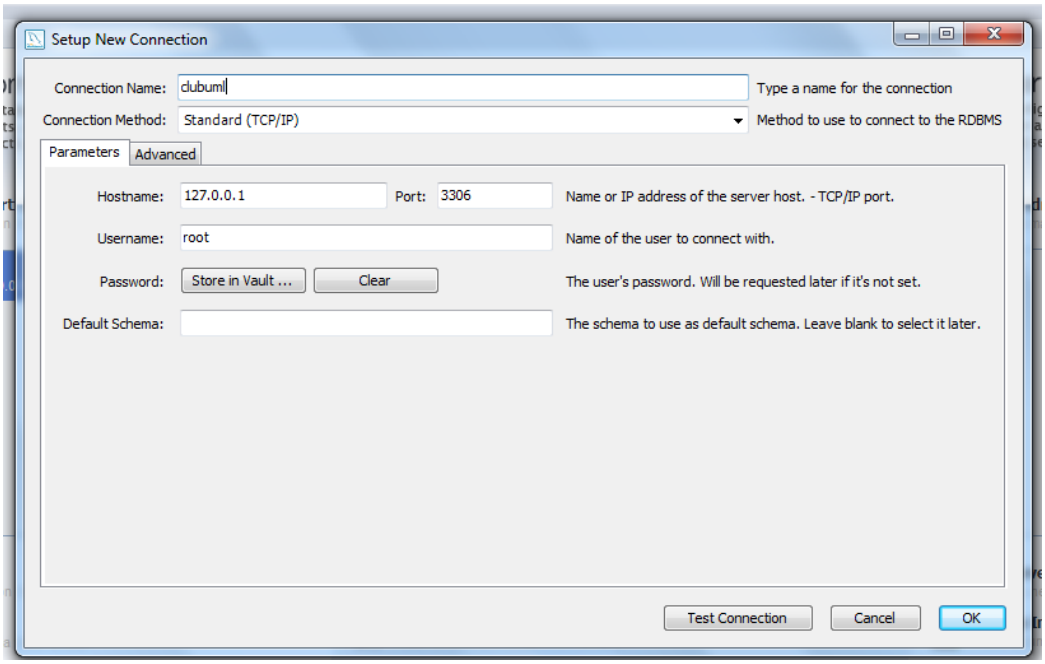


Figure 74: MySQL Workbench New Connection Setup

5. Open the new connection you just made. In the left pane (Object Browser), **right-click** any schema object and choose **Create Schema**.

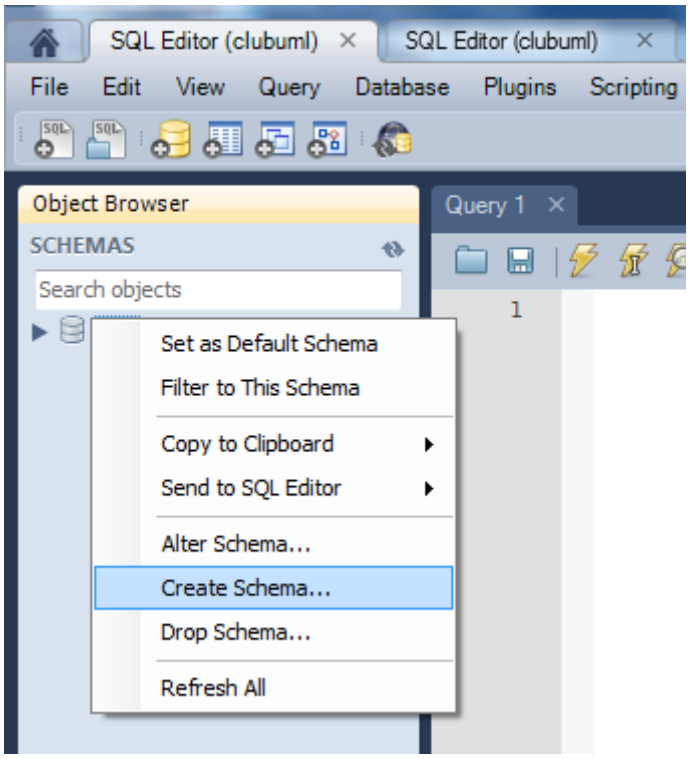


Figure 75: MySQL Create Schema

6. Enter “ClubUML” as the Schema name and hit **OK**.

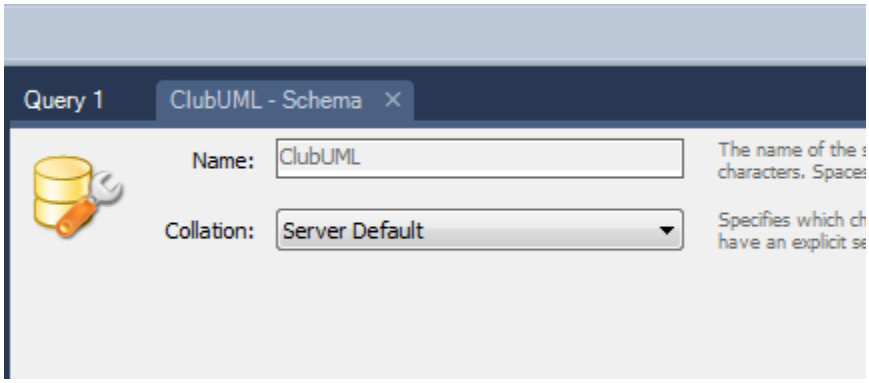


Figure 76: MySQL Schema Name

7. Go to **File->Open SQL Script** and copy in the query text from Appendix B: Database Query.

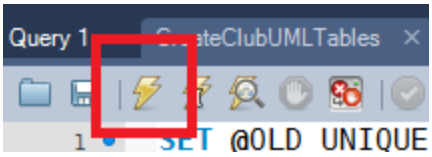


Figure 77: Query Execution Button

8. **Right-click** on ClubUML in the Object Browser and **Refresh All**. Tables should be available under ClubUML.

Troubleshooting MySQL Workbench

When installing MySQL Workbench, if “MSVCR100.dll is missing” error appears when installing MySQL Workbench, you will need to install Microsoft Visual C++ 1020 Redistributable Package (x86/x64) (http://www.microsoft.com/en-us/download/details.aspx?id=5555)

Setup Tortoise SVN

1. Install Tortoise SVN.
2. Open Windows Explorer and create a folder to store ClubUML repository.
3. **Right-Click** on the folder to store the repository**->SVN Checkout…**

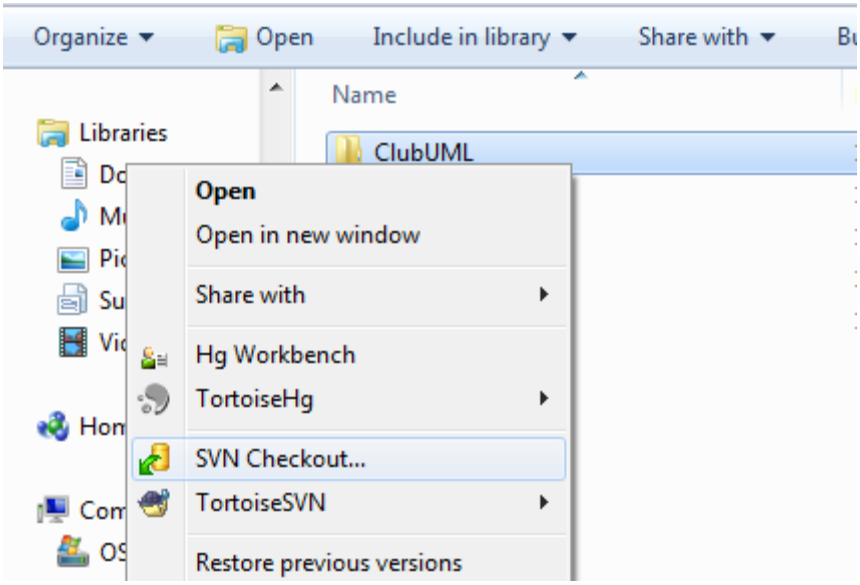


Figure 78: SVN Checkout Menu

4. Enter repository url https://nuforge.coe.neu.edu/svn/clubuml2012 and hit **OK**. Files will be imported.

Setup TomCat 7.0

1. Download TomCat's 32-bit/64-bit Windows Service Installer.
    a. **Make sure it is Tomcat 7.00 or 7.01. Newer versions (in particular 7.039) have caused errors – but could be fixed in the future.**
2. Installing TomCat 7.0 should be straight forward. Use default settings and enter the optional Administer user name and password to avoid editing the user access XML file later.

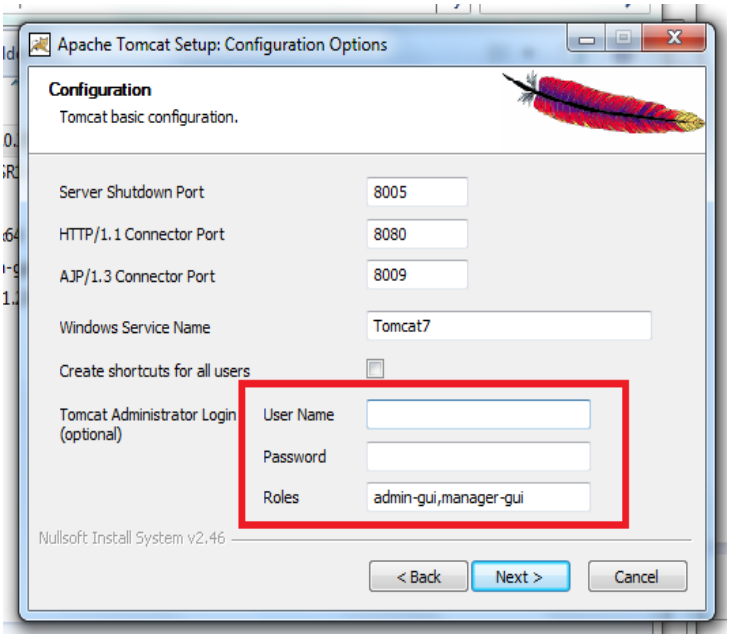


Figure 79: Tomcat User Setup

3. If you didn't follow step 2, then you have to edit the user access XML file to set a "User Name" and "Password" to enter http://localhost:8080/manager/html

   Open "Tomcat\conf\tomcat-users.xml". Add two lines into the file:

   <role rolename="manager-gui"/>
   <user username="XXX" password="XXX" roles="manager-gui"/>

```
<tomcat-users>
<!--
  NOTE:  By default, no user is included in the "manager-gui" role required
  to operate the "/manager/html" web application.  If you wish to use this app,
  you must define such a user - the username and password are arbitrary.
-->
<!--
  NOTE:  The sample user and role entries below are wrapped in a comment
  and thus are ignored when reading this file. Do not forget to remove
  <!.. ..> that surrounds them.
-->
<role rolename="manager-gui"/>
<user username="XXX" password="XXX" roles="manager-gui"/>
</tomcat-users>
```

Figure 80: Tomcat Config File

Setup Eclipse EE

1. Open Eclipse EE and in the menu bar: **File->Import…**
2. Select **General->Existing Projects into Workspace…**
3. Navigate to your ClubUml trunk folder imported from SVN and hit **OK**
4. A list of projects will be displayed. Only select the ClubUMLLocalTest project and hit **OK**.

(Note: There is also an SVN tool that can be downloaded in Eclipse's Market that can perform this task of importing from the respository.)

Troubleshooting Eclipse EE

If there are any issues, it is usually a library path problem. Right-click on *ClubUMLLocalTest* and go to properties. Go to Java Build Paths->Library and check if there are any issues. If there are issues, remap the libraries. Libraries can be found in *ClubUMLLocalTest/WebContent/lib*.

Create WAR file and Launch Webapp

1. The project will appear in Eclipse's Project Explorer,
2. **Right-click** *ClubUMLJan2013* folder**->Export->WAR file**

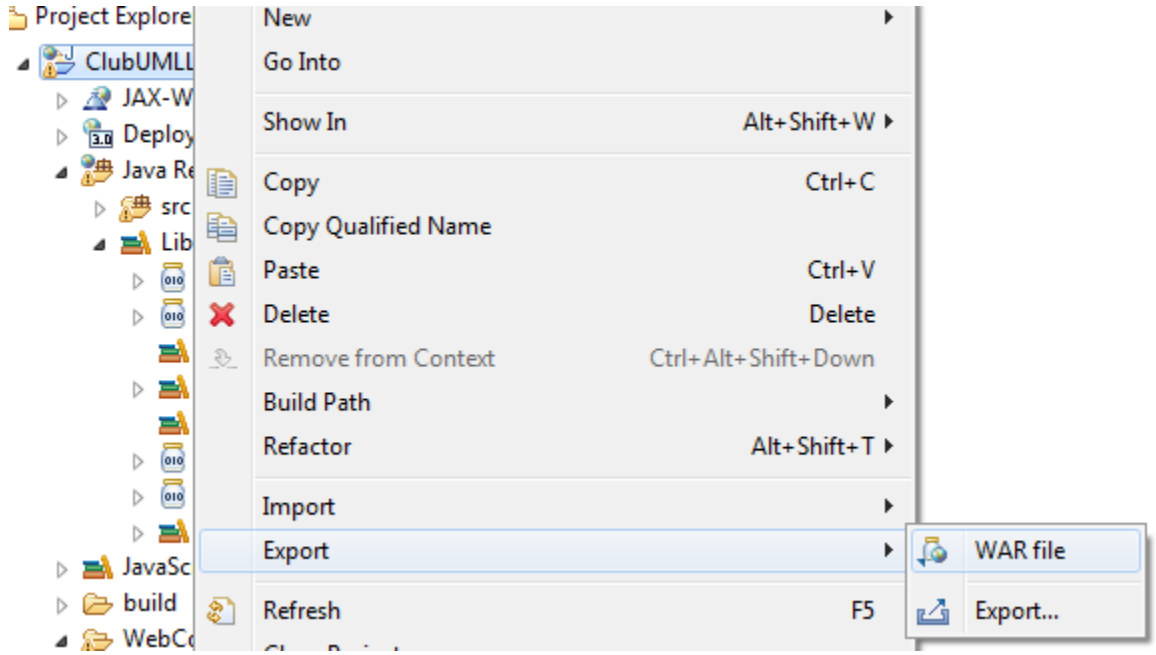


Figure 81: Exporting WAR File

3. Set the destination of the WAR file to your *TomCat7/webapp* folder. (Default directory: C:\Program Files\Apache Software Foundation\Tomcat 7.0\webapps)
4. Open a browser with URL: http://localhost:8080/manager/html
5. You should see a row for */ClubUMLLocalTest* with running = true.
6. Click on */ClubUMLLocalTest* and it'll launch you into the web program.

(Note: The project can also run in Eclipse using the "Run on Server" option, but I had an issue with file paths writing and reading in different directories.)

Troubleshooting Launch of Webapp

If http://localhost:8080/manager/html appears broken, make sure TomCat7 server is running. (Default directory: C:\Program Files\Apache Software Foundation\Tomcat 7.0\bin\Tomcat7.exe)

If error **HTTP Status 500 – controller/RegisterServlet : Unsupported major.minor version 51.0 (unable to load class controller...)** appears when attempting to register a new user or log in when running application, one solution is to uninstall and reinstall all Java versions. (link)

Web browser compatibility issue with Internet Explorer 7. Recommended browsers are IE 9+, Fire Fox, and Chrome.

Appendix B: Database Query

Here is the database query used to initially set up the MySQL tables used by the project:

```
/*
Created: 2013/2/15
Modified: 2013/3/24
Model: MySQL 5.0
Database: MySQL 5.0
*/

-- Create tables section -------------------------------------------------

-- Table project

CREATE TABLE project
(
  projectId Int(11) NOT NULL AUTO_INCREMENT,
  projectName Varchar(45) NOT NULL,
  starDate Varchar(45),
  desceiption Varchar(255),
 PRIMARY KEY (projectId)
)
;

-- Table user

CREATE TABLE user
(
  userId Int(11) NOT NULL AUTO_INCREMENT,
  userName Varchar(45) NOT NULL,
  email Varchar(45) NOT NULL,
  password Varchar(45) NOT NULL,
  securityQ Varchar(60),
  securityA Varchar(60),
 PRIMARY KEY (userId)
)
;

-- Table diagram

CREATE TABLE diagram
(
  diagramId Int(11) NOT NULL AUTO_INCREMENT,
  projectId Int(11) NOT NULL,
  userId Int(11) NOT NULL,
  diagramType Varchar(255),
  diagramName Varchar(255),
  createTime Timestamp NULL,
  filePath Varchar(255),
  fileType Varchar(255),
  merged Tinyint,
  notationFileName Varchar(255),
  notationFilePath Varchar(255),
  diFlieName Varchar(255),
  diFilepath Varchar(255),
 PRIMARY KEY (diagramId)
)
;

-- Table report

CREATE TABLE report
```

```
(
  reportId Int(11) NOT NULL AUTO_INCREMENT,
  diagramA Int(11) NOT NULL,
  diagramB Int(11) NOT NULL,
  mergedDiagram Int(11),
  type Varchar(20) NOT NULL,
  time Timestamp NOT NULL,
  reportFilePath Varchar(45) NOT NULL,
  reportFileName Varchar(45) NOT NULL,
 PRIMARY KEY (reportId)
)
;

-- Table comment

CREATE TABLE comment
(
  commentId Int(11) NOT NULL AUTO_INCREMENT,
  userId Int(11),
  reportId Int(11),
  content Varchar(255),
  writenTime Timestamp NULL,
 PRIMARY KEY (commentId)
)
;

-- Create relationships section ------------------------------------------------

ALTER TABLE diagram ADD CONSTRAINT diagramHaveOwnerId FOREIGN KEY (userId) REFERENCES
user (userId) ON DELETE NO ACTION ON UPDATE NO ACTION
;

ALTER TABLE diagram ADD CONSTRAINT diagramHaveProjectType FOREIGN KEY (projectId)
REFERENCES project (projectId) ON DELETE NO ACTION ON UPDATE NO ACTION
;

ALTER TABLE report ADD CONSTRAINT Relationship4 FOREIGN KEY (diagramA) REFERENCES
diagram (diagramId) ON DELETE NO ACTION ON UPDATE NO ACTION
;

ALTER TABLE report ADD CONSTRAINT Relationship5 FOREIGN KEY (diagramB) REFERENCES
diagram (diagramId) ON DELETE NO ACTION ON UPDATE NO ACTION
;

ALTER TABLE report ADD CONSTRAINT Relationship6 FOREIGN KEY (mergedDiagram) REFERENCES
diagram (diagramId) ON DELETE NO ACTION ON UPDATE NO ACTION
;

ALTER TABLE comment ADD CONSTRAINT Relationship7 FOREIGN KEY (userId) REFERENCES user
(userId) ON DELETE NO ACTION ON UPDATE NO ACTION
;

ALTER TABLE comment ADD CONSTRAINT Relationship8 FOREIGN KEY (reportId) REFERENCES
report (reportId) ON DELETE NO ACTION ON UPDATE NO ACTION
;
-- insert sample data---------------------------------------
insert into project value (1,'clubuml1',now(),'des1');

insert into project value (2,'clubuml2',now(),'des2');

-- auto-increment diagram
ALTER TABLE diagram AUTO_INCREMENT = 1;
```

Appendix C: Pic2Plot Setup

Pic2Plot Installation

Download Path:

http://gnuwin32.sourceforge.net/packages/plotutils.htm

Figure 82: Pic2Plot Download Page

Set Up Environment Variable:

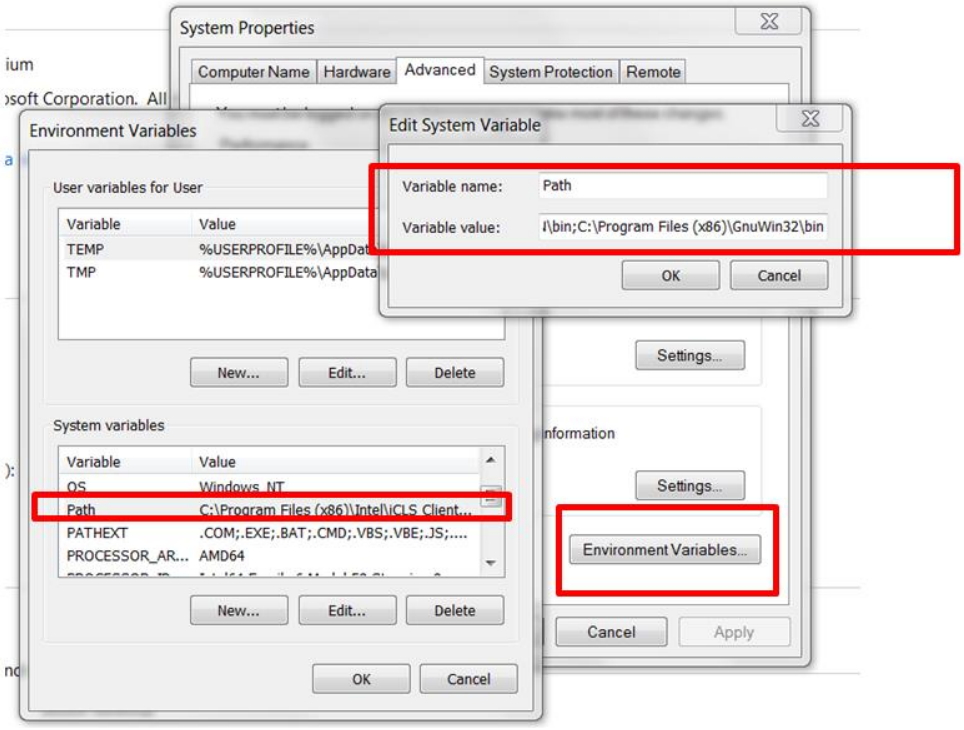


Figure 83: Pic2Plot Environment Variable Setup

Pic2Plot Server-Side Setup

The pic2plot program has been put under E:\clubuml2012\trunk\ClubUMLSpring2013\tools\ and has been checked into SVN.

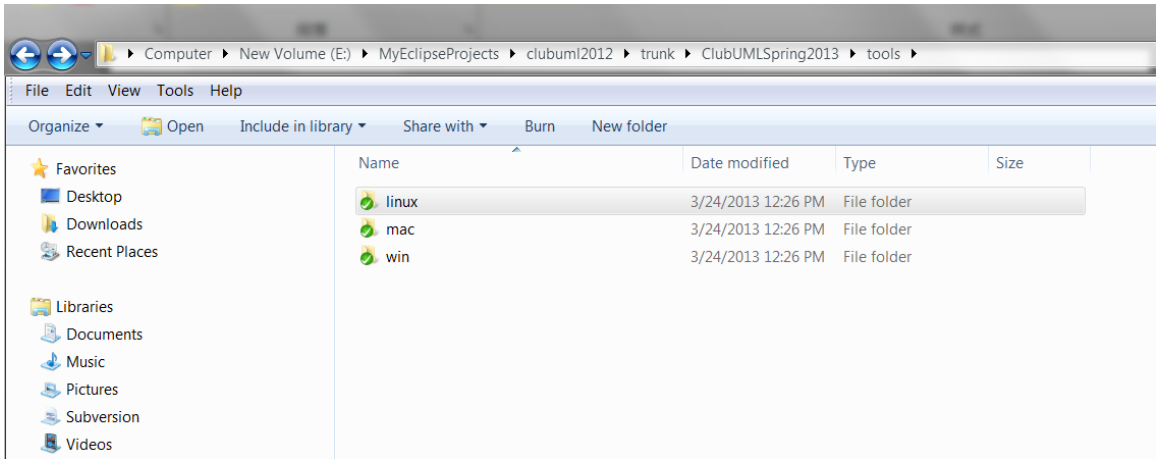


Figure 84: Repository tools Folder

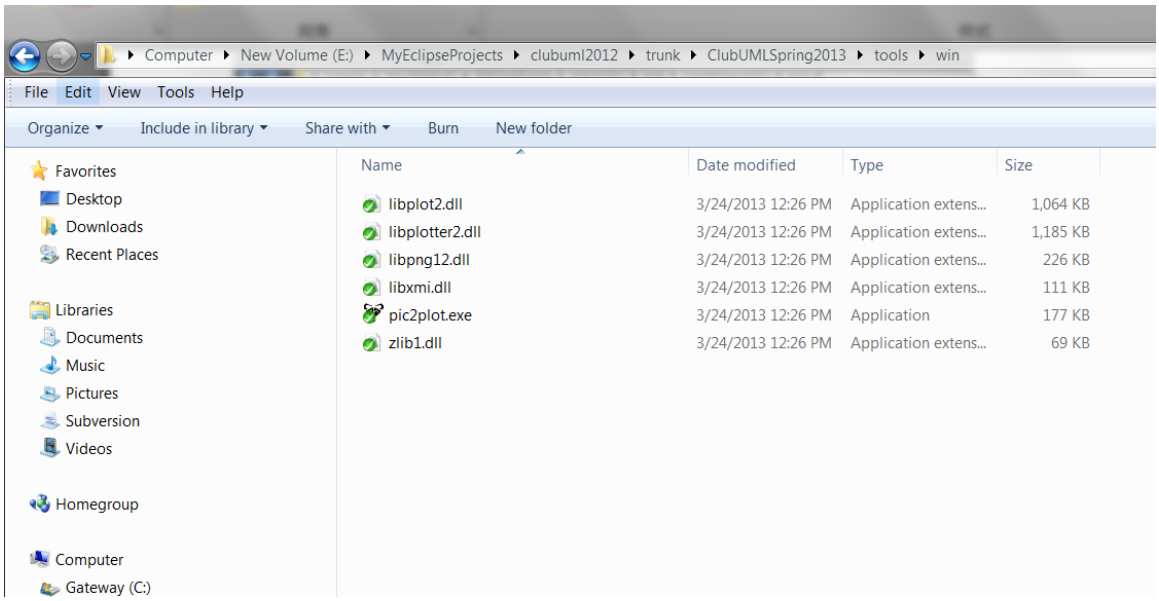


Figure 85: Pic2Plot in tools\win Folder

Appendix D: Glossary

Term	Description
Ecore	A type of file that is produced by the Eclipse Tools plugin. It allows for class diagram UML1.0 creation.
GIF	Graphics interchange format
GraphViz	Open source graph (network) visualization project from AT&T Research.
JavaBeans	Are reusable software components for Java. Practically, they are classes written in the Java programming language conforming to a particular convention.
JavaScript	An interpreted computer programming language, commonly used for scripting in web related files.
JDBC	Java Database Connectivity
JSP	Java Server Pages is a technology that helps software developers create dynamically generated web pages based on HTML, XML, or other document types.
Meta-model	The analysis, construction and development of the frames, rules, constraints, models and theories applicable and useful for modeling a predefined class of problems. For this project, it is related to XML.
MySQL	My Structured Query Language is an open source version of SQL
MVC	Model View Controller
OpenUP	Open Unified Process is an iterative agile software development process
PNG	Portable network graphics is a type of file for a picture.
Papyrus	An Eclipse plugin that allows for UML2.0 diagrams creation and exporting in xml.
Pic	A domain-specific programming language by Brian Kernighan for specifying diagrams in terms of objects, such as boxes with arrows between them.
Pic2Plot	This program takes one or more files in the Pic language, and either displays the figures that they contain on an X Window System display, or produces an output file containing the figures. Many graphics file formats are supported.
PDF	Portable Document Format
SAX Parser	A Java class that defines the API that wraps an XMLReader implementation class.
SQL	Structured Query Language used for database applications
UML	Unified Modeling Language
UMLGraph	Allows the declarative specification and drawing of UML class and sequence diagrams, open source.
XMI	XML Metadata Interchange
XML	Extensible markup language