SOEN 341 Project Document

Astou Chanel Aba Sene
Jean-Pierre Escalante
Ryan Held
Jenia Ivlev
Denis Lau
Marwa Malti
Amine Najahi
Rolf Schmidt

Table of Contents

Table of Figures

Table of Tables

1. Introduction

The students of the Software Process course SOEN 341 are asked to design an online academic scheduler for Concordia University students. This document deals with the development of such a project.

It first describes the project by detailing the functional and non-functional requirements, the goals and constraints and the technical and human resources. It then elaborates on the scope of the implementations, presents a solution sketch, the costs and a schedule for the development.

The document also describes the architectural design of the application using a 4+1 architectural view and the design of the subsystems and their interfaces. The logical view is represented by a design class diagram, the process view by an activity diagram. A component diagram is used to illustrate the development view while a deployment diagram shows the physical view. Scenarios (the '+1' in the 4+1 architectural view) are given in the form of a use case diagram. A detailed module diagram presents the internal structure of the subsystems and each class is briefly described.

Two use cases serve as dynamic design scenario examples. These examples include a list of system operations, contracts and sequence diagrams. A commented list of modules that have already been implemented is included in this document. As well, the initial graphical user interface (GUI) design is explained with respect to the various modules and the design process.

Moreover, this document explains all testing carried out on the final product. It presents the test coverage which shows what aspects of the system have already been tested along with the test cases used. While all requirements are tested, only two units are used as examples for unit testing. Stress testing is not actually performed, but a list of test designs is included to explain how the performance of the system in extreme usage situations could potentially be tested.

Furthermore, this document includes an installation and a user manual. The installation manual shows in detail how to install and execute the system and the user manual explains how to use the application while providing an easy to follow step by step guide that covers every feature of the system.

The final cost estimate is presented in tabular form to show the final costs associated with the documentation and development steps.

2. Project description

The purpose of the project is to create a simple to use and dynamic auto-scheduler that will provide its user with an array of potential schedules for a given semester. Based on courses a student needs to take in order to graduate from his or her program, the application will generate a series of schedules that will match the user's requirements for classes he wishes to take, taking into account personal time constraints and unavailability. From there, the user chooses one of the generated schedules that fits his/her preference best, and the application will save the schedule as well as the classes chosen for the semester, and display the schedule on demand. It also gives the user an option to print the schedule so that he need not check the site every time he wishes to know which class is next and where.

2.1. Deliverables

2.1.1. Deliverable 0

System Overview Document. This document is a very brief summary of the application that will be presented in the following report. It includes first and foremost, the domain, or rather the context and environment between the users and the application. It also includes a short description of the data files that will be used in order to generate the schedules, and attend to the functions of the application. Furthermore, there are a short description of the actual application interface and a diagram that shows off how it works. Finally, it includes the diagram of the functionalities and their relations in the application.

2.1.2. Deliverable 1

Scope and Plan Document. This document delineates the scope of the project along with all the capabilities and strengths. It also features the functional requirements and their use cases, in addition to the qualities it should have. This document sets into motion the plan to complete the project, from budget and schedules, tools used, human resources, the architecture of the application to the technologies used to build it. Finally, it includes details on the individual activities and the artifacts that they produce.

2.1.3. Deliverable 2

Project Design Document. This document is a high-level description of the architectural design in terms of modules, their purposes and interfaces, and diagrams and module interfaces specifications. It also comes with a detailed design diagram treating individual subsystems, dynamic scenario simulations, a rundown of prototype elements and their various stages of completions as well as risk analysis and scope changes.

2.1.4. Deliverable 3

This document will present an in depth analysis of both tested and untested items in the application. It is a report on various test cases on particular subsystems, test cases of functional requirements and stress tests to check for extreme situations and the application response. Moreover, it includes a user manual or a help menu as well as finalized estimations and a history log of all revisions done to previously submitted documents and the impact of the changes.

3. Goals and constraints

This section contains the functional requirements of the application presented in the form of a use case diagram and use cases. Each use case has an action, the actors causing it, the purpose of the action in summary, the trigger that activates such an action, the preconditions that lead up to the trigger, the post-conditions and a summary of the steps to achieve said action. One more characteristic for the use case is the importance of the use case, ranked on its indispensability. The next section is the UML standard domain model and this section finishes with the inclusion of non-functional requirements, qualities and the software or hardware constraints that would prevent users from being able to use this application to its fullest extent.

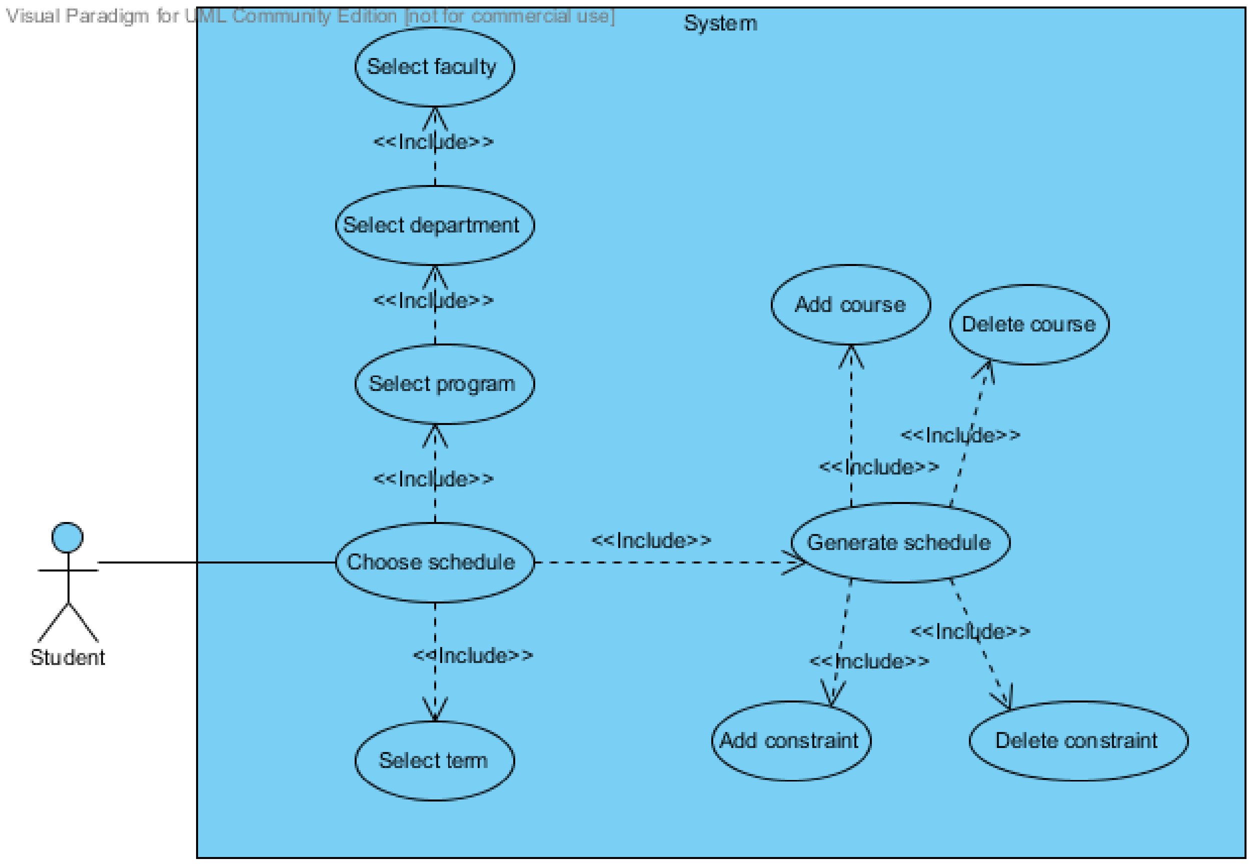


Figure 1: Use case diagram of the scheduler

Table 1. Use case Select program

Action	*Select program*	*Importance 5/5*	*Difficulty 4/5*
Actor	*Student*		
Purpose	*Select a program from a list depending on faculty and department*		
Trigger	*User selects a faculty*		
Preconditions	● *None*		
Post-conditions	● *All the courses required to complete the selected program have been displayed*		
Happy Flow	***User***	***System***	
	1. Selects a faculty	*2. Displays all the departments of that faculty*	
	3. Selects a department	*4. Displays all programs offered by that department*	
	5. Selects a program	*6. Indicates the selected program* *7. Displays all the courses of the selected program*	

Table 2. Use case Select term

Action	*Select term*	*Importance 5/5*	*Difficulty 2/5*
Actor	*Student*		
Purpose	*Select a term (Fall or Winter)*		
Trigger	*User selects a specific term from a choice of two (Fall/Winter)*		
Preconditions	● *Use case select program*		
Post-conditions	● *Courses offered in the selected term have been marked as available*		
Happy Flow	***User***	***System***	
	1. The user chooses a term	*2. Displays the courses of the selected term and program*	

Table 3. Use case Add student

Action	*Add course*	*Importance 5/5*	*Difficulty 1/5*
Actor	*Student*		
Purpose	*Add a course to the selected course list*		
Trigger	*User selects a course from the available courses and chooses 'add'*		
Preconditions	● *Use case select program*		
Post-conditions	● *The course selected by the student has been added to the selected course list*		
Happy Flow	***User***	***System***	
	1. Selects a course	*2. Indicates the selected course*	
	3. The user chooses 'add'	*4. Displays the selected course as part of the selected course list*	

Table 4. Use case Delete term

Action	*Delete course*	*Importance 5/5*	*Difficulty 1/5*
Actor	*Student*		
Purpose	*Delete a course from the selected course list*		
Trigger	*User selects a course from the selected course list and chooses 'delete'*		
Preconditions	● *Use case select program*		
Post-conditions	● *The course deleted by the student has been removed from the selected course list*		
Happy Flow	***User***	***System***	
	1. Selects a course from the selected course list	*2. Indicates the selected course*	
	3. Chooses 'delete'	*4. Displays the selected course list without the deleted course*	

Table 5. Use case Add constraint

Action	*Add constraint*	*Importance 2/5*	*Difficulty 5/5*
Actor	*Student*		
Purpose	*Add a time constraint to the schedule*		
Trigger	*User defines a time constraint and chooses 'add'*		
Preconditions	● *Use case select program*		
Post-conditions	● *The time constraint has been added*		
Happy Flow	***User***	***System***	
	1. Defines a constraint		
	2. Chooses 'add'	*3. Indicates the constraint has been added*	

Table 6. Use case Delete constraint

Action	*Delete constraint*	*Importance 2/5*	*Difficulty 5/5*
Actor	*Student*		
Purpose	*Delete a time constraint from the schedule*		
Trigger	*User selects a time constraint and chooses 'delete'*		
Preconditions	● *Use case select program*		
Post-conditions	● *The time constrain has been deleted*		
Happy Flow	***User***	***System***	
	1. Selects a constraint in the schedule	*2. Indicates the constraint has been deleted*	
	3. Chooses 'delete'	*4. Indicates the selected constraint*	

Table 7. Use case Choose schedule

Action	*Choose schedule Importance 5/5 Difficulty 5/5*	
Actor	*Student*	
Purpose	*Choose one schedule from all available schedules and save it*	
Trigger	*User picks one schedule and chooses 'save'*	
Preconditions	• *Use case select program* • *Use case add course*	
Post-conditions	• *The selected schedule has been saved*	
Happy Flow	***User***	***System***
	1. The user chooses 'generate schedules'	*2. The system displays the generated schedules*
		3. The systems prompts the user to select a schedule to save
	4. The user selects a schedule	*5. The system indicates that the selected schedule has been saved*

3.1. Functional requirements

3.1.1. Logging into the system

When a student wishes to use the application, upon opening it in a compatible web browser, he/she will be prompted to enter his/her identification number and password.

- If the password corresponding with the authentication ID is correct, the page will open to the main application page.
- If the password corresponding with the authentication ID is incorrect, the page will display an error message, and clear the ID and password fields.

3.1.2. Main page

This is the primary page of the application where most of the user interaction takes place. It includes the course display, term selection, and constraints window.

3.1.2.1. Course Display

The display area is on the page and it displays in blue the courses already taken and passed. As well, it displays in green the courses the student is eligible to take in this term. Finally, it displays in grey the classes the student cannot select due to prerequisite restrictions. This display automatically shows all available courses the student can take in a given term (Fall or Winter) depending on completed courses.

- Students can add courses for general electives into the course display from a list. General electives are courses outside of the requisite for a program's completion.

- Students may choose to disable/hide/remove certain courses from the list of available classes. When generating a schedule, these classes will not appear in the generated schedules.

- Tooltips: Hovering over available classes with the mouse will give more details about the class, including classroom, professor teaching the class, laboratories and tutorials associated with the class.

3.1.2.2. Term selection
A small option that toggles whether the courses to be added are from the Fall or Winter session.

3.1.2.3. Constraints
A time constraint window that allows a student to specify what time constraints he would like to impose on the schedule. This functions by specifying the times and days where the student would like to avoid having classes.

3.1.2.4. Course load
Student specifies how many courses he or she wishes to take in that semester. Part time students are restricted to a maximum of 3 courses per semester and full time students must take at least 4 courses per semester with a maximum of 7 courses. Generated schedules will be based on how many courses the student wishes to take.

3.1.2.5. Generate schedule
A single button that will generate an array of schedules for the student to choose from based on the courses available and the time constraints entered.

- All generated schedules will appear on the same page without forcing the user to go back and forth to choose.

- When the application generates no schedules, it will generate an error message prompting the student to change his or her availabilities and/or disable/hide/remove classes from the course display.

- A user can select and compare an array of generated schedules and choose one to become his/her actual schedule for the given semester. Once confirmed, courses selected for the schedule will be highlighted in yellow in the Course Display box.

3.1.2.6. Drop course
Once a schedule has been confirmed, the student has the option to drop a course he or she does not feel compelled to complete. If this is done, the schedule is automatically updated.

3.1.3. Logout
Logging the user out of the system and exiting the application.

3.2. Domain model

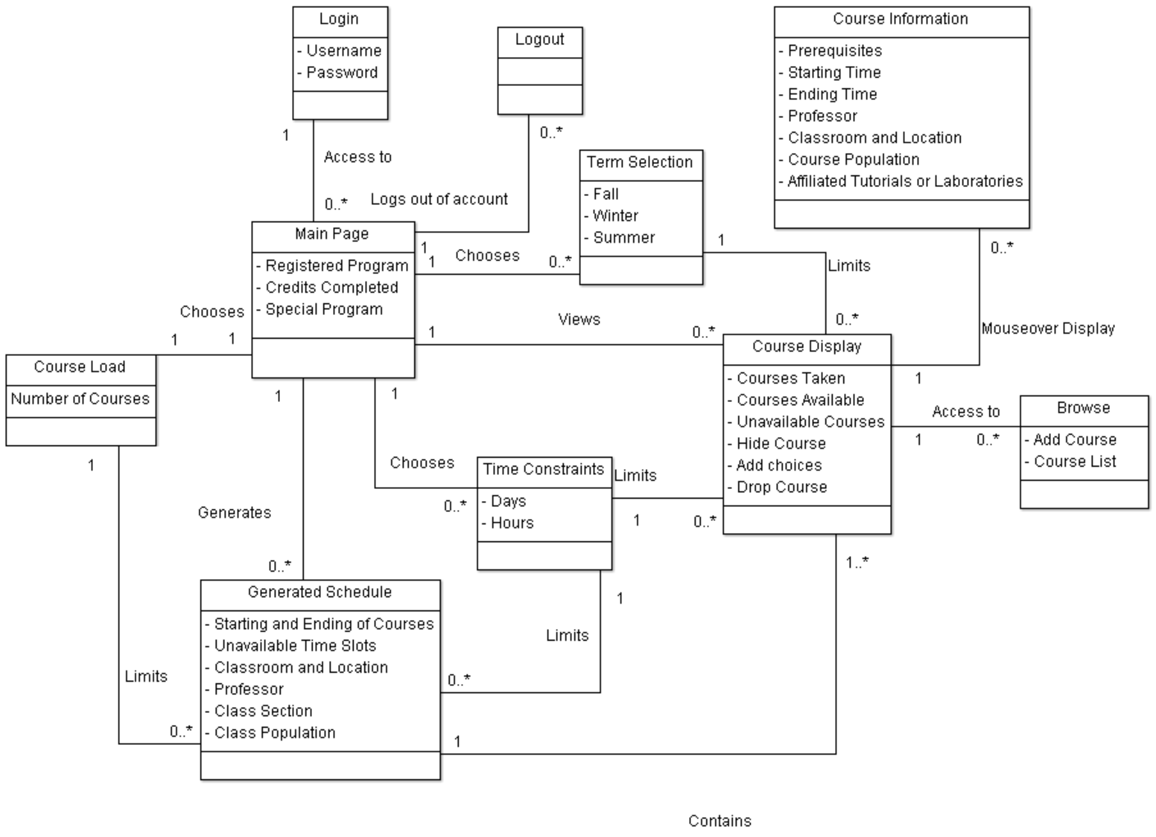


Figure 2. Domain model

3.3. Qualities

3.3.1. Performance

Schedule generation should not exceed 30 seconds after analyzing all valid inputs.

- Error: should the delay exceed 30 seconds, a message will appear to reassure users that nothing is wrong, or warn users to try again if the operation failed.

3.3.2. Usability

The application should be easy to use and intuitive.

- The user interface design is simple with large windows, few options and clutter on the main application page. The personalization of the schedule should not be arduous.
- Context-sensitive help is available as tooltips in order for the users to navigate and understand the options they can choose from while creating their schedule.
- The courses in the course display window are color coded depending on whether or not a course has already been taken, is available, or not yet available.

- Large fonts make for an easy read, and for more information, there are tooltips for course descriptions and detailed explanations for the options.

3.3.3. Reliability

The application should be accessible at all times except for planned maintenance periods. Maintenance periods should be at regular intervals, and the application itself should remind users the day before that there will be an outage period. Maintenance should be done at hours that would affect the least amount of users.

3.3.4. Accessibility

Users should be able to easily access the application.

- Browser: users should minimally still be able to access the application through the latest versions of Internet Explorer and Firefox. Further accessibility through other popular browsers such as Google Chrome and Safari are planned.

- Operating System: users should be able to access the application when using Windows XP, Windows Vista or Windows 7 at its most basic level. Compatibility with Linux, Mac OS, Ubuntu or other operating systems might be implemented at a later date.

3.3.5. Security

Basic login security. User IDs should not be shared.

- Password length: passwords should have a minimum length of 6 alphanumeric characters and a maximum of 12 characters.

- Password choice: when first setting up the password, there must be at least one number and one capitalized letter in the password. A pop-up also appears and discourages users from using their birthday or names as their password, but the application should not restrict them from doing so.

- Login fail protection: when attempts to log in with an erroneous password fail 5 times straight within a certain time period, the system will lock login attempts for an hour to discourage would-be account thieves from hard guessing user passwords.

- Locked accounts: when an account becomes locked, the application should send an email to the address registered in the account to re-open or reconfirm the user password.

3.4. Constraints

3.4.1. User constraints

Users cannot use the application without internet access.

3.4.2. Resource constraints

The actual experience in the field for most people in the project is non-existent. Though everyone does know a fair share in terms of programming language, not all of us have database or website design experience.

3.4.3. Browser compatibility

For the moment, compatibility for browsers other than Internet Explorer and Firefox has not been implemented. Also, the application must be run on fairly recent versions of Firefox and Explorer, as it is likely not able to support old versions of said browsers.

3.4.4. Platform requirements

Currently, our application is not optimized to run on Mac OS or Linux operating systems. Some requirements are untested and may not meet expectations, though they should function fairly well regardless.

4. Resource evaluation

This section focuses on the physical, technical and human resources available to our team and what we can do with all of the above. It also scopes out the project in terms of how much we can work with and whether or not we have the resources necessary to complete certain aspects of the project.

4.1. Human resources

Here is an overview of the team members working on this project, their unique abilities and experiences, what they contribute to the project as well as their availabilities and/or the amount of time they can spend in this project per week.

Table 8. Human resources

Astou Chanel Aba Sene	
Role:	Documentation
Knowledge:	C++, Assembly language, VHDL
Experience	3rd year student Computer Engineering Program, Quality Assurance testing
Availability:	Monday, Tuesday, Wednesday, Thursday, 1:00-4:15 pm, and weekends.

Jean-Pierre Escalante	
Role:	Documentation
Knowledge:	C++, Java, Assembly, HTML, VHDL
Experience	School projects
Availability:	Weekends, by request on week days.

Ryan Held	
Role:	Team leader, documentation
Knowledge:	C++, Java, HTML
Experience	Was on the organizing committee for a summer festival with 600 in attendance, and currently working on implementing database software for a non-profit company.
Availability:	At least 6h a week, and when needed.

Jenia Ivlev	
Role:	Programmer, architect
Knowledge:	C++, Java, CSS, HTML, PHP and Python
Experience	Programming , working in teams
Availability:	Monday and Thursday evening, weekends.

Denis Lau	
Role:	Team Coordinator, Relief Leader, Documentation.
Knowledge:	C/C++, Java, NXC embedded programming language.
Experience	Project management for other projects.
Availability:	In excess of 20h a week.

Marwa Malti	
Role:	Documentation/Web Design
Knowledge:	Java, HTML, PHP, CSS
Experience	2nd year student Software Engineering Program
Availability:	Tuesdays, Thursday or Friday between 2:45 to 5:30 pm.

Amine Najahi	
Role:	Programmer & System Administrator.
Knowledge:	C/C++, C#, Java, PHP, HTML, CSS, JavaScript, Perl, TCL, MySQL
Experience	4 years of professional programming in embedded systems, real-time system and high-level application software.
Availability:	10 h a week.

Rolf Schmidt	
Role:	Programmer
Knowledge:	Delphi, Java, C++, XML/XSL, PHP, HTML, MySQL
Experience	Work experience
Availability:	At least 6h a week.

4.2. Technical resources

4.2.1. Documentation

Google Docs, Microsoft Word 2007, Microsoft Word 2010.

All of the above are word processors that allow several users to collaboratively work on the same document. We will use any (and all) of the above to work on the submission documents. Final formatting will be done in Word.

4.2.2. SVN server

Privately set-up SVN server, Google code.

There are many version control systems available, both open-source and commercial. Given the familiarity of most group members with Subversion, we will be using a privately set-up SVN server and/or Google code.

4.2.3. Programming languages

JavaScript, PHP, SQL.

A good number of dynamic websites uses the combination of JavaScript, PHP and SQL. Most of our group members are familiar with at least one of these technologies and hence the choice to use this combination.

4.2.4. Hardware

ENCS computers, personal computers and laptops.

Development will happen at the university on ENCS computers or student laptops and at home on student laptops and/or desktops.

5. Scoping

Our prospective system will allow students enrolled in the B. Eng. in Software Engineering program to choose the courses that they want to take in a given semester and define time constraints. A set of schedules will automatically be generated for the chosen courses and selected time constraints.

In the following, the details of the functionality we intend to include will be given and the functionality that will be excluded will be described.

5.1. Included features

5.1.1. Description

After login into our system, individual student information such as status (full/part-time) and courses taken will be available. This information will be used to color-code the display of courses, distinguishing between completed, available (i.e. prerequisites are completed) and not available courses. The student will then choose from the available courses list (adding them to the selected course list) and impose a set of time constraints and a set of schedules will be generated for the student to choose from.

The schedules will display the course information and time constraints. The student can then decide to drop courses and/or time constraints. The schedule display will automatically update with every change.

The system can also be used by guests without login. In this case, all courses will be marked as available as prerequisites cannot be checked. This feature will be useful for informational purposes, e.g. when looking for a classroom of a specific class.

5.1.2. Detailed list of included features

- Login/logout
- Set term credit limit based on student information
- Select term (fall or winter)
- Dynamic display of all program courses in a tabbed interface
- Color-code courses as completed, available and not available
- Add available course to selected course list
- Remove course from selected course list
- Clear selected course list
- Add time constraint (day, start time, end time) to schedule
- Remove single time constraint from schedule
- Remove all time constraints from schedule
- Display total number of generated schedules
- Browse available schedules (backward and forward)

5.2. Excluded features

It seems reasonable to integrate our system with MyConcordia with respect to student information and for schedule generation and the Concordia registration system. This is clearly outside of the purview of this project though.

On the other hand, the dynamic nature of our system will allow extending it to other programs like the Bachelor of in Computer Science with relative ease or other Engineering degrees. Dropdown menus to choose the faculty, department and program will be included in our system to indicate this.

We will not include the possibility to create a schedule for both semesters or for the Fall/Winter term or the possibility to suggest courses to offer depending on the students enrolled in the program.

6. Solution sketch

6.1. Architecture

The architecture of our design will be based on the Model, View, Controller (MVC) software design pattern. The strategy will allow the program to be divided in an intuitive and ordered manner, plus it will reduce development time and increase maintainability. The MVC design pattern is often used in applications that need the ability to maintain multiple views of the same data. In fact, the MVC establishes a clean separation of objects in three categories - the model for maintaining data, views for displaying information and the controller for handling events.

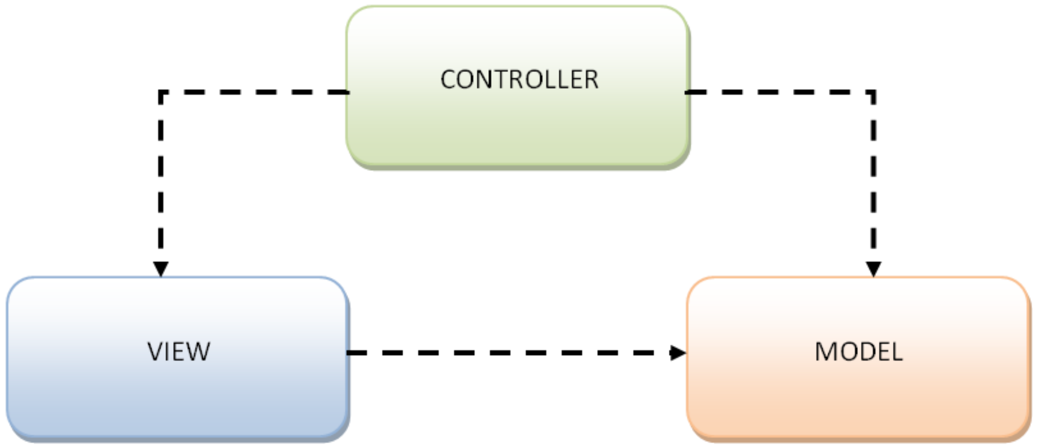


Figure 3. MVC architecture

6.1.1. Model

The model will contain all the information on the courses, students, passwords, restrictions, program information etc. In our case, the model will be implemented using a MySQL database. The model is a passive object, which means that it will only react to requests from the controller or the view. Usually, only the controller has write access to the data inside the model, while the views have read access.

6.1.2. View

In our application we will only have one view and it will be a jQuery, CSS and HTML based GUI. The view will fetch information from the model after it has been notified by the controller.

6.1.3. Controller

The controller that we will be designing is a simple object that handles event from the user interface. Its main purpose is to keep the model and the view synchronized. The controller will notify the view when an event is triggered and it will also update the model information. The controller will mainly be coded in jQuery, a JavaScript library

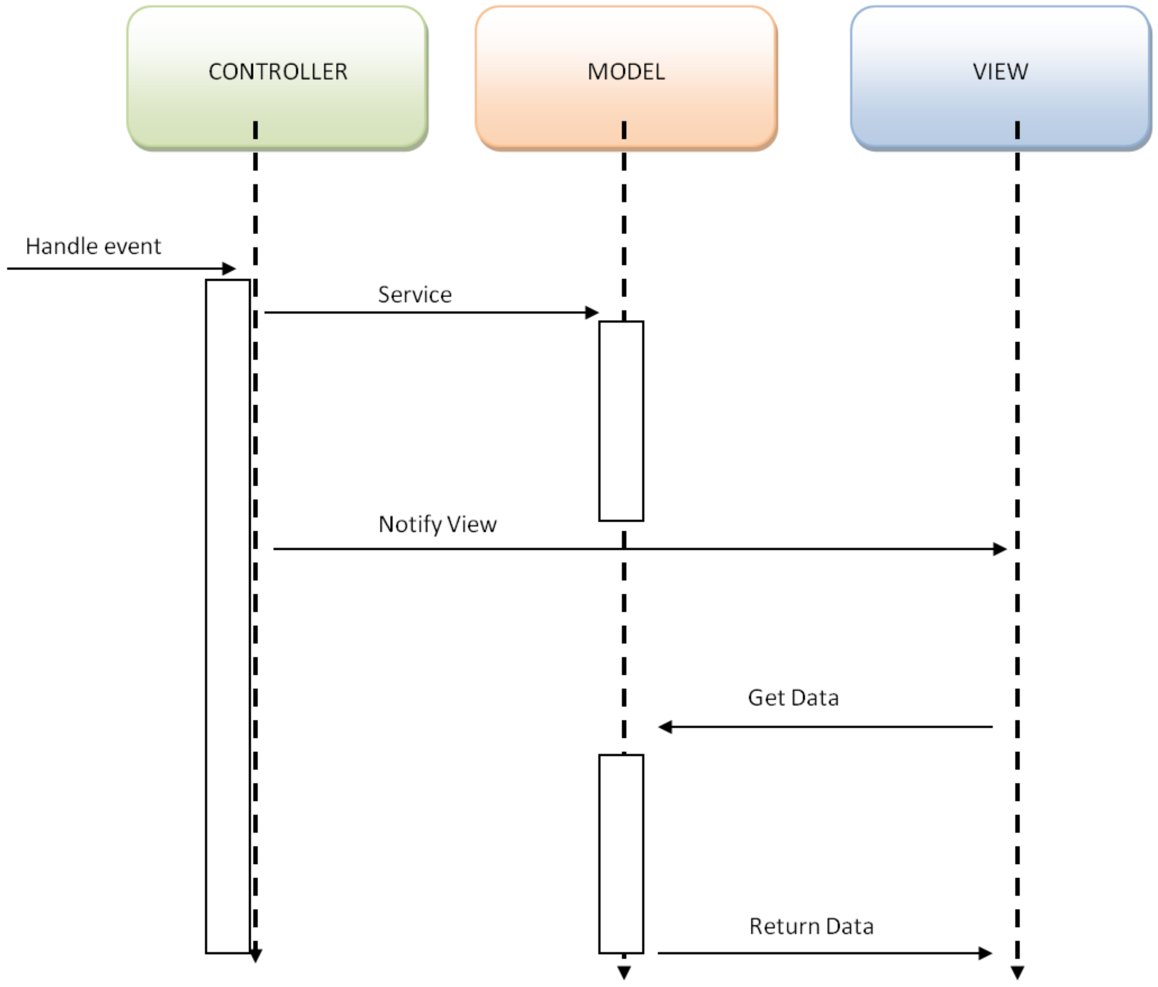


Figure 4. MVC control flow

6.2. Technologies in use

6.2.1. XAMPP

XAMPP is an easy to install Apache distribution containing MySQL, PHP, PERL and all the tools needed to host and administrate a private server. This technology was used to setup the working environment for coding, testing and debug across the entire team.

6.2.2. TortoiseSVN

TortoiseSVN is a software versioning and revision control system. We used an SVN server and client to share and control files during the entire development process. The SVN server was integrated inside XAMP to allow all the developer to modify sources, commit files and tests the application using any browser.

6.2.3. ProjectPier

ProjectPier is a free, open-source, PHP application for managing tasks, projects and teams through an intuitive web interface. ProjectPier was installed inside the XAMP web server. ProjectPier is intended to organize, communicate, collaborate and get things done in a similar way to professional project management methodology.

6.2.4. PHP

PHP is a general-purpose server-side scripting language designed to produce dynamic web pages. For this purpose, PHP code is embedded into HTML source documents and interpreted by a web server with a PHP processor. PHP is free and easy to use, it also comes with plenty of libraries with APIs that help web developers to accomplish tedious tasks.

6.2.5. JavaScript

JavaScript is a client side scripting language used to create dynamic web pages. Its main goal is to interact with the user actions and send HTTP requests to the server.

6.2.6. CSS

CSS is a style sheet language used to format a document written in a mark-up language. In our application CSS will give a predefined look to our HTML and XHTML web pages. By using CSS we are separating the document content from the document presentation such as layout, colors and fonts, thus giving more flexibility and reducing complexity of the design.

6.2.7. Adobe Photoshop

Adobe Photoshop is a commercial graphics editing software used to create, edit and modify multimedia content.

6.2.8. jQuery

jQuery is a JavaScript Library that simplifies HTML document traversing, event handling, animating, and Ajax interactions for rapid web development. The jQuery library will be used to handle all the effects and events from the GUI.

6.3. Database design

The database design is shown as an entity-relationship diagram in Figure 5. There are 12 tables that contain information about students, courses and schedules. The adjacency list contains information about the prerequisite structure and can be used to generate a directed acyclic graph of the course sequence for a program.

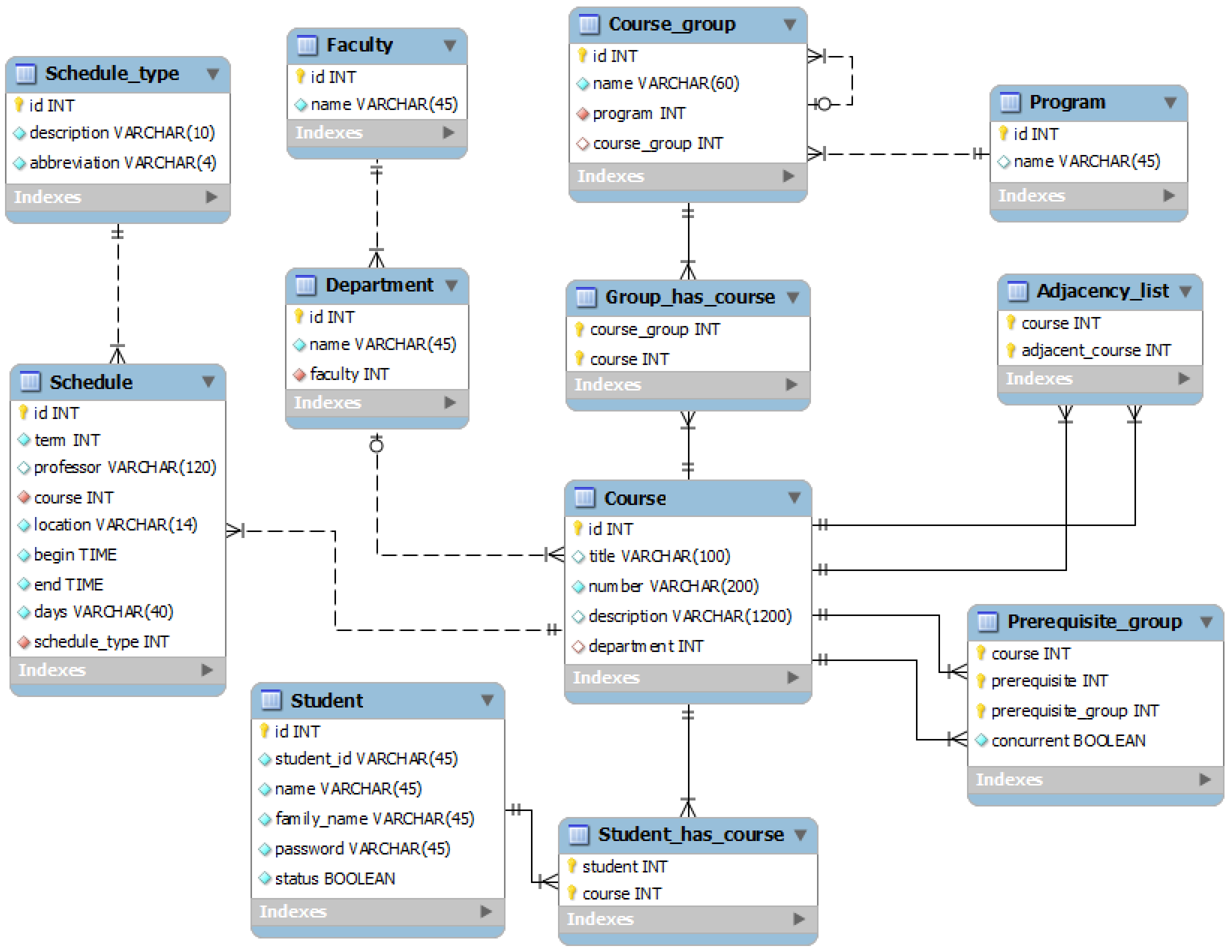


Figure 5. ER diagram of the database

6.3.1. Data sources

The data required to populate the database tables was obtained from two sources: the Concordia University undergraduate calendar and the Concordia University class schedule.

6.3.1.1. Concordia University undergraduate calendar

The calendar contains information on which courses are required to fulfill program requirements, which options and electives are available, as well as general course information such as title, description, number of credits, prerequisites, etc. Moreover, it contains information about restrictions to those outside of the program, as well as course sequences for students in special programs like the Co-Op program, or independent students.

6.3.1.2. Concordia University class schedule

The class schedule gives specific course information, in particular the number of sections, laboratories and tutorials for a course, the scheduled times and location and the name of the professor teaching the course.

6.3.1.3. Student information

For obvious reasons, student information is not freely available. We will create student profiles based on the course sequence from the undergraduate calendar for 1st, 2nd and 3rd year students, both full and part-time. There are three options to choose from in the BEng in Software Engineering program: Computer Games (CG), Web Services and Applications (WSA) and Real-Time, Embedded, and Avionics Software (REA). Student profiles will include examples of all three options.

The profiles will contain information such as passed courses, program, option, status, login and password.

6.3.2. Data extraction

Two utilities were written in Delphi to semi-automatically extract information from the undergraduate calendar and the class schedule into an XML file. Prerequisites needed to be extracted manually as there are subtle inconsistencies in the format that make fully automated extraction difficult.

The XML file was then transformed with XSL to output SQL scripts to populate the database tables. This approach is quite flexible and is easily adapted if the database design changes.

7. Plan

7.1. Activities

7.1.1. Phase 1: Systems overview document

7.1.1.1. Activity 1

Email exchange. The initial 5 team members from the group exchanged email addresses. The group size increased afterwards, so only 5 of 9 could be reached at the given time.

7.1.1.2. Activity 2

Individual work on the systems overview document. As we did not know each other or our experience yet, we simply assigned tasks to those who were present at the time.

Output: Parts of the systems overview document.

7.1.1.3. Activity 3

Collate and assemble completed tasks into one formal document. All assigned tasks were submitted to one member to assemble into a single cohesive document.

Output: Systems overview document.

7.1.2. Phase2: First team meeting

7.1.2.1. Activity 4

Introductions and set up communications. We introduced ourselves to each other with the goal of sizing up what kind of experience and knowledge we can pool together. Then we exchanged email addresses and contact numbers between key team members. We also settled on a team leader and established a Google group to share data and work.

Output: Google Group, communications channel, human resources.

7.1.2.2. Activity 5

Set up group framework. Assigning temporary positions based on experience, program, knowledge and availabilities of team members, and organizing future team meetings based on everyone's availability. Amine volunteered to set up an SVN server for the group, and Jenia chose to work on setting up the database.

Output: Tasks assignment and availability schedule.

7.1.2.3. Activity 6

Discussion of project requirements and brainstorming. We discussed ideas, and sorted through those that can be handled and implemented as soon as possible. We also started our preliminary design on what our end product should do.

Output: Preliminary design.

7.1.2.4. Activity 7

Decide on technical resources and generalized scope. We chose to use PHP, MySQL and JavaScript as most team members are familiar with the above, making it easier for others to work together.

Output: Determined scope.

7.1.3. Phase 3: Second team meeting

7.1.3.1. Activity 8

Finalized design. Put the finishing touches on our preliminary design and decided on the functional requirements and qualities of the application.

Output: Design prototype.

7.1.3.2. Activity 9

Dividing the work for the Scope and Plan Document. Assign sections of the Scope and Plan document that needs to be done amongst those not primarily engaged in the creation of SVN server, the databases and the preliminary design.

Output: Work division.

7.1.3.3. Activity 10

Completed SVN server. Walk-through of the capabilities and functions of our SVN server and tutorial.

Output: SVN server.

7.1.3.4. Activity 11

Completed databases. Completion of the various databases we will require in order to properly run the application. Also a rundown of the uses for each database and how to use them within our application.

Output: Student database, course list database, program requirement database.

7.1.4. Phase 4: Coding the application

7.1.4.1. Activity 12

Implementing basic application interface. Starting to code some of the functional requirements of the application. This includes the displays for the main page, including the course display, term toggle button, constraints window and course load and login/logout.

Output: Basic application framework completed.

7.1.4.2. Activity 13

Implementing classes and functions. Start coding the functional parts of the application. Ultimately should be able to generate at least one schedule.

Output: Completed application, but untested.

7.1.5. Phase 5: Third deliverable

7.1.5.1. Activity 14

Completion of detailed design. Completion of the in depth class diagrams and the various explanations and clarifications, as well as the unit description of every class and the dynamic design scenarios.

Output: Detailed design portion complete.

7.1.5.2. Activity 15

Completion of architectural design. Completion of the architecture diagram and the rationale behind it, as well as the subsystem interfaces specifications and the description for their parameters.

Output: Architectural design portion complete.

7.1.5.3. Activity 16

Prototype, risk report and completed report. Complete the report of prototype functions and whether or not they will be scoped out ultimately by the end of the project given current time constraints. Completion and collation of the third deliverable.

Output: Completed architecture and design document.

7.1.6. Phase 6: Product testing

7.1.6.1. Activity 17

Testing the final application. Hard testing the application's functional requirements. Verify implementation versus requirements.

Output: Requirements and function progress.

7.1.6.2. Activity 18

Product debugging. Testing application for errors and problems that prevent normal use of the application and fixing them.

Output: Error Log.

7.1.6.3. Activity 19

Quality testing. Testing non-functional requirements like usability, reliability, performance and security.

Output: Quality assurance log.

7.1.7. Phase 7: Final report

7.1.7.1. Activity 20

Finalized report. Unifying all parts of the final submission worked on by the various members and combining all previous submissions into a cohesive and complete document. User manual and installation manual included.

Output: Final Document.

7.2. Artifacts

7.2.1. Systems overview document

This is deliverable 0. The systems overview document is the introduction of what the team plans to do and how it will be done.

- Domain description
- Data files
    - *i.* Concordia University's course calendar
    - *ii.* Course sequence list
    - *iii.* Student information database
    - *iv.* Generated schedules
    - *v.* Student schedule preferences
- Interface
    - *i.* Inputs
    - *ii.* Interaction
- Block diagram
- User interface
- Team members

7.2.2. Availability schedule

The availability schedule gives the team leader a view of what are the optimal times to have meetings.

7.2.3. Quality assurance log

- Report of system defects
- Report of system security features
- Report of system algorithm

7.2.4. Error log

- Test plan
- Test results
- Statistical results

This set of artifacts, called quality assurance log, are used to assess the system's reliability.

7.2.5. Preliminary design

The preliminary design is the design the team members discussed and agreed to try and achieve at the end of the project.

7.2.6. Design prototype

The design prototype is the finalized version of the preliminary design including functional requirements and the visual aspect of the desired final product.

7.2.7. Scope and plan document

This document is the first deliverable. It describes the range of the system. It also contains and explains how the project will be carried out.

- Introduction
- Project description
- Goals and constraints
  - Functional requirements
  - Domain model
  - Qualities
  - Constraints
- Resource evaluation
  - Human resources
  - Technical resources
- Scoping
- Solution sketch
  - Architecture
  - Technologies in use

- Plan
    - Activities
    - Artifacts
    - Project estimates
    - Activities assignments
    - Schedule
    - Risks

7.2.8. Architecture and design document

This document is the second deliverable. It gives a very detailed description of the design and will be the base for implementing the project.

- Introduction
- Architectural design
    - Architecture diagram
    - Subsystem interface specifications
- Detailed design
    - Detailed design diagram
    - Units descriptions
- Dynamic design scenarios
- Rapid prototyping and risk

7.2.9. Implementation and testing document

This document is the third deliverable. The objective of this document is to describe how to test the requirements and design of the project. It also encloses information for the user.

- Introduction
- Testing report
    - Test coverage
    - Test cases
- System delivery
    - Installation manual
    - User manual
- • Final cost estimate

7.2.10. Complete report

This document is the last deliverable. The objective of this document is to present the project from start to finish, including the previous documents.

7.3. Project estimates

This estimate is done in hours of work because that is the resource spent to complete the project and is approximated to the closest hour. Also, the project estimate will be divided following the phases described in the plan to be clear.

Table 9. Activity cost estimation

Phase	Activity Name	Cost	Phase cost
1			4
	e-mail exchange	1	
	Individual work on the systems overview document	2	
	Collate and assemble completed task.	1	
2			6
	Introductions and set up communications	1	
	Set up group framework	1	
	Discussion of project requirements and brainstorming	1	
	Decide on technical resources and generalized scope	3	
3			40
	Finalized design	4	
	Dividing the work for the scope and plan document	1	
	Completed SVN server	3	
	Completed databases	32	
4			70
	Implementing basic application interface	20	
	Implementing classes and functions	50	
5			50
	Completion of detailed design	10	
	Completion of architectural design	10	
	Prototype and risk report and completed report	20	
6			120
	Testing the final application	40	
	Product debugging	40	
	Quality testing	40	
7			5
	Finalize report	5	
		Total Cost:	**295**

7.4. Activities assignment

Activity 1: Everyone
Activity 2: Everyone had one section
Activity 3: Rolf
Activity 4: Denis
Activity 5: Ryan
Activity 6: Everyone
Activity 7: Amine, Jenia, Rolf
Activity 8: Everyone
Activity 9: Astou, Marwa, Jean-Pierre, Ryan
Activity 10: Amine
Activity 11: Jenia, Rolf, Amine, Denis
Activity 12: Jenia, Rolf, Amine, Ryan
Activity 13: Denis, Marwa
Activity 14: Astou, Marwa, Ryan, Denis
Activity 15: Astou, Marwa, Ryan, Denis, Rolf
Activity 16: Everyone
Activity 17: Jenia, Rolf, Amine
Activity 18: Everyone
Activity 19: Everyone
Activity 20: Everyone

7.5. Schedule

Table 10. Schedule of the individual activities

Activity	Duration	W1	W2	W3	W4	W5	W6	W7	W8	W9	W10	W11	W12	W13	W14
A1	1	■													
A2	2	■													
A3	1		■												
A4	1		■												
A5	1		■	■											
A6	1			■											
A7	3			■											
A8	4			■	■										
A9	1			■	■	■									
A10	3		■	■	■	■									
A11	32		■	■	■	■									
A12	20					■	■	■	■	■					
A13	50						■	■	■	■	■	■			
A14	10						■	■	■						
A15	10								■	■	■				
A16	20										■				
A17	40										■	■	■	■	
A18	40											■	■	■	
A19	40											■	■	■	
A20	5													■	■

7.6. Risk

- There is currently no mechanism devised to help users recover their passwords in the case they forget. This means that if they cannot remember their log in name and or password, they cannot access the system.
- Due to time constraints, many optional functions may be scoped out before the project approaches deadline. Though they would not affect the usability of the program, they could reduce the ease of use and simplicity of the program.

- Our current choice of programming language for the application may not prove to be the most efficient or most effective. Given the experience and knowledge of our team, it was decided to follow the lowest common denominator in terms of compatibility with our programmers.
- There is nothing preventing users that are not Concordia students to gain access to the application so long as they know the user name and password of a student.
- A slow processing speed would be another potential risk of the design we chose. This is because the application can generate a large pool of potential schedules, based on the classes available and whether or not the students has entered any time constraints. As well, certain combinations of criteria may cause great strain on the application and slow it down drastically.
- Potential problems can arise when a user suddenly loses access to the internet, and may risk losing any changes that he or she may have made to their schedule prior to losing their connection.

8. Architectural design

The architecture design of the system considers the 4+1 diagrams in section 8.1. This includes the logical view, the development view, the process view and the physical view. As well, to elaborate on the design of the system each individual subsystem is described in section 8.2.

8.1. Architecture diagram

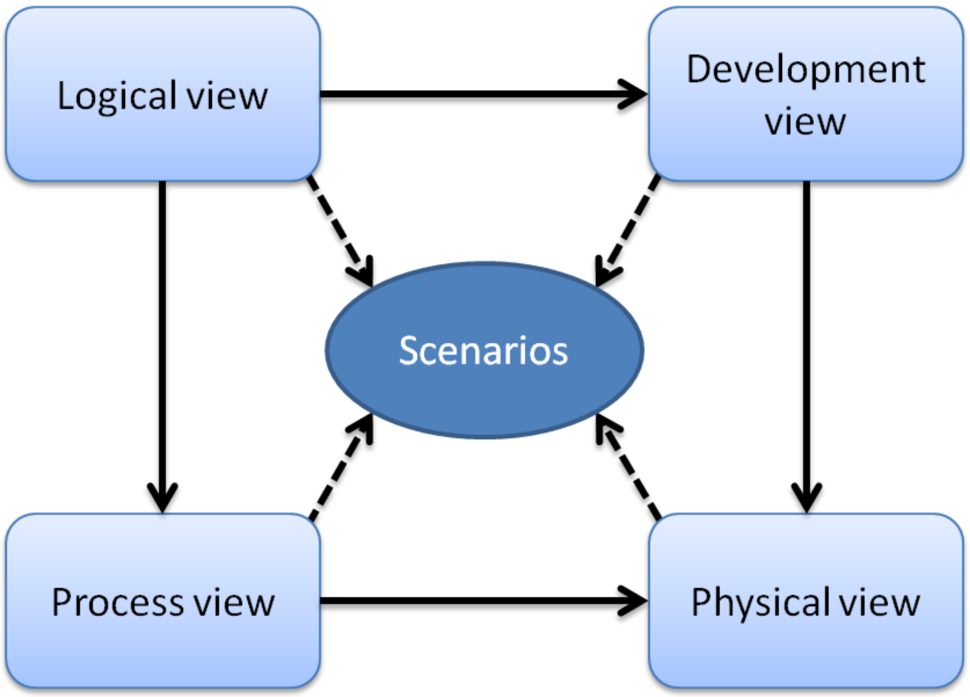


Figure 6. 4+1 architecture view model

The 4+1 architectural view model is composed of the logical view, the development view, the process view and the physical view. This particular view model is used to describe the architecture for software engineering projects from different points of view such as the project manager's, the developers and the end user.

8.1.1. Logical view

In the logical view of the 4+1 model, the functionality that the application provides to the user is displayed. Following the MVC architectural model, the class diagram in Figure 7 details end user actions at the view level, relayed through the controller to retrieve, manipulate or change data in the model. The controller does not change the information itself since the model is the one that performs all changes. After this is done, the model returns the information back to the controller which feeds it back to the view.

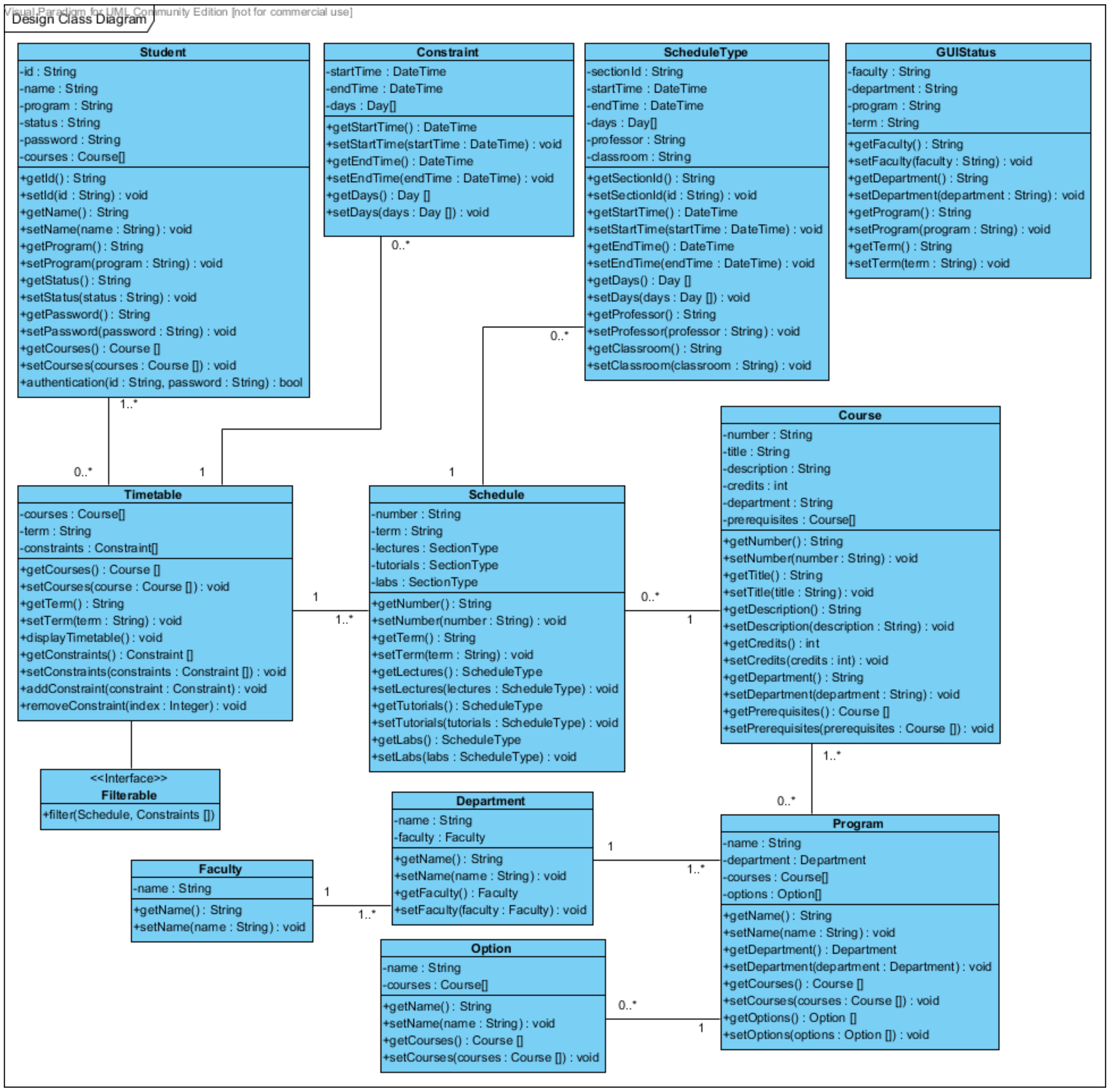

Visual Paradigm for UML Community Edition [not for commercial use]
Design Class Diagram
Student
-id : String
-name : String
-program : String
-status : String
-password : String
-courses : Course[]
+getId() : String
+setId(id : String) : void
+getName() : String
+setName(name : String) : void
+getProgram() : String
+setProgram(program : String) : void
+getStatus() : String
+setStatus(status : String) : void
+getPassword() : String
+setPassword(password : String) : void
+getCourses() : Course []
+setCourses(courses : Course []) : void
+authentication(id : String, password : String) : bool
Constraint
-startTime : DateTime
-endTime : DateTime
-days : Day[]
+getStartTime() : DateTime
+setStartTime(startTime : DateTime) : void
+getEndTime() : DateTime
+setEndTime(endTime : DateTime) : void
+getDays() : Day []
+setDays(days : Day []) : void
ScheduleType
-sectionId : String
-startTime : DateTime
-endTime : DateTime
-days : Day[]
-professor : String
-classroom : String
+getSectionId() : String
+setSectionId(id : String) : void
+getStartTime() : DateTime
+setStartTime(startTime : DateTime) : void
+getEndTime() : DateTime
+setEndTime(endTime : DateTime) : void
+getDays() : Day []
+setDays(days : Day []) : void
+getProfessor() : String
+setProfessor(professor : String) : void
+getClassroom() : String
+setClassroom(classroom : String) : void
GUIStatus
-faculty : String
-department : String
-program : String
-term : String
+getFaculty() : String
+setFaculty(faculty : String) : void
+getDepartment() : String
+setDepartment(department : String) : void
+getProgram() : String
+setProgram(program : String) : void
+getTerm() : String
+setTerm(term : String) : void
0..*
0..*
1..*
0..*
1
1
Timetable
-courses : Course[]
-term : String
-constraints : Constraint[]
+getCourses() : Course []
+setCourses(course : Course []) : void
+getTerm() : String
+setTerm(term : String) : void
+displayTimetable() : void
+getConstraints() : Constraint []
+setConstraints(constraints : Constraint []) : void
+addConstraint(constraint : Constraint) : void
+removeConstraint(index : Integer) : void
Schedule
-number : String
-term : String
-lectures : SectionType
-tutorials : SectionType
-labs : SectionType
+getNumber() : String
+setNumber(number : String) : void
+getTerm() : String
+setTerm(term : String) : void
+getLectures() : ScheduleType
+setLectures(lectures : ScheduleType) : void
+getTutorials() : ScheduleType
+setTutorials(tutorials : ScheduleType) : void
+getLabs() : ScheduleType
+setLabs(labs : ScheduleType) : void
Course
-number : String
-title : String
-description : String
-credits : int
-department : String
-prerequisites : Course[]
+getNumber() : String
+setNumber(number : String) : void
+getTitle() : String
+setTitle(title : String) : void
+getDescription() : String
+setDescription(description : String) : void
+getCredits() : int
+setCredits(credits : int) : void
+getDepartment() : String
+setDepartment(department : String) : void
+getPrerequisites() : Course []
+setPrerequisites(prerequisites : Course []) : void
1
1..*
0..*
1
1..*
0..*
<<Interface>>
Filterable
+filter(Schedule, Constraints [])
Department
-name : String
-faculty : Faculty
+getName() : String
+setName(name : String) : void
+getFaculty() : Faculty
+setFaculty(faculty : Faculty) : void
1
1..*
Program
-name : String
-department : Department
-courses : Course[]
-options : Option[]
+getName() : String
+setName(name : String) : void
+getDepartment() : Department
+setDepartment(department : Department) : void
+getCourses() : Course []
+setCourses(courses : Course []) : void
+getOptions() : Option []
+setOptions(options : Option []) : void
Faculty
-name : String
+getName() : String
+setName(name : String) : void
1
1..*
Option
-name : String
-courses : Course[]
+getName() : String
+setName(name : String) : void
+getCourses() : Course []
+setCourses(courses : Course []) : void
0..*
1

Figure 7. Design class diagram

8.1.2. Development view

The whole application has been divided into three packages. The system can best be represented by the client side package, corresponding to the user interactions, the back-end side, called the service side package which is the background code servicing the application requests, and the resource package, that holds the CSS codes and image repository for the site. This can also be seen as an MVC modeled package diagram as shown in Figure 8.

The client side package contains the HTML view, the JavaScript controller and model that handles scripted events on the application and relies on the jQuery JavaScript library. The service side package contains all the PHP code that runs the application and depends on the data from the resource package, like any controller would depend on the model in the MVC architecture.

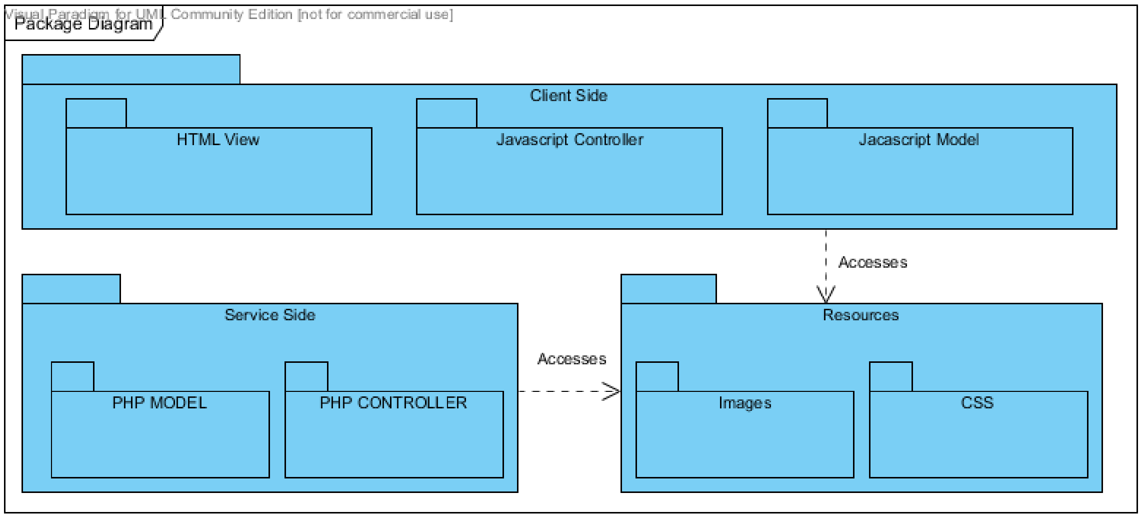


Figure 8. Package diagram

The following component diagram shows how the components are connected to each other. It illustrates the service providers and the service consumers. As well, it shows how these components are assembled together to form larger components. Finally, it shows the levels of relation and dependency of the components relative to the MVC architecture, and how each component fits into the system such that they all interface through a port or a connector. In Figure 9, the controller component is shown along with the components it encompasses. The components of the view serve as connections between the components of the model and the components of the view. They act as a sort of operation management by providing services to the view, and consuming the services provided by the model.

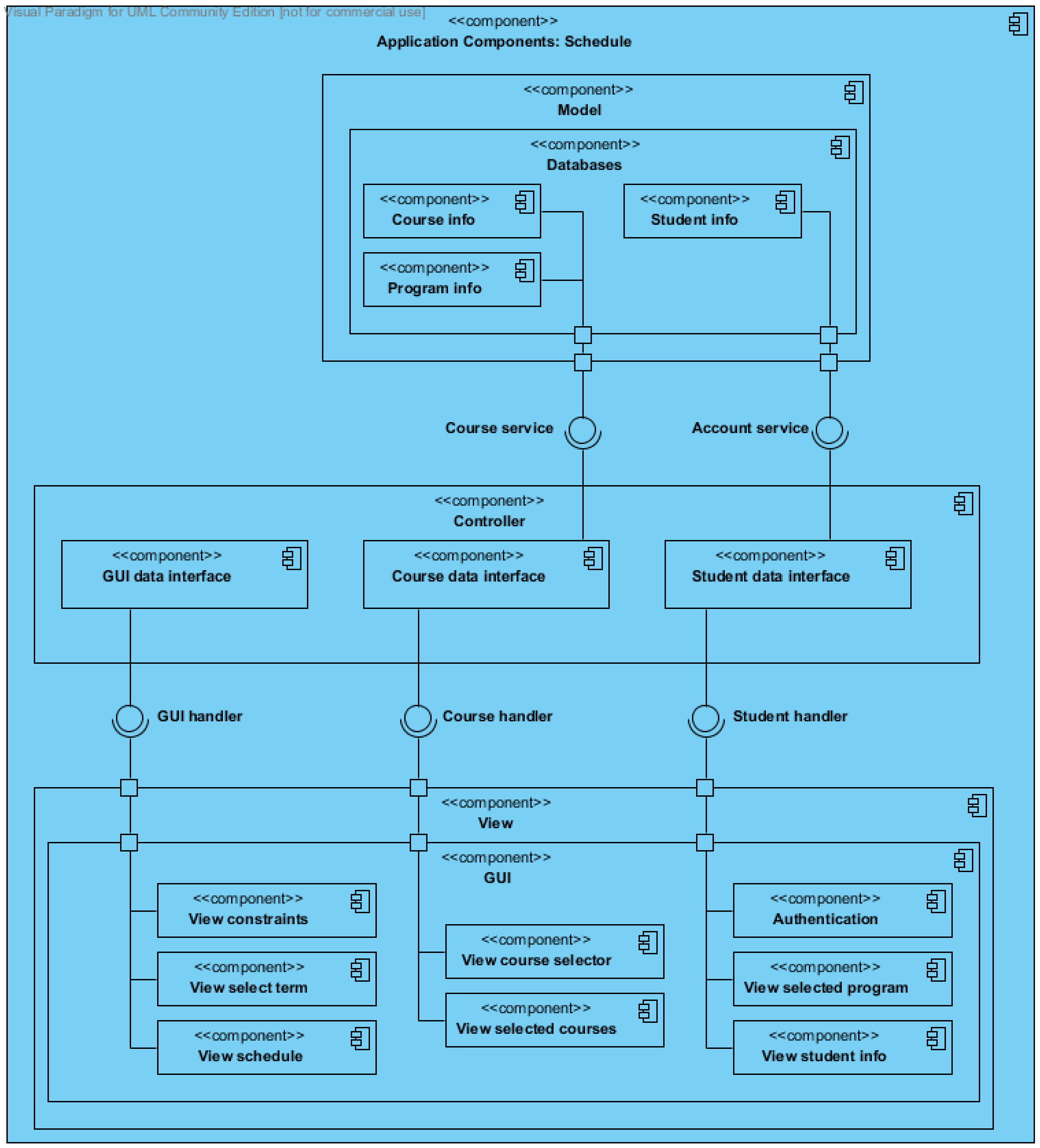


Figure 9. Component diagram

8.1.3. Physical view

The application was conceived using the MVC architecture, which allows the system to be built modularly based on three different components. These are the view portion of the MVC architecture which represents the GUI of the application and is constructed using HTML and JavaScript. The controller for our system will be coded in PHP, the model in MySQL and PHP. The controller and the model will communicate with each other using Ajax.

The deployment diagram (Figure 10) displays the deployment layers of the system. At its base, the system runs on any modern operating system, including Windows. The web server runs on Apache (within the XAMPP application environment) which was chosen because of its reliability. The Apache server also contains execution environments for PHP and the JavaScript modules.

User requests and interaction with the application that needs data from the database are handled also in AJAX. MySQL is used as the database for this project.

In Figure 10, the view of the user interaction with the primary system is shown. The system allows the user to use any kind of computer, on any kind of operating system, whether it is Windows XP/Vista/7, Linux or Mac OS. The user may access the application with any web browser, and though primarily tested for optimal use in Google Chrome, the application may also be accessed with Internet Explorer, Firefox, Opera or Safari.

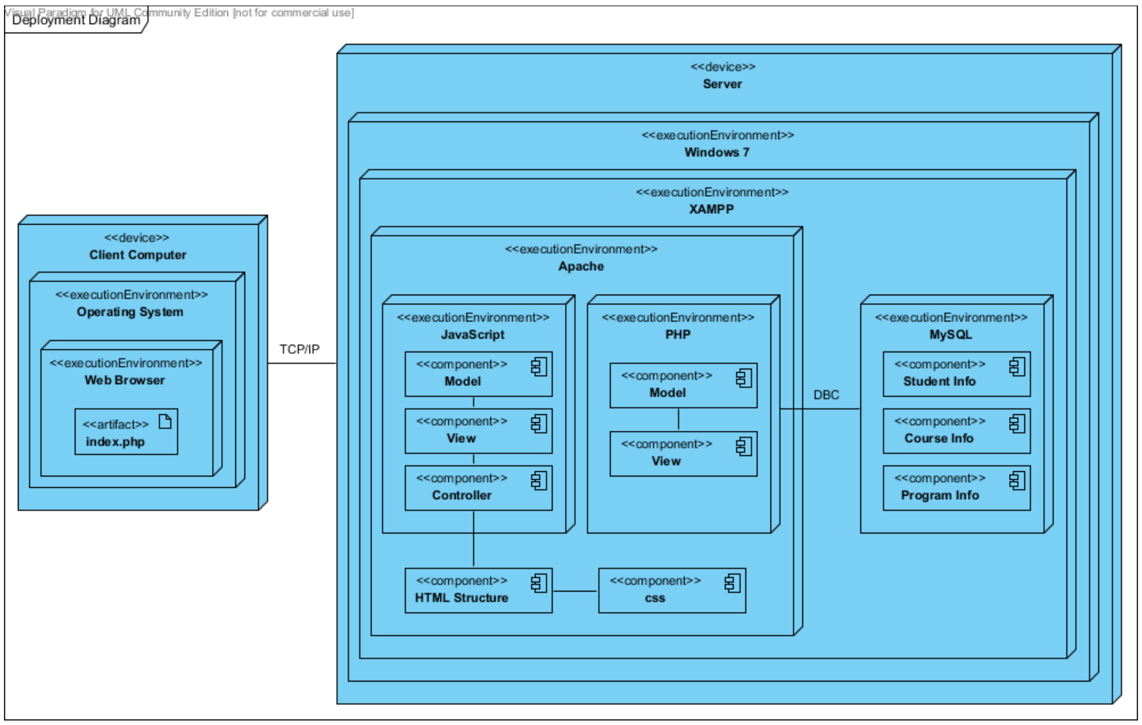

Visual Paradigm for UML Community Edition [not for commercial use]
Deployment Diagram
<<device>> Server
<<executionEnvironment>> Windows 7
<<executionEnvironment>> XAMPP
<<executionEnvironment>> Apache
<<executionEnvironment>> JavaScript
<<component>> Model
<<component>> View
<<component>> Controller
<<executionEnvironment>> PHP
<<component>> Model
<<component>> View
<<component>> HTML Structure
<<component>> css
DBC
<<executionEnvironment>> MySQL
<<component>> Student Info
<<component>> Course Info
<<component>> Program Info
<<device>> Client Computer
<<executionEnvironment>> Operating System
<<executionEnvironment>> Web Browser
<<artifact>> index.php
TCP/IP

Figure 10. Deployment diagram

8.1.4. Process view

In order to illustrate the dynamic aspects of our system, an activity diagram is used to explain the steps required to generate a set of schedules (Figure 11). In the following, the general flow of the system is explained.

The application is accessed with a web browser and the user has the option to log on with his credentials. After the user logs on, the system displays user information such as user name, program, course load, status (full/part-time) and program option. If the user logs on, the display will reflect which courses have been passed, which can be taken this term and which courses are not available due to missing prerequisites. The application will work even if the user does not log on. In that case, all courses of a selected program will be available for selection, no prerequisite checks will be performed and no course load information is displayed.

After the user selects a program, the application updates the course display to show the courses in the selected program. Furthermore, courses that are not offered in the selected term are displayed but not enabled. By default, the courses for the fall term are shown, but the user can select the winter term as well.

The user then selects at least one course to take in the selected term. The system will add the course(s) to the 'selected courses list' and will display course load information based on the status of the user (full/part-time) if the user logged on. The user can remove courses from the 'selected courses list' and the system updates the display accordingly. Moreover, the user can add time constraints (*i.e.* time slots that are not available to take courses). In response, the system adds the constraints to the schedule display. Obviously, the user can also delete a time constraint. After the user removes a constraint from the schedule, the system updates the schedule display.

Finally, the user can generate the schedule. The application will create a set of possible schedules, display them and indicate the total number of schedules generated. The schedules will include the time constraints (if any) and the courses the user selected to take. The user can then browse through all available schedules, select the ones that are convenient and save them.

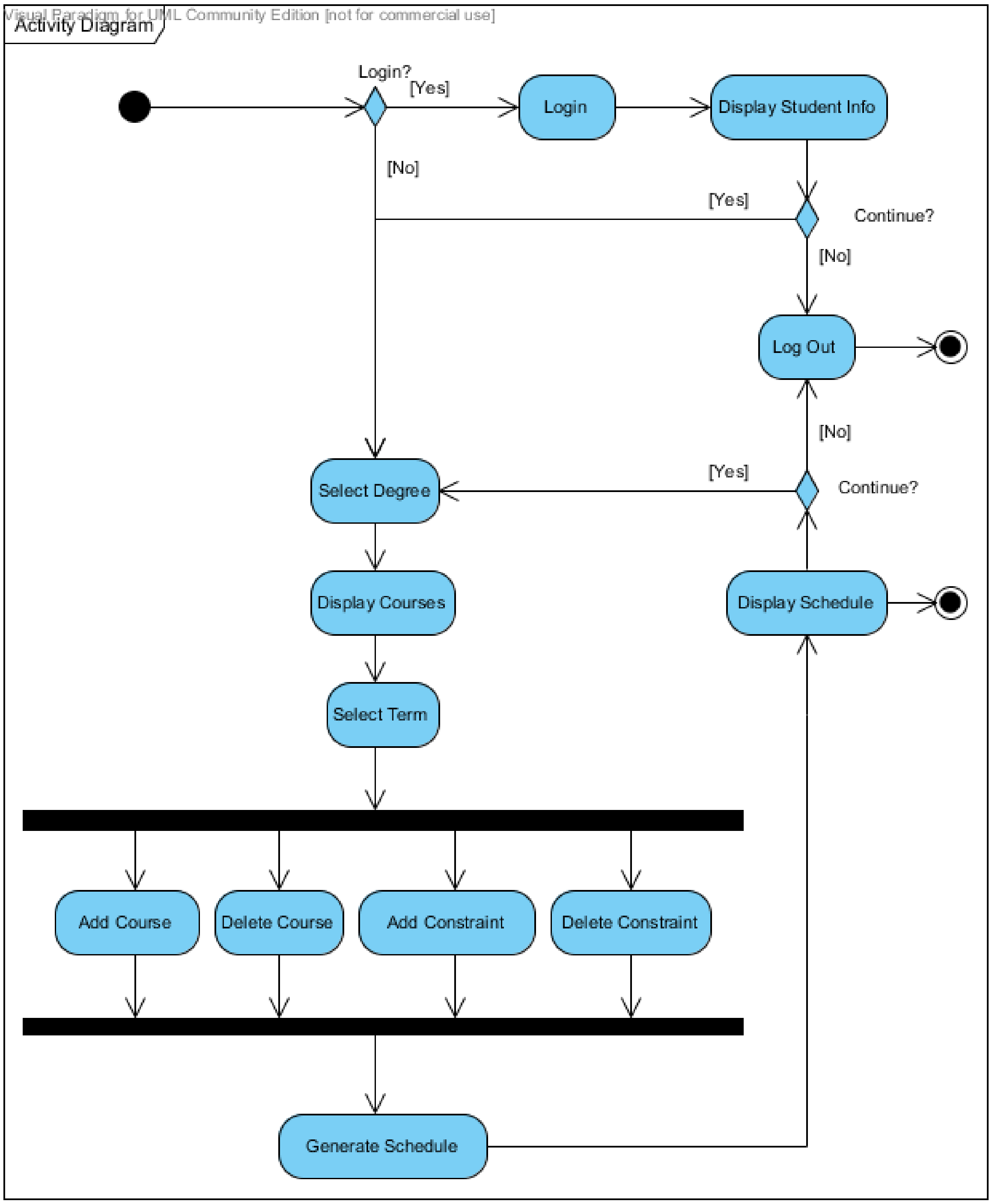


Figure 11. Activity diagram

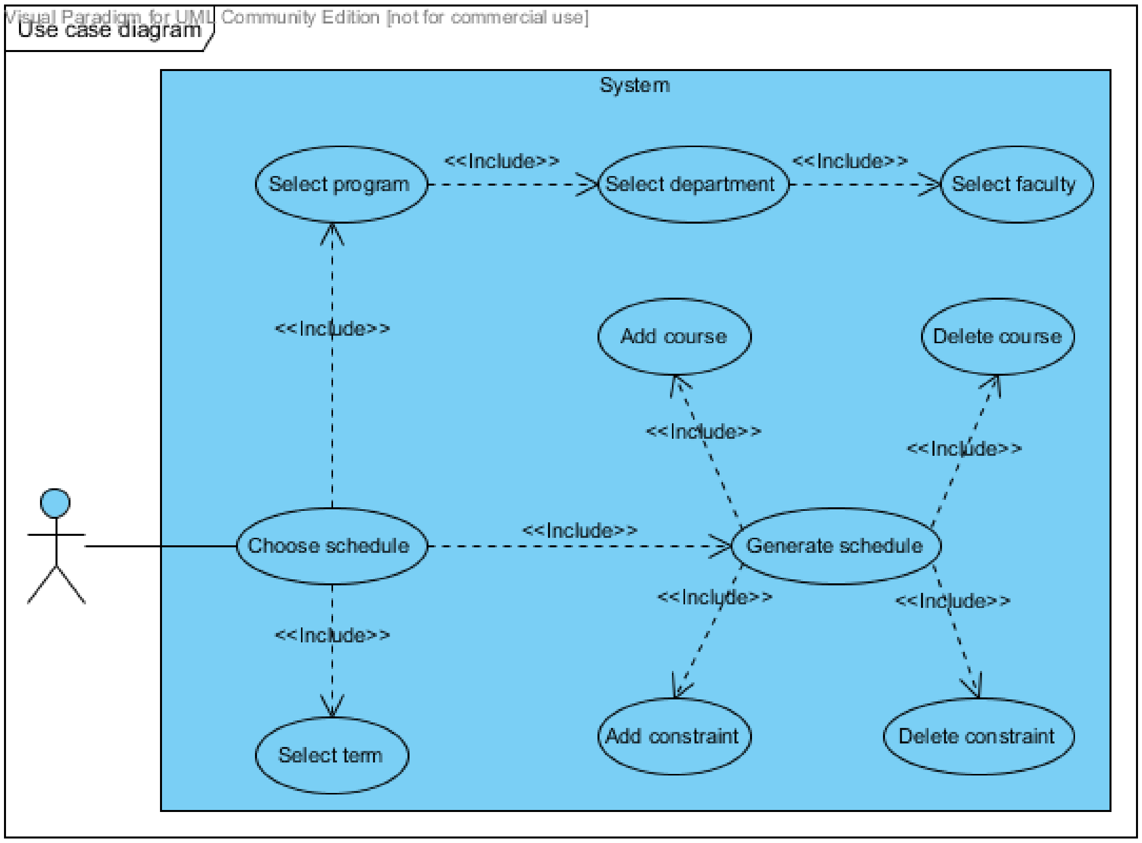


Figure 12. Scenarios

8.2. Subsystem interface specifications

Each module is part of both the server side and the client side, meaning it is implemented in both the server and client side – for different purposes. The module at the client side is responsible to get the object information from its counterpart at the server side. The server side is responsible to retrieve the information from the database and send it to the client side script. And finally, the client side code must unmarshal the information and display it to the user.

As the program runs on a system that is not completely object oriented, some of the components represented in the component diagram are not actual classes, though they do function with that purpose in mind.

Essentially, there are two major internal interaction components: the controller components and the model components. The controller components interface with the view by handling the events on the HTML page triggered by the user. The model components provide the services related to the user account, courses and program information.

8.2.1. Faculty

- void Faculty.setName(name: String)
- String Faculty.getName()

8.2.2. Department

- void Department.setName(name: String)
- String Department.getName()

8.2.3. Program

- void Program.setName(name: String)
- String Program.getName()

8.2.4. Course service

- GUIStatus.getFaculty(): String
- Course.getPrerequisites(): Course[]
- Course.getCredits(): int
- Schedule.getTerm(): String
- String ScheduleType.getStartTime()
- String ScheduleType.getEndTime()
- String ScheduleType.getDay()
- String ScheduleType.getProfessor()
- String ScheduleType.getClassroom()

  The above ScheduleType methods are time period specific methods, called and enacted upon individual lecture, tutorial or laboratory schedules.

- ScheduleType[] Schedule.getLectures()
- ScheduleType [] Schedule.getTutorials()
- ScheduleType [] Schedule.getLabs()
- String ScheduleType.getSectionId()

  The getLectures, getTutorials and getLabs methods return arrays of ScheduleType representing lectures, tutorials and laboratories.

8.2.5. Account service

- string Student.getName()
- string Student.getProgram()
- string Student.getPassword()
- Course[] Student.getCourses()
- string Student.getStatus()

  The above methods will return the corresponding information. In particular, getCourses returns an array of Course objects that represent the courses a student has successfully completed.

8.2.6. Course handler

- bool Timetable.addCourse(course: Course)
- bool Timetable.removeCourse(id: int)
- void Course.displayInfo()

  addCourse requires a Course object and returns a Boolean value to indicate if the course was added. deleteCourse takes an ID for the course to be removed and returns a Boolean value to indicate if the course was removed. The displayInfo function displays the course information directly to the view.

8.2.7. Student handler

- bool Student.authentication(id: String, password: String)

  The above method requires two string values as parameters: the ID and the password of the user. It returns a Boolean value to indicate if the ID and password are correct.

- String[] displayStudentInfo(student: Student)

  displayStudentInfo takes a Student object and directly displays the student information in the view.

8.2.8. GUI handler

- bool Timetable.addConstraint(constraint: Constraint)
- bool Timetable.deleteConstraint(id: int)
- void Timetable.displaySchedule()
- String GUIStatus.getTerm()

  addConstraint requires a Constraint object and returns a Boolean value to indicate if the constraint was added. deleteConstraint takes an ID for the constraint to be removed and returns a Boolean value to indicate if the constraint was removed. The getTerm function returns a string that will be used to update the course display. displaySchedule updates the schedule shown on the HTML page.

9. Detailed design

9.1. Subsystems

9.1.1. Detailed design diagram

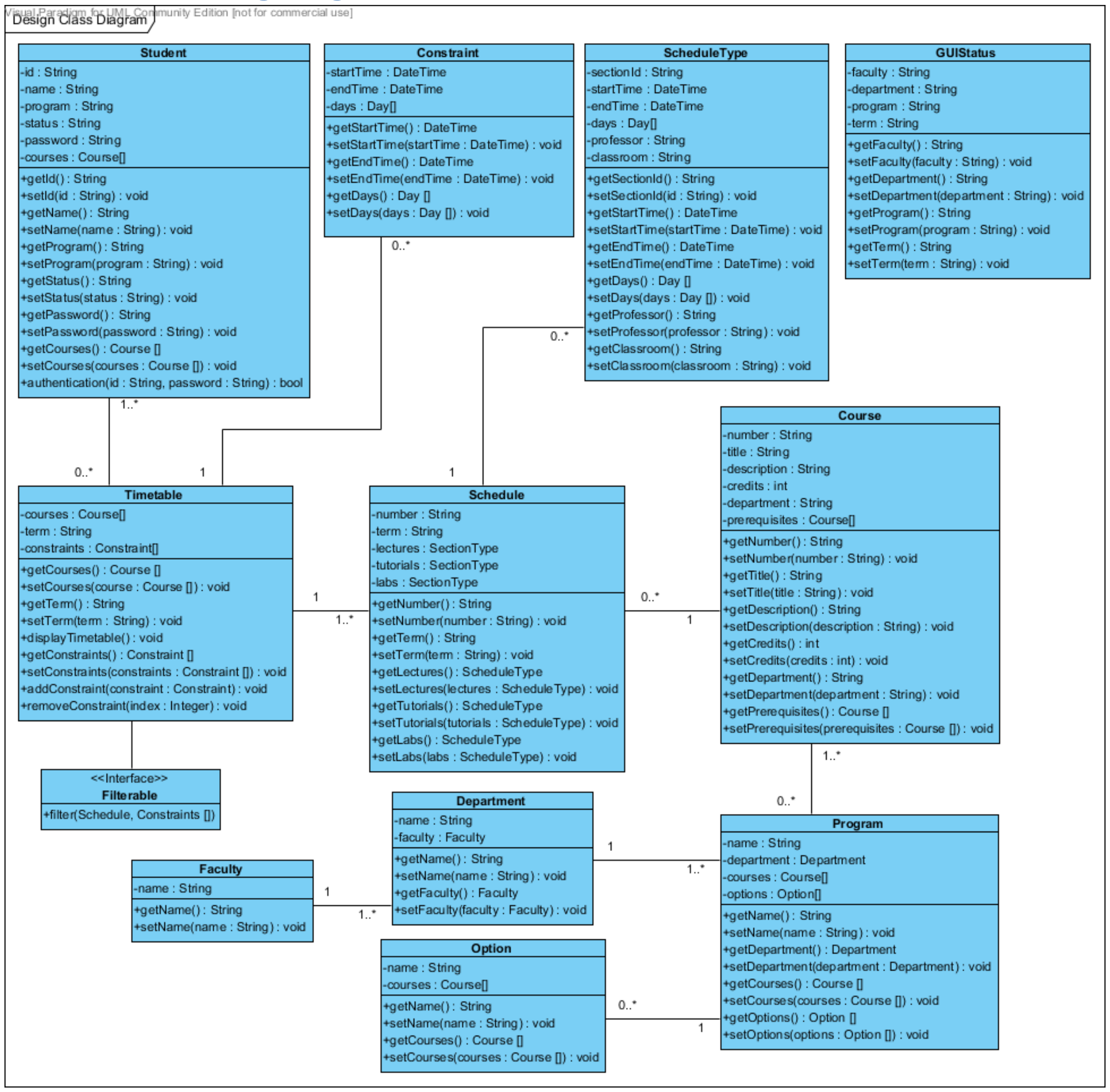

Visual Paradigm for UML Community Edition [not for commercial use]
Design Class Diagram
Student
-id : String
-name : String
-program : String
-status : String
-password : String
-courses : Course[]
+getId() : String
+setId(id : String) : void
+getName() : String
+setName(name : String) : void
+getProgram() : String
+setProgram(program : String) : void
+getStatus() : String
+setStatus(status : String) : void
+getPassword() : String
+setPassword(password : String) : void
+getCourses() : Course []
+setCourses(courses : Course []) : void
+authentication(id : String, password : String) : bool
Constraint
-startTime : DateTime
-endTime : DateTime
-days : Day[]
+getStartTime() : DateTime
+setStartTime(startTime : DateTime) : void
+getEndTime() : DateTime
+setEndTime(endTime : DateTime) : void
+getDays() : Day []
+setDays(days : Day []) : void
ScheduleType
-sectionId : String
-startTime : DateTime
-endTime : DateTime
-days : Day[]
-professor : String
-classroom : String
+getSectionId() : String
+setSectionId(id : String) : void
+getStartTime() : DateTime
+setStartTime(startTime : DateTime) : void
+getEndTime() : DateTime
+setEndTime(endTime : DateTime) : void
+getDays() : Day []
+setDays(days : Day []) : void
+getProfessor() : String
+setProfessor(professor : String) : void
+getClassroom() : String
+setClassroom(classroom : String) : void
GUIStatus
-faculty : String
-department : String
-program : String
-term : String
+getFaculty() : String
+setFaculty(faculty : String) : void
+getDepartment() : String
+setDepartment(department : String) : void
+getProgram() : String
+setProgram(program : String) : void
+getTerm() : String
+setTerm(term : String) : void
Course
-number : String
-title : String
-description : String
-credits : int
-department : String
-prerequisites : Course[]
+getNumber() : String
+setNumber(number : String) : void
+getTitle() : String
+setTitle(title : String) : void
+getDescription() : String
+setDescription(description : String) : void
+getCredits() : int
+setCredits(credits : int) : void
+getDepartment() : String
+setDepartment(department : String) : void
+getPrerequisites() : Course []
+setPrerequisites(prerequisites : Course []) : void
Timetable
-courses : Course[]
-term : String
-constraints : Constraint[]
+getCourses() : Course []
+setCourses(course : Course []) : void
+getTerm() : String
+setTerm(term : String) : void
+displayTimetable() : void
+getConstraints() : Constraint []
+setConstraints(constraints : Constraint []) : void
+addConstraint(constraint : Constraint) : void
+removeConstraint(index : Integer) : void
Schedule
-number : String
-term : String
-lectures : SectionType
-tutorials : SectionType
-labs : SectionType
+getNumber() : String
+setNumber(number : String) : void
+getTerm() : String
+setTerm(term : String) : void
+getLectures() : ScheduleType
+setLectures(lectures : ScheduleType) : void
+getTutorials() : ScheduleType
+setTutorials(tutorials : ScheduleType) : void
+getLabs() : ScheduleType
+setLabs(labs : ScheduleType) : void
<<Interface>>
Filterable
+filter(Schedule, Constraints [])
Department
-name : String
-faculty : Faculty
+getName() : String
+setName(name : String) : void
+getFaculty() : Faculty
+setFaculty(faculty : Faculty) : void
Faculty
-name : String
+getName() : String
+setName(name : String) : void
Program
-name : String
-department : Department
-courses : Course[]
-options : Option[]
+getName() : String
+setName(name : String) : void
+getDepartment() : Department
+setDepartment(department : Department) : void
+getCourses() : Course []
+setCourses(courses : Course []) : void
+getOptions() : Option []
+setOptions(options : Option []) : void
Option
-name : String
-courses : Course[]
+getName() : String
+setName(name : String) : void
+getCourses() : Course []
+setCourses(courses : Course []) : void
1..*
0..*
1
0..*
0..*
1
1
1..*
0..*
1
1..*
0..*
1
1..*
1
1..*
0..*
1

Figure 13. Design class diagram

9.1.2. Unit descriptions

9.1.2.1. Course

List of courses that a student can add to their schedule.

Attributes: name, time, description, department

Function: N/A

9.1.2.2. Course group

The group that a specific course belongs to. This is used based on the guidelines of the academic calendar, which describes certain course groupings that students must select courses from.

Attributes: name, program, ID

Function: N/A

9.1.2.3. Department

List of departments at the university

Attributes: name, ID

Function: N/A

9.1.2.4. Faculty

List of faculties at the university

Attributes: faculty name, ID

Function: N/A

9.1.2.5. Generated Schedule

The class that generates a schedule into the graphical user interface (GUI) based on the selections input by the user. This needs to be built in conjunction with the GUI developer to ensure that it is able to send the right variables to produce a proper schedule.

Attributes: courses, times

Function: generateSchedule()

9.1.2.6. Prerequisite group

This group contains a group of courses that must be taken prior to the student being allowed to take another course. When a student adds a course, the scheduler would check for a prerequisite group required on that course. If the student has not taken any of the prerequisites, then the scheduler would not allow the student to add that course to their schedule.

Attributes: ID, name, ID, concurrent

The concurrent flag checks if a certain course is allowed to be taken concurrently with the course the student wants to add. If so, it alerts the student.

Function: N/A

9.1.2.7. Program

This is used to identify the program that the student is in. It will determine whether they are allowed to take certain courses that are reserved for students that are in-program.

Attributes: name, ID

Function: N/A

9.1.2.8. Schedule Generator

This is a key class in the scheduler software. It is what implements the scheduler algorithm to determine, based on all the adjacencies, prerequisites, and any other data the system provides, what alternatives the student has when building a schedule. It will search through all algorithms, and output to the student, using the generated schedule class, various sample schedules that the student can consider. This is done by running the algorithm and searching through all the data to find results that do not produce errors, based on the given input.
Attributes: adjacencies, program, faculty, courses, time constraints, department
Function: generateSchedule()

9.1.2.9. Student

This contains the information that allows the student to properly access and use the system.
Attributes: firstName, lastName, ID, status (full time/part time), password and option.
Function: N/A

9.1.2.10. Time Constraint

This tells the scheduler not to generate schedules that fall during the user-defined time constraint blocks.
Requires: constraints
Functions: addTimeConstraint(), showTimeConstraint(), removeTimeConstraint()

10. Dynamic design scenarios

Figure 14 shows the public interface of our system, listing all system operations. Some of these system operations are used in the following to give more detail about two substantial use cases, namely 'Choose schedule' and 'Select program'.

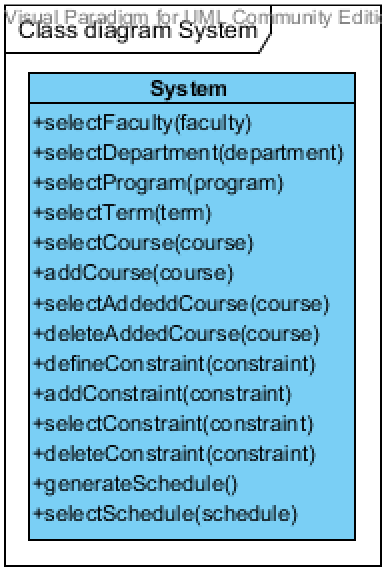

Visual Paradigm for UML Community Editi
Class diagram System
System
+selectFaculty(faculty)
+selectDepartment(department)
+selectProgram(program)
+selectTerm(term)
+selectCourse(course)
+addCourse(course)
+selectAddeddCourse(course)
+deleteAddedCourse(course)
+defineConstraint(constraint)
+addConstraint(constraint)
+selectConstraint(constraint)
+deleteConstraint(constraint)
+generateSchedule()
+selectSchedule(schedule)

Figure 14. Public system interface

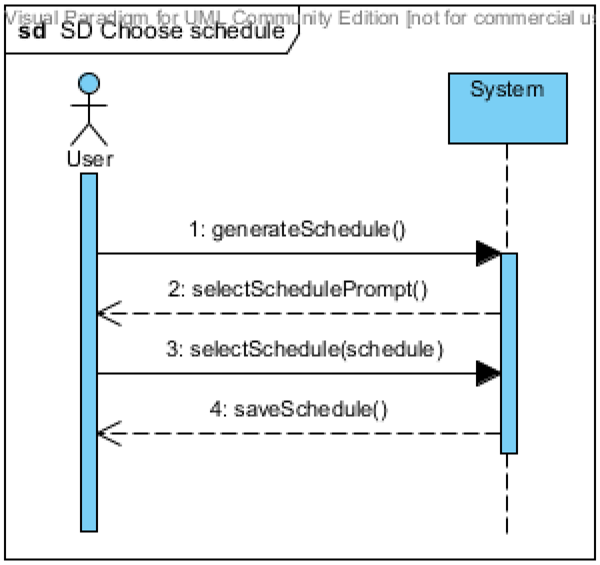

Visual Paradigm for UML Community Edition [not for commercial u
sd SD Choose schedule
User
System
1: generateSchedule()
2: selectSchedulePrompt()
3: selectSchedule(schedule)
4: saveSchedule()

Figure 15. System sequence diagram for use case 'Choose schedule'

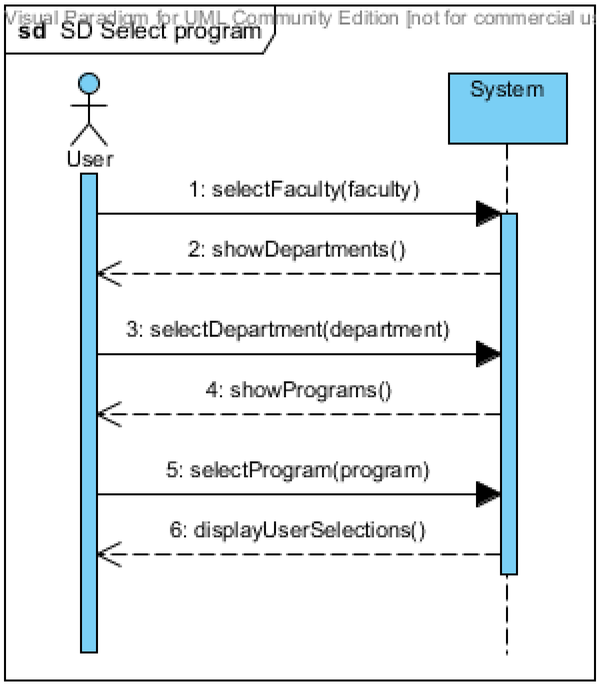

Visual Paradigm for UML Community Edition [not for commercial u
sd SD Select program
User
System
1: selectFaculty(faculty)
2: showDepartments()
3: selectDepartment(department)
4: showPrograms()
5: selectProgram(program)
6: displayUserSelections()

Figure 16. System sequence diagram for use case 'Select program'

10.1. Operational contracts

10.1.1. Use case "Choose schedule"

Operation Name	generateSchedule()
Cross-References	♦ Choose schedule ♦ Add course ♦ Delete course ♦ Add constraint ♦ Delete constraint
Preconditions	♦ User is logged in ♦ User has selected a term ♦ User has added course(s)
Postconditions	♦ A set of Schedule objects was created

Operation Name	selectSchedulePrompt()
Cross-References	♦ Choose schedule
Preconditions	♦ A set of all possible schedules has been created
Postconditions	♦ A GUI prompt was created

Operation Name	selectSchedule(schedule)
Cross-References	♦ Choose schedule
Preconditions	♦ Student was asked to choose a schedule
Postconditions	♦ The Schedule object was set to Selected

Operation Name	saveSchedule()
Cross-References	♦ Choose schedule
Preconditions	♦ Student has selected the desired schedule
Postconditions	♦ The selected Schedule object was associated with the Student object ♦ The value Selected was removed

10.1.1. Use case "Select Program"

Operation Name	selectFaculty(faculty)
Cross-References	♦ Choose Schedule
Preconditions	♦ GUI displayed existing faculties to the student
Postconditions	♦ The Faculty object was set to Selected

Operation Name	displayDepartmentsOfFaculty()
Cross-References	♦ Choose Schedule
Preconditions	♦ Student has selected a faculty displayed in the GUI
Postconditions	♦ The list of Department objects was created

Operation Name	selectDepartment(department)
Cross-References	♦ Choose schedule
Preconditions	♦ Student has access to a list of departments displayed in the GUI
Postconditions	♦ The Department object was set to Selected

Operation Name	displayProgramsOfferedByDepartment()
Cross-References	♦ Choose schedule
Preconditions	♦ Student selected a department from the choices displayed in the GUI
Postconditions	♦ The list of Programs objects was created

Operation Name	selectProgram(program)
Cross-References	♦ Choose schedule
Preconditions	♦ All programs in department were displayed in the GUI to the student
Postconditions	♦ The Program object was set to Selected

Operation Name	displayUserSelections()
Cross-References	♦ Choose Schedule
Preconditions	♦ Student has selected a program from the department
Postconditions	♦ The Program object was associated with the Student object ♦ The value Selected was removed from the selected Faculty object, Department object and Program object.

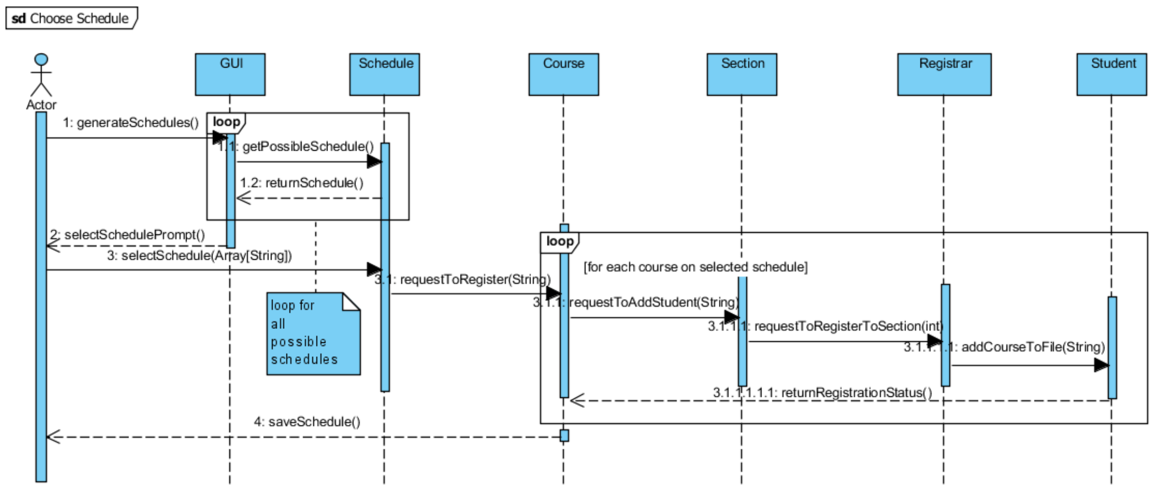

sd Choose Schedule
Actor
GUI
Schedule
Course
Section
Registrar
Student
1: generateSchedules()
loop
1.1: getPossibleSchedule()
1.2: returnSchedule()
2: selectSchedulePrompt()
3: selectSchedule(Array[String])
loop for all possible schedules
loop
[for each course on selected schedule]
3.1: requestToRegister(String)
3.1.1: requestToAddStudent(String)
3.1.1.1: requestToRegisterToSection(int)
3.1.1.1.1: addCourseToFile(String)
3.1.1.1.1.1: returnRegistrationStatus()
4: saveSchedule()

Figure 17. Sequence diagram for use case 'Choose schedule'

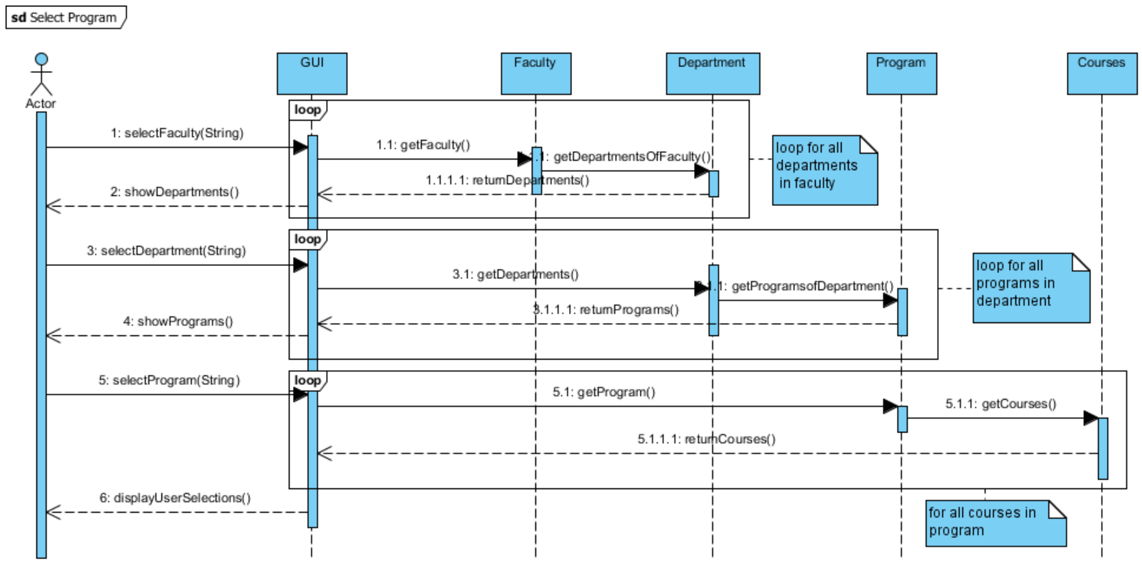

sd Select Program
Actor
GUI
Faculty
Department
Program
Courses
loop
1: selectFaculty(String)
1.1: getFaculty()
1.1.1: getDepartmentsOfFaculty()
1.1.1.1: returnDepartments()
2: showDepartments()
loop for all departments in faculty
loop
3: selectDepartment(String)
3.1: getDepartments()
3.1.1: getProgramsofDepartment()
3.1.1.1: returnPrograms()
4: showPrograms()
loop for all programs in department
loop
5: selectProgram(String)
5.1: getProgram()
5.1.1: getCourses()
5.1.1.1: returnCourses()
6: displayUserSelections()
for all courses in program

Figure 18. Sequence diagram for use case 'Select program'

11. Rapid prototyping and risk

A rapid prototyping report is useful to see if the desired implemented system has any evident flaws. It can also be used to show to the client and get feedback to know if anything needs modification such as requirements or functional behavior.

Table 11 is a commented list of modules, drivers and stubs of the system being implemented.

Table 11 . List of implemented modules, drivers and stubs

Module	Drivers	Stubs	Comments
Login		Login functions	The user interface is shown after the login
Select program	Student information check	Access user database Access course database	Available courses are displayed in the course display area
Select course category	Sequence check	Access course database	Courses corresponding to the selected category are displayed
Select term	Sequence check	Access course database	Course display area is updated
Add course	Sequence check	Access course database Verify sequence	Schedule display area is updated
Input constraint	Add/Clear all check	Access course database	Schedule display area is updated
Logout			The user is returned to the login interface

Rapid prototyping can affect the design decisions by modifying the scope of the project because it is concluded that the needs of the client are not met. Rapid prototyping can be beneficial in the early stages of development because it results in cost savings in the long run. Design decisions on the visual aspects of the system can also be affected by rapid prototyping. One example includes the projected user interface by modifying it to allow the addition or removal of elements depending on the need for them.

The initial risks identified can be asserted or discarded due to rapid prototyping. In this case, the implementation of all the elements found in the requirements and scope cannot be met when doing rapid prototyping due to lack of time and resources. In other words, having a large scope means that only the most important elements will be included in the prototypes. Rapid prototyping can also affect the cost estimates. For example, if any changes have to be done to the established design, an increase of the estimated time will follow. Rapid prototyping can affect the scope as well, causing the rescoping of the project. This implies that it has the potential to affect the design decisions, risks, estimates and scope. All these effects of rapid prototyping might seem negative. However, the earlier it is done, the better the result in terms of cost savings. Table 12 shows the modifications to the estimates established in the first deliverable for the parts still not completed of the project.

Table 12. Modified cost estimates

Phase	Activity name	Old Cost		New Cost	
5			30		60
	Completion of detailed design	6		10	
	Completion of architectural design	4		10	
	Prototype and risk report and completed report	20		40	
6			56		120
	Testing the final application	20		40	
	Product debugging	16		40	
	Quality testing	20		40	
7			4		5
	Finalize report	4		5	

The modifications are a result of rapid prototyping and observing that changes had to be done to the estimated cost in the first deliverable. Phase 5 cost doubled mostly because of rapid prototyping. Phase 6 also doubled but more because the estimate done in the first deliverable was not very realistic for this phase. The risks identified in the first deliverable remain the same and are not affected by rapid prototyping. Scope changes have not occurred either by doing rapid prototyping.

12. GUI design and implementation

The initial GUI design of the scheduling application is shown in Figure 19 and described in detail in the following. Most of the interface elements are part of the jQuery UI framework and all interface elements (even those that are not part of jQuery UI) can be 'themed' by using the jQuery UI ThemeRoller. The ThemeRoller can also be integrated in the webpage so that the user can choose a different theme if desired.

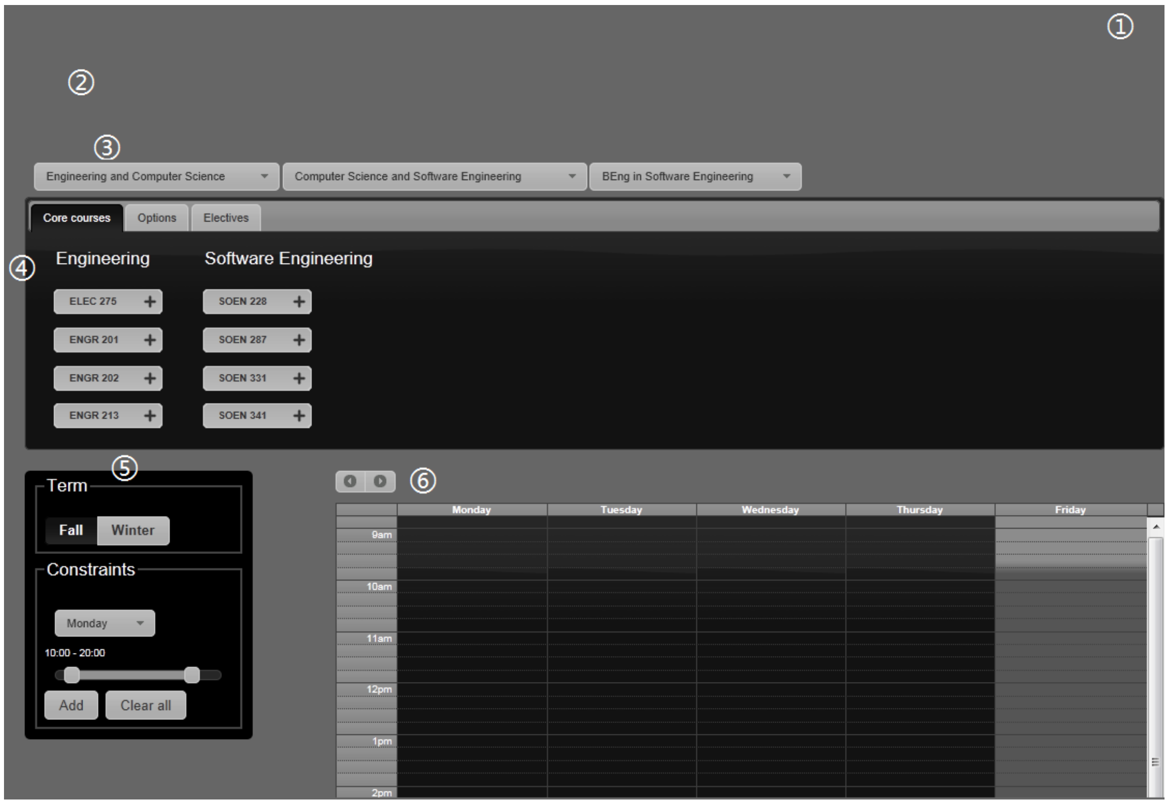


Figure 19. GUI design

① Login: the login area is still not designed in the diagram, but it will be located at that point. The login area is where the student will input a username and password to access the schedule manager.

② Student information: the student information area is still not designed in the diagram, but it will be located at that point. This area will display the student information such as status (full/part-time), program and option.

③ Department/program selection: this area is used to select the faculty, department and program the user wishes to display courses from.

④ Course selection: in this area, the user can choose different tabs (core, options, electives) to display the available courses and then add a course.

⑤ Term and constraints: this area is used to select the term for which classes are displayed. Time constraints can be defined and added to the generated schedule (see below) in this area as well. In addition, there will be a 'selected course list' right above this area which shows the courses the user selected to be added to the schedule.

⑥ Generated schedule: this area is where the generated schedules are displayed. Browsing through all available schedules can be accomplished with the scroll buttons. Furthermore, the total number of generated schedules will be shown here along with the number of the current schedule.

13. Testing report

This section presents all the testing activities undertaken on the final product, as well as all the individual test cases used.

13.1. Test coverage

13.1.1. Tested items

13.1.1.1. Log in – Importance 5/5

The *Log in* test consists of checking if the system allows access when the user enters a valid username and password. Invalid credentials should not grant access to the system.

13.1.1.2. Log out – Importance 2/5

The *Log out* test consists of checking if the system successfully closes the current user session. The user should also be automatically logged out after 10 minutes of inactivity.

13.1.1.3. Course display – Importance 5/5

The *Course display* test consists of checking if the system accurately displays the courses a user can take/select. If the user is logged on, the display will use a color-coded scheme and enable only the courses that the user can take in the selected term. If the user is not logged on, all courses that are offered in the selected term will be available for selection.

13.1.1.4. Add course – Importance 5/5

The *Add course* test consists of checking if a selected course can be added to the schedule. The method returns an error when the course selected conflicts with another course or with a constraint.

13.1.1.5. Remove course – Importance 4/5

The *Remove course* test consists of checking that the course is removed from the schedule.

13.1.1.6. Add constraint – Importance 4/5

The *Add constraint* test consists of checking that the system accepts constraints added by the user. The schedule generated in the timetable displays all constraints imposed by the student.

13.1.1.7. Display schedule – Importance 5/5

The *Display schedule* test consists of checking if a schedule is generated properly where all the courses and added constraints have been taken into account.

13.1.1.8. Browser compatibility

The functionality of the interface on different browsers is important so that the user does not encounter any difficulty while using the system. Browser compatibility was not tested.

13.1.1.9. Data modification

The data used by the system is from the university's database. Some errors could occur in case this database changes the implementation of its data. This is not tested because weekly data exports from the University database are not performed.

13.1.1.10. Operating system compatibility

Operating system compatibility is meant to measure the compatibility of our system with the most recent versions of Mac OS, older versions of Windows (including Windows Vista and Windows XP) as well as Linux based operating systems such as UNIX and Ubuntu. Operating system compatibility was not tested.

13.1.2. Untested items of interest

13.1.2.1. System security

Some network connection features such as the condition that a user is logged in only on one computer at a time are important to make sure that nobody can modify the settings of another user. However, these features are not tested because such restrictions are not implemented.

13.1.2.2. System response

The system should have a reasonably fast response time regardless of the number of users, since its task is to generate a relatively great number of schedules. This is not tested due to the fact that the application is not hosted on a regular web server.

13.1.2.3. Data modification

The data used by the system is from the university's database. Some errors could occur in case this database changes the implementation of its data. This is not tested because weekly data exports from the university database are not performed.

13.2. Test cases

This section gives a descriptive analysis of the test cases used on the items that were tested with an array of techniques and strategies. Listed below is a set of reproducible tests, the procedures used and the results from the operations as well as the results expected based on the program design.

13.2.1. Requirements testing

The following tests affect the view module in the MVC architecture and are carried out as black box tests.

Table 13. Test for *Log in*

Item tested	Log in				
Purpose	Verifying login information validity and security				
Precondition	Valid internet access; internet browser open, application page open				
Procedure	1. Enter username 2. Enter password 3. Select 'Log in'				
Condition	**Username**	**Password**	**Expected result**	**Tested result**	**Status**
Valid entries	john_jones	jameson	Logged in	Logged in	Pass
Invalid entries	john_jones	Jameson	Error	Error	Pass
		jamson	Error	Error	Pass
		3asdf87t8	Error	Error	Pass
		"blank"[1]	Error	Error	Pass
	John_jones	jameson[2]**	Error	Error	Pass
Blank entries	"blank"*	"blank"*	Error	Error	Pass
Same entries	john_jones	john_jones	Error	Error	Pass

Table 14 . Test for *Log out*

Item tested	Log out			
Purpose	Verifying user information removed upon logging out			
Precondition	User is currently logged in to the application			
Procedure	1. Click on the logout button			
Condition	**Action**	**Expected result**	**Tested result**	**Status**
Logout	Press logout button	Student information removed; restore default configuration; courses list restored to default; schedule field, constraint cleared.	Student information removed; restore default configuration; courses list restored to default; schedule field, constraint cleared.	Pass

Table 15. Test for *Course display*

Item tested	Course display			
Purpose	Check to see if a logged in student can see courses that are available to him			
Precondition	User is logged in			
Procedure	1. User selects his or her faculty 2. User selects his or her department 3. User selects his or her program 4. User selects a term			
Condition	**Action**	**Expected result**	**Tested result**	**Status**
Display courses	Only select faculty, but not the department, program or term.	No courses shown.	No courses shown.	Pass
	Select faculty and department, but not the program or term.	No courses shown.	No courses shown.	Pass
	Select faculty and department and program, but not the term.	Courses available shown in green, courses taken in grey, courses with missing prerequisite in red.	Courses available shown in green, courses taken in grey, courses with missing prerequisite in red.	Pass
	Select faculty, department, program and a term.	Courses available shown in green, courses taken in grey, courses with missing prerequisite in red.	Courses available shown in green, courses taken in grey, courses with missing prerequisite in red.	Pass

[1] "blank" here means nothing was entered, not the string "blank"
[2] Assuming jameson is also not the password for the 2nd user John_jones

Table 16. Test for *Add course*

Item tested	Add course			
Purpose	Verify that a course can be added to pool of selected classes			
Precondition	User logged in, there are courses being displayed			
Procedure	1. Click on a course from the course display list 2. Repeat until satisfied, or reached maximum course			
Condition	**Action**	**Expected result**	**Tested result**	**Status**
Add a course	Click on green course	Adds course to pool of selected courses.	Adds course to pool of selected courses.	Pass
Completed course	Click on grey course	Does nothing. No course added.	Does nothing. No course added.	Pass
Course with incomplete prerequisite	Click on red course	Error message: "Prerequisite missing"; no course added.	Error message: "Prerequisite missing"; no course added.	Pass
Full course load	Click on green course	Error message: "Maximum course load reached"; no course added.	Error message: "Maximum course load reached"; no course added.	Pass

Table 17. Test for *Remove course*

Item tested	Remove course			
Purpose	Verify courses can be removed from selected class pool once added			
Precondition	There must be courses added to the selected course pool			
Procedure	1. Click on a course within the "selected course" pool 2. Repeat until satisfied or no courses left in pool			
Condition	**Action**	**Expected result**	**Tested result**	**Status**
Remove course	Click on a course in pool	Course removed from pool.	Course removed from pool.	Pass

Table 18. Test for *Display schedule*

Item tested	Display schedule			
Purpose	Check if a schedule properly displays a variety of schedule based on added courses			
Precondition	Courses in selected course pool			
Procedure	- (automatic display)			
Condition	**Action**	**Expected result**	**Tested result**	**Status**
No course in pool & no constraints	- (do nothing)	Empty Display	Empty Display	Pass
	Add a course	Shows time slot taken by course in a weekly schedule. If course has more than one permutation, will display a number of available schedules.	Shows time slot taken by course in a weekly schedule. If course has more than one permutation, will display a number of available schedules.	Pass
No course in pool, numerous constraints	- (do nothing)	Display chosen time constraints in the weekly schedule	Display chosen time constraints in the weekly schedule	Pass
	Add a course	Display the first possible schedule with no conflicts with the given time constraints and a number of schedules that fit the criteria.	Display the first possible schedule with no conflicts with the given time constraints and a number of schedules that fit the criteria.	Pass
Schedule completely filled with constraints	Do nothing	Weekly schedule displays full constraint.	Weekly schedule displays full constraint.	Pass
	Add course	Weekly schedule displays full constraint, no course added to schedule.	Weekly schedule displays full constraint, no course added to schedule.	Pass
Courses in pool and numerous constraints	Do nothing	Displays courses and constraints in weekly schedule with list of potential schedules.	Displays courses and constraints in weekly schedule with list of potential schedules.	Pass
	Add a course	Display updated weekly schedule with new course if it fits. Update the list of possible schedules.	Display updated weekly schedule with new course if it fits. Update the list of possible schedules.	Pass
	Remove a course	Display updated weekly schedule without the removed course. Update the list of possible schedules.	Display updated weekly schedule without the removed course. Update the list of possible schedules.	Pass

Table 19. Test for *Choose schedule*

Item tested	Choose schedule			
Purpose	Make sure choosing that a logged in user can successfully choose and save a selected schedule from among a list of generated ones fitting the user's course and constraint requirements.			
Precondition	User is logged in; there is at least 1 course chosen, and 1 schedule to choose from			
Procedure	1. Select 'Save schedule'			
Condition	**Action**	**Expected result**	**Tested result**	**Status**
Empty schedule	Select save	Error: "This schedule is empty".	Error: "This schedule is empty"	Pass
Schedule with full constraint, and no courses		Error: "You have not chosen any courses".	Error: "You have not chosen any courses"	Pass
Schedule with at least one course		Successfully saved the schedule.	Successfully saved the schedule.	Pass

13.2.2. Unit testing

The following section is done entirely following the white box testing principle, and primarily tests functions in the controller module of the MVC architecture. These are functions that calculate and manipulate data from the model into something that the view can display.

13.2.2.1. Function CompareTime(Time1, Time2)

This function first transforms a "time" string taken from the database and turns into comparable set of integers, separated into hours and minutes. It checks to see if Time1 is before, after or equal to Time2. It will return a zero if both Time1 and Time2 are equal, returns -1 if Time2 is after Time1, and returns 1 when Time1 is after Time2.

Table 20. Test for function *CompareTime*

Item tested	Times: 14:45, 14:30; 11:45, 18:00; 15:15, 08:45; 1, 999; asdf, 3214894			
T1	**T2**	**Expected result**	**Tested result**	**Pass/fail**
14:45	14:30	1	1	Pass
14:45	14:45	0	0	Pass
11:45	18:00	-1	-1	Pass
15:15	08:45	1	1	Pass
X*	999	Error: "Invalid input."	Error: "Invalid input."	Pass
1800	X*	Error: "Invalid input."	Error: "Invalid input."	Pass

13.2.2.2. Function CompareDay(Day1, Day2)

This function takes two strings representing days of the week as arguments. These strings are in the form of "M-W--" where the letters represent days that the course is being held, and the dashes meaning there are no courses on that day. It then calls the function *DayStr* that converts those days into an array of integers, where "1" represents days with class, and "0" days without class. It then iterates through both arrays, and returns *false* there is a "1" at the same index of both arrays.

Table 21. Test for function *CompareDay*

Item tested	Days: Monday & Wednesday; Wednesday; Tuesday and Thursdays				
Condition	**Week 1**	**Week 2**	**Expected Result**	**Tested Result**	**Pass/Fail**
No conflicting days	M-W--	-T-H-	True	True	Pass
One conflicting day	M-W--	--W--	False	False	Pass
Multiple conflicting days	M-W--	M-W--	False	False	Pass

13.2.2.3. function IsTimeConflict(D1,S1,E1,D2,S2,E2)

This function takes two days, two starting times and two ending times as arguments. It checks if there are any days that overlap and if so if there is a time conflict.

This is a lengthy test relying on entering a series of different days, some with overlapping days, but with no time overlap, some with no day overlap, some with both day and time overlaps. This test returns *true* if there is a time conflict and *false* otherwise.

Table 22. Test for function *IsTimeConflict*

Condition	Day1	Day2	Start1	Start2	End1	End2	Expected	Tested	Pass/fail
No day overlap	Monday	Tuesday	11:45	11:45	13:00	13:00	False	False	pass
Day overlap but no time overlap	Monday	Monday	11:45	14:45	13:00	16:00	False	False	Pass
	Tuesday	Tuesday	16:15	08:45	17:30	10:00	False	False	Pass
	Friday	Friday	8:45	10:15	10:00	11:30	False	False	Pass
Day overlap with time overlap	Monday	Monday	8:45	10:15	11:30	11:30	True	True	Pass
	Monday	Monday	14:45	14:45	16:00	17:30	True	True	Pass
	Monday	Monday	11:45	10:15	13:00	13:00	True	True	Pass

13.2.3. Functional requirements analysis

This subsection details the fundamental capabilities of the application as required by the scope of the project within a concrete scenario and the intended system reaction. It also elaborates on security, reliability and individual functions of importance regarding the segments of the scenario and the expected results of user actions.

13.2.3.1. Accessing the site

- The user can access the application at the address where it was installed using a supported browser.
- The application displays to the user the main login screen with the choice to continue as a guest or a user.

13.2.3.2. Login

- The user can view the page where login credentials are entered.
- The password is not displayed in plain text.
- If the user enters correct credentials, the system will log them in.
- If a user has logged on to multiple accounts from the same IP address, the system will lock them out temporarily.
- If the user enters incorrect credentials, the system presents an error message and does not permit entry.
- After 10 failed login attempts, the system locks the user out for 30 minutes.
- The user is able to logout of the application.
- Once the user logs out, the session is cleared and cached private information pertaining to that account cannot be viewed anymore.

13.2.3.3. Schedule

- The user can see the schedule layout after logging in with valid credentials or as a guest user.
- The schedule is automatically updated with the latest information that the user has saved to their account. This does not apply to guest accounts.

13.2.3.4. Course listing

- The course list can be accessed from the schedule page and has the latest available information.
- The user can also view all courses in an inputted discipline or department.
- The user’s program sequence is shown on the program listing page as a guide to what courses the user should be searching for. It only shows courses which the user has not yet completed.
- No program sequence is shown in guest accounts.

13.2.3.5. Schedule generator

- The schedule generator functions without noticeable lag or delay.
- The user can input “break times” where no courses are to be scheduled.
- The system shows all possible schedule variants based on the user’s choice of courses and break times.
- The user can save selected schedules.
- There is a print-friendly option available for every schedule variant that prints properly on 8.5 x 11 letter sized paper.
- An error is shown if no schedule alternatives can be generated.

13.2.3.6. General

- User data is saved when logging out.
- The layout of the application is the same for all user accounts.
- The system displays an “overloaded” message should there be more than 5,000 concurrent users.

13.2.4. Stress testing

Stress testing is a form of testing performed on systems in order to observe, analyze and verify how a system reacts when pushed to extreme conditions. The following are various hypothetical tests that could be used for this purpose.

The first possible test would be to send many login requests to the system at the same time. This could be repeated by making sure to increase the number of login requests each time. It would be useful to do this in order to observe the systems limit of simultaneous login requests that could be sent.

A second possible test would be to try and exceed the maximum number of accounts that could be created on the system. One reason to do this is to observe the correlation between the size allocated to the system and the maximum number of accounts that can be created. Another reason would be to observe how the system would react when this limit is reached.

Another possible test would be to try running the system without complying with the minimal system requirements. This would allow the observation of the system reaction to these suboptimal conditions.

13.3. Security testing

The project makes extensive use of a back-end database to store and retrieve information. Therefore, SQL injection vulnerability is of prime concern to the system.

First, the system is to be tested for elementary vulnerabilities. If such vulnerabilities are discovered, potential solutions will be examined. When a solution is chosen, more elaborate testing schemes will be implemented. If the first stage succeeds without demonstrating any vulnerabilities, more elaborate testing schemes can be implemented. The result of these tests will determine the next steps, which include either use of a third party solution or developing our own injection protection solution.

SQL Inject-Me is Firefox extension used to test for SQL Injection vulnerabilities. It has been considered as a tool to perform automatic testing for such vulnerabilities. However, SQL Inject-Me only performs test on forms. The first and most elementary vulnerability is when servicing a GET request: the user may change the URL manually and try a brute force attack that way. Therefore, SQL Inject-Me does not conform to the stated goal to test for the simplest attacks first and has been discarded.

Furthermore, more elaborate tools, such as Nikto (a web server scanner which performs comprehensive tests against web servers) have been considered but all discarded as too elaborate and as such impractical, since they require too many resources for basic testing.

As a result, brute-force testing has been performed: one of the GET parameters in the URL has been surrounded by two quotes. The fact that the SQL query failed immediately demonstrates a simple and easy to perform vulnerability of the system. It is now known that the database is not protected from malicious (or accidental) injection. Third party libraries to deal with this security threat must be considered.

14. System delivery

This section describes how to install the application and how to run it on a local machine.

14.1. Installation manual

The typical installation process is a general description of the installation procedure. It briefly shows all the different components that are needed to start using the website. In this installation manual only a local installation of the website will be covered.

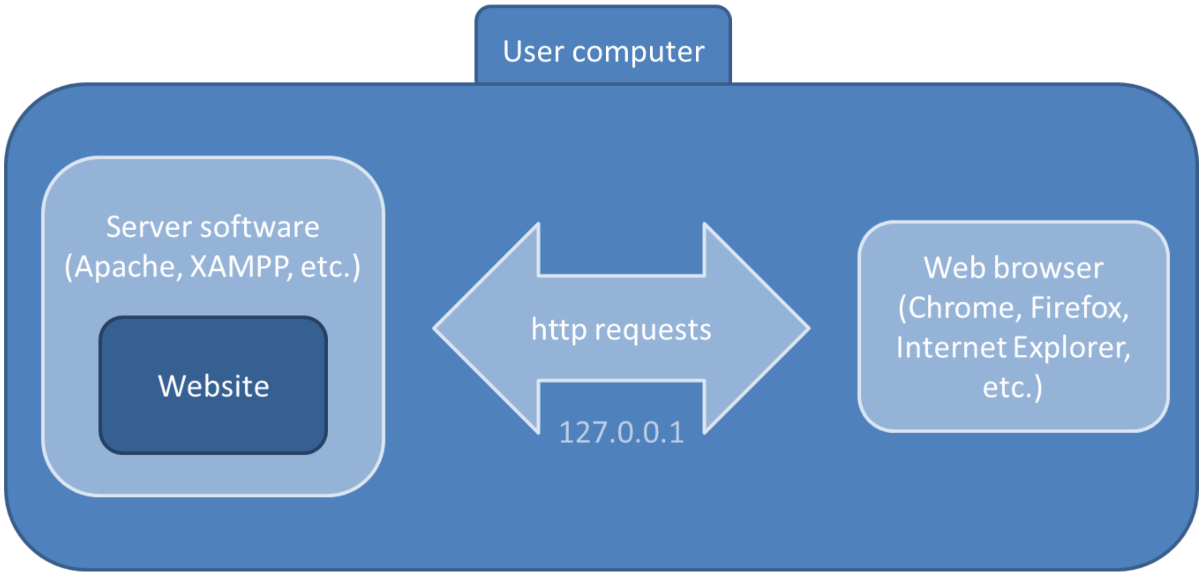

User computer
Server software (Apache, XAMPP, etc.)
Website
http requests
127.0.0.1
Web browser (Chrome, Firefox, Internet Explorer, etc.)

Figure 20. Local installation architecture

14.1.1. Installing the server (XAMPP) on windows

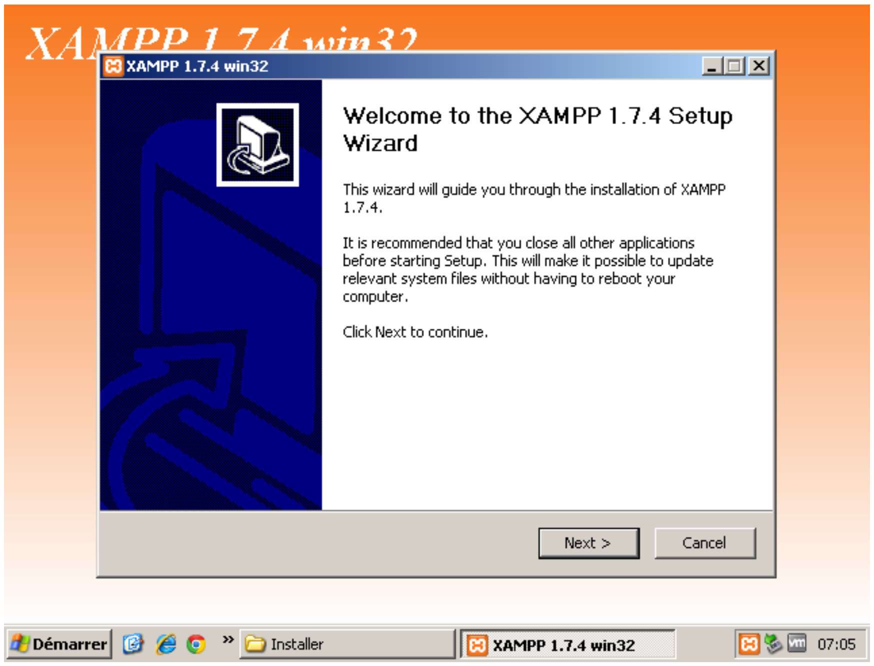
A web server is needed to run the website locally on our machine. It will be in charge of sending back and forth the information of our website to our web browser. In this manual we will show how to install, configure and use a popular web server package called XAMPP. XAMPP comes with a predefined set of utilities that are essential to the website such as MySQL, Apache, and PHP.

14.1.1.1. Getting XAMPP

XAMPP can be downloaded at http://www.apachefriends.org/en/xampp-windows.html or installed from the distributed CD or compressed files (XAMPP installation executable: xampp-win32-1.7.4-VC6-installer.exe)

14.1.1.2. Running the installation files

Double-click on the xampp-win32-1.7.4-VC6-installer.exe file.



Welcome to the XAMPP 1.7.4 Setup Wizard

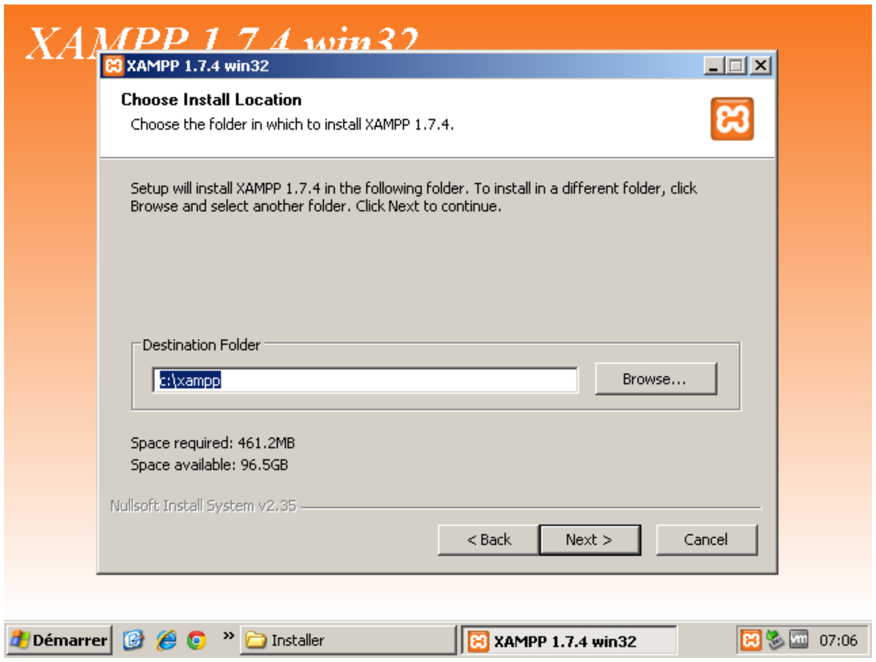
This wizard will guide you through the installation of XAMPP 1.7.4.

It is recommended that you close all other applications before starting Setup. This will make it possible to update relevant system files without having to reboot your computer.

Click Next to continue.

Figure 21. XAMPP setup wizard

- Click on the Next > button.



Choose Install Location

Choose the folder in which to install XAMPP 1.7.4.

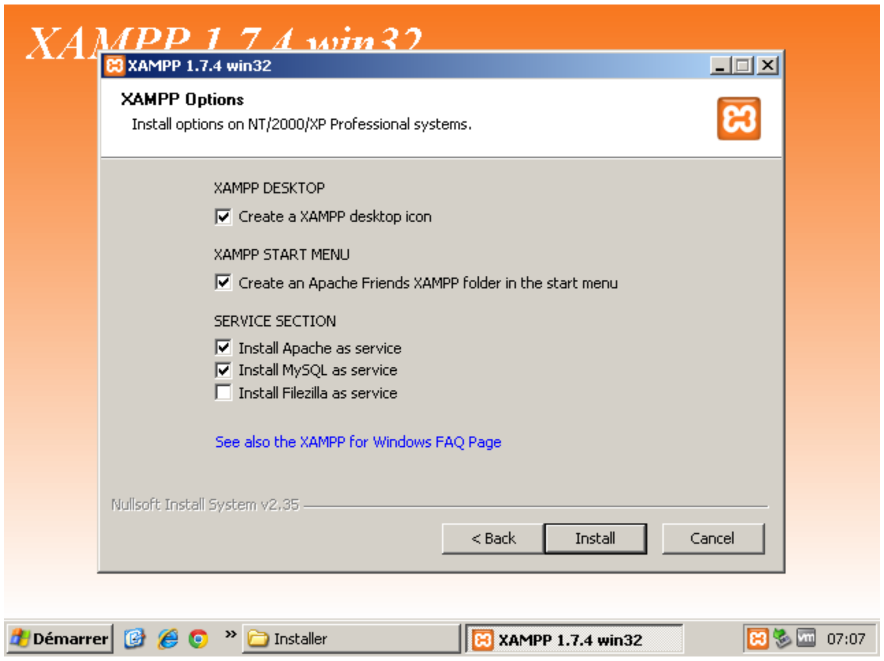
Setup will install XAMPP 1.7.4 in the following folder. To install in a different folder, click Browse and select another folder. Click Next to continue.

Destination Folder: c:\xampp

Space required: 461.2MB
Space available: 96.5GB

Figure 22. XAMPP installation folder

- Choose where to install XAMPP in c:\xampp\ and click on the Next > button.



XAMPP Options
Install options on NT/2000/XP Professional systems.

XAMPP DESKTOP
- [✓] Create a XAMPP desktop icon

XAMPP START MENU
- [✓] Create an Apache Friends XAMPP folder in the start menu

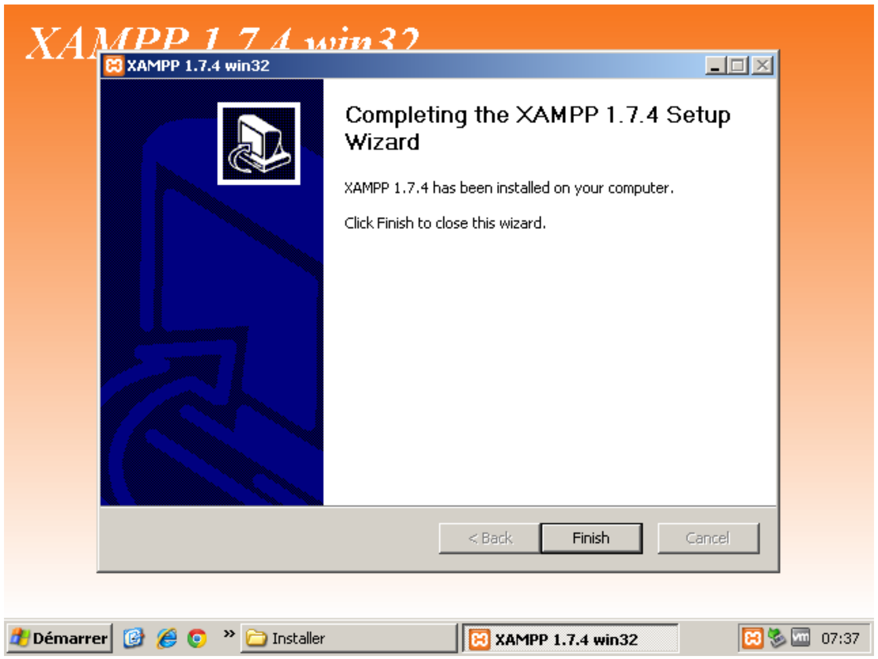
SERVICE SECTION
- [✓] Install Apache as service
- [✓] Install MySQL as service
- [ ] Install Filezilla as service

See also the XAMPP for Windows FAQ Page

Nullsoft Install System v2.35

Figure 23. XAMPP installation options

- Select Install Apache as service.
- Select Install MySQL as service.
- Click on the Install button.



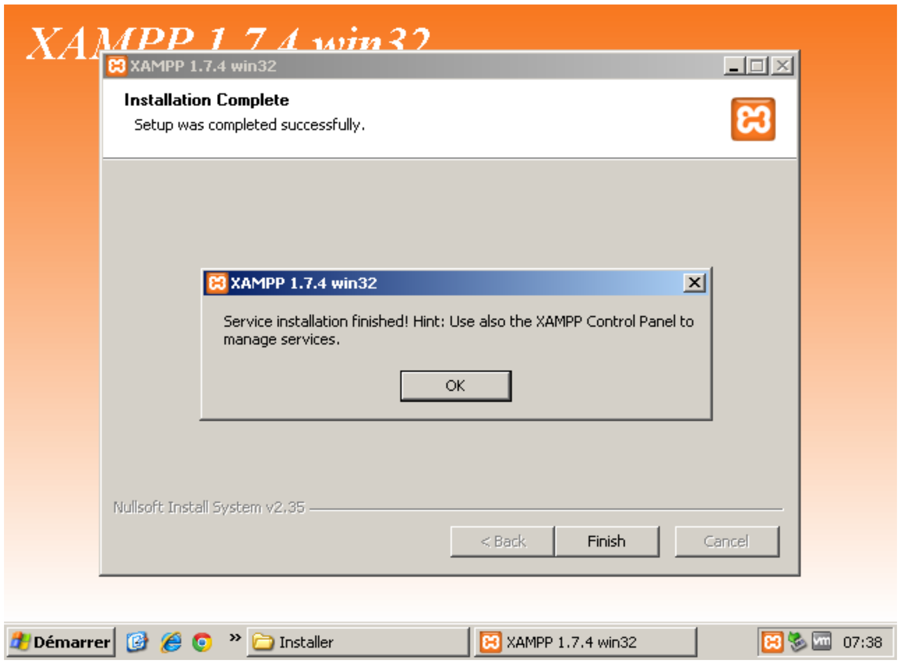
Completing the XAMPP 1.7.4 Setup Wizard

XAMPP 1.7.4 has been installed on your computer.

Click Finish to close this wizard.

Figure 24. XAMPP completed installation

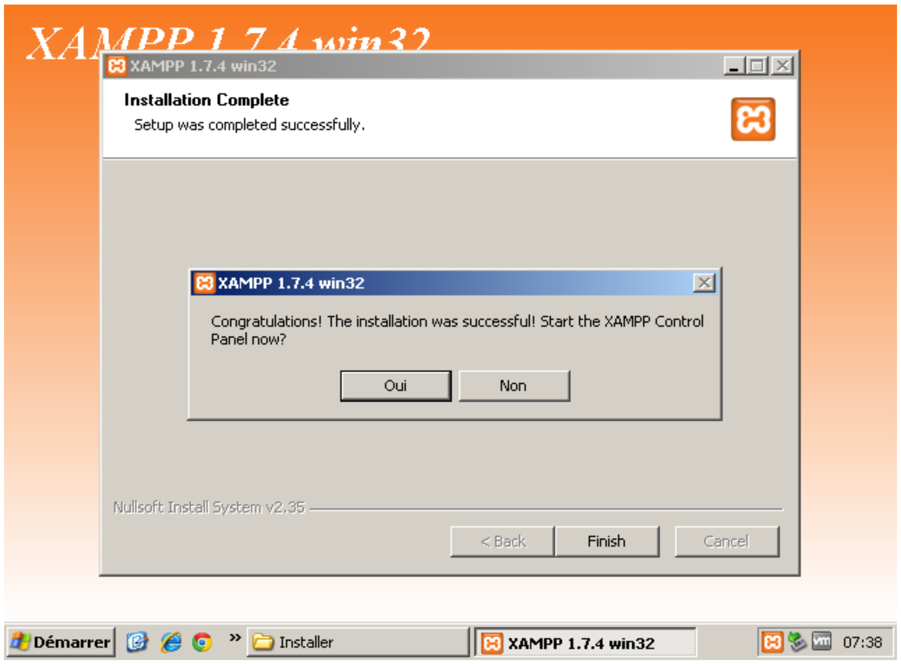
- Click on the Finish button.



Service installation finished! Hint: Use also the XAMPP Control Panel to manage services.

Figure 25. XAMPP service installation message

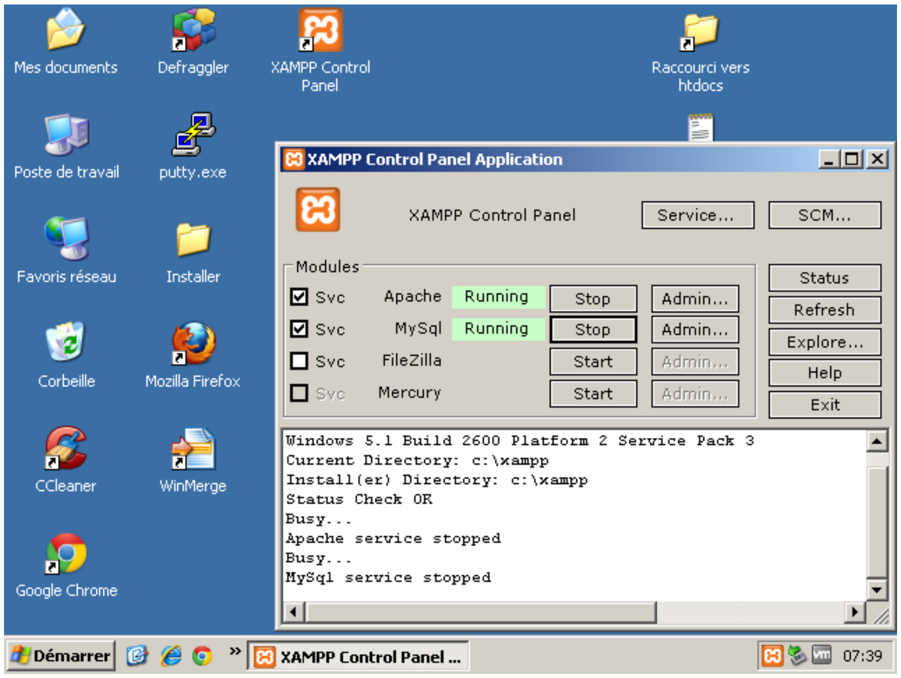
- Click on the OK button.



Congratulations! The installation was successful! Start the XAMPP Control Panel now?

Figure 26. XAMPP Control Panel message

- Click on the Yes/Oui button.



```
Windows 5.1 Build 2600 Platform 2 Service Pack 3
Current Directory: c:\xampp
Install(er) Directory: c:\xampp
Status Check OK
Busy...
Apache service stopped
Busy...
MySql service stopped
```

Figure 27. Checking the status of the Apache and MySQL services

- The XAMPP icon should appear on the taskbar at the bottom right of your screen.
- Click on the XAMPP icon to open the XAMPP Control Panel Application.
- Make sure that Apache and MySQL are started and running.

14.1.2. Installing the website in the XAMPP server.

In order to install the website on the XAMPP server, simply copy the entire website folder (*i.e.* SOEN341) into the C:\xampp\htdocs folder.

14.1.3. Test the application

Using a browser, navigate to http://127.0.0.1/SOEN341/index.html to access the application.

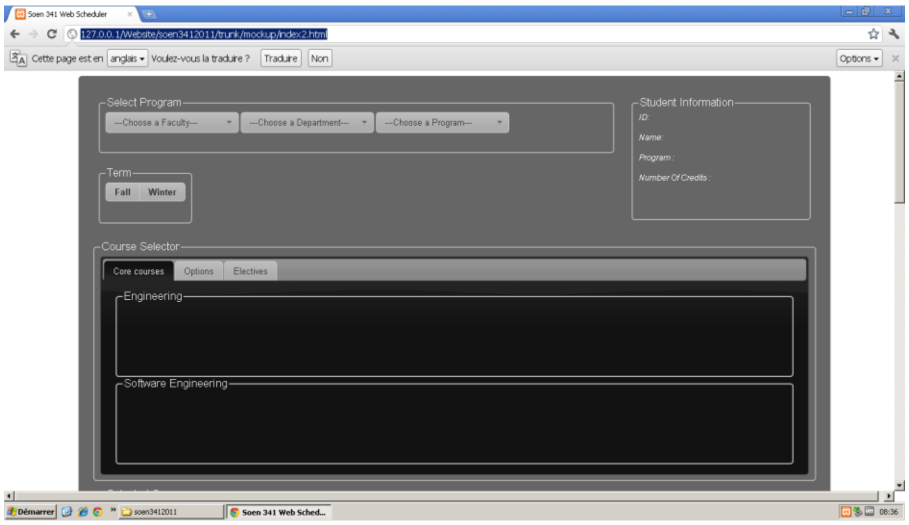


Figure 28. Testing the application installation

14.2. User manual

14.2.1. Introduction

The user manual is divided into two parts. The first part describes the use of the application without logging in, the second with logging in. The basic instructions are the same in both cases, but the second case (*i.e.* with log in) offers more functionality by providing information tailored for the logged on user.

14.2.2. User manual

At Concordia University, students must build schedules for the fall and winter semester every April. They must choose from a preselected list of courses and find sections that will result in no time conflicts with other courses or personal commitments. This involves spending hours drawing out sample schedules on paper until the best alternative is found, often after countless queries to the online course schedule database. It is common to miss the ideal schedule simply because an available option was overlooked.

SOEN341 Planner will allow you, the student, to choose the courses that you want to take, when you want to take them and search for every possible variant of the schedule.

SOEN341 Planner puts you in charge: making the perfect schedule was never that easy!

14.2.2.1. Getting started (any student)

Start the application by opening your browser and navigating to:

http://127.0.0.1/SOEN341/index.html

Select your faculty, department and program from the dropdown boxes at the top of the window.

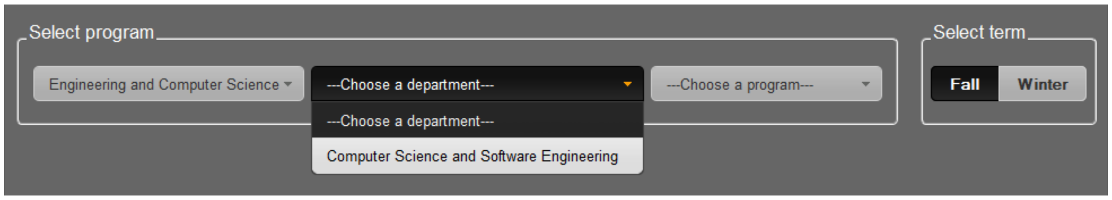

Select program
Engineering and Computer Science
---Choose a department---
---Choose a department---
Computer Science and Software Engineering
---Choose a program---
Select term
Fall
Winter

Figure 29. Select your program and term

The courses that constitute the selected program will appear in the course selection area grouped into core courses, options and electives. When you hover over a course with your mouse, more information about the course will appear.

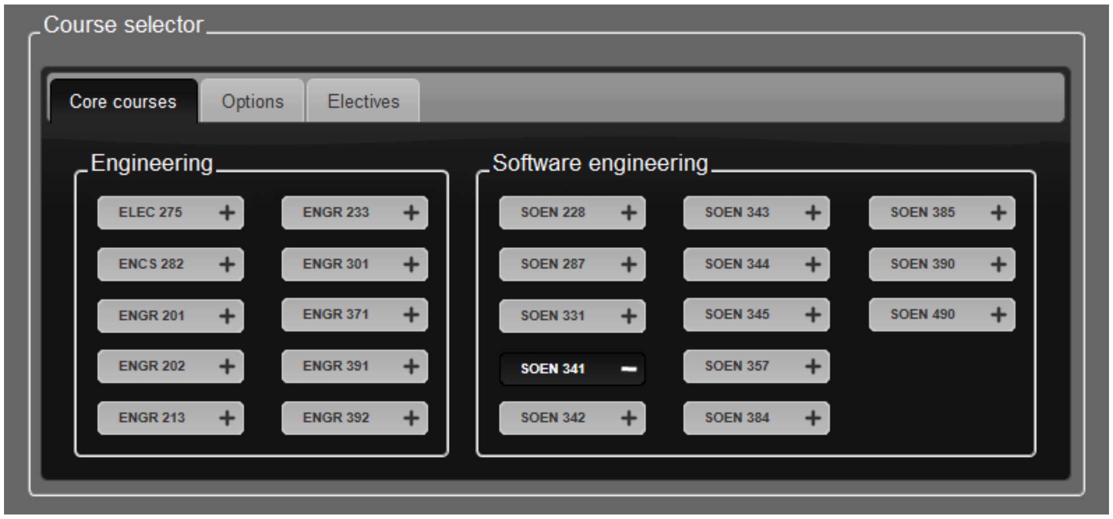


Figure 30. Select from the program courses

Before proceeding, please select the term for which you want to select courses. Then, browse the tabbed course selection area to find the courses you would like to take in this term. If a course is not offered in the selected term, it will be grayed out. Click on the plus sign to add a course to the list of selected courses, click the minus sign to remove a course from the list. You are free to add as many courses as you like.

Optionally, you can add time constraints to your schedule by defining time slots of any length for a given day. In order to do this, select a day from the dropdown box and define the time slot with the slider control. The left slider corresponds to the start time of the constraint and the right slider corresponds to the end time constraint. Click on 'Add' to add the constraint to your schedule. The newly added constraint is displayed as a red box in the schedule. Click 'Clear all' to remove all time constraints from the schedule display.

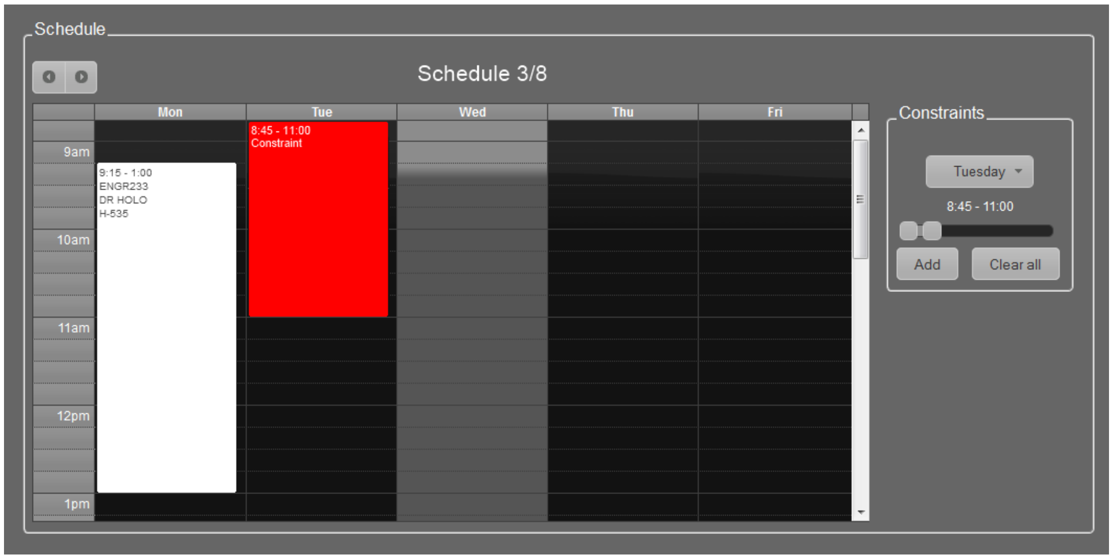


Figure 31. Schedule display and constraints

Click on 'Generate schedule' to see possible schedules for the course you selected and the time constraints you added. The resulting schedule will be displayed in the form of a weekly calendar indicating your courses and constraints.

If your courses and time constraints result in more than one possible schedule, navigation arrows appear above the schedule area and the current and total number of schedules is shown.

Browse to a schedule that meets your approval and click 'Save schedule' to save the schedule to a file. Click on 'Print schedule' to be presented with a printable view of your chosen schedule.

14.2.2.2. Getting started (student with username and password)

Start the application by opening your browser and navigating to:

http://127.0.0.1/SOEN341/index.html

Enter your username and password in the corresponding boxes and click 'Log in'. As a result, your personal information will be shown in the student information box and the program you are registered in will be automatically selected (including faculty and department information).

The courses that constitute the selected program will appear in the course selection area grouped into core courses, options and electives. When you hover over a course with your mouse, more information about the course will appear. The course selection area shows you the courses you have successfully passed in grey, the courses you can take in green and the courses you cannot take due to missing prerequisites in red. You can only select green courses to add to your schedule. If a course is not offered in the selected term, it will be grayed out.

You are free to add as many courses as you like. The application will show you if the selected number of courses corresponds to your status as full or part-time student.

Optionally, you can add time constraints to your schedule by defining time slots of any length for a given day. In order to do this, select a day from the dropdown box and define the time slot with the slider control. The left slider corresponds to the start time of the constraint and the right slider corresponds to the end time. Click on 'Add' to add the constraint to your schedule. The newly added constraint is displayed as a red box in the schedule. Click 'Clear all' to remove all time constraints from the schedule display.

Click on 'Generate schedule' to see possible schedules for the course you selected and the time constraints you added. The resulting schedule will be displayed in the form of a weekly calendar indicating your courses and constraints.

If your courses and time constraints result in more than one possible schedule, navigation arrows appear above the schedule area and the current and total number of schedules is shown.

Browse to a schedule that meets your approval and click 'Save schedule' to save the schedule to a file. Click on 'Print schedule' to be presented with a printable view of your chosen schedule.

15. Final cost estimate

The following table determines the final cost estimates. It lists all the components of all the phases of this project and specifies the person hours cost of each component.

Table 23. Final cost estimate

Phase	Activity Name	Cost	Phase cost
1			4
	E-mail exchange	1	
	Individual work on the systems overview document	2	
	Collate and assemble completed task.	1	
2			6
	Introductions and set up communications	1	
	Set up group framework	1	
	Discussion of project requirements and brainstorming	1	
	Decide on technical resources and generalized scope	3	
3			40
	Finalized prototype	4	
	Dividing the work for the scope and plan document	1	
	Completed SVN server	3	
	Completed databases	32	
4			70
	Implementing basic application interface	20	
	Implementing classes and functions	50	
5			65
	Completion of architectural design	13	
	Completion of detailed design	12	
	Completion of dynamic design	10	
	Prototype and risk report and completed report	20	
6			120
	Final application testing (Designing test cases, System testing)	40	
	Product debugging	40	
	Quality testing	40	
7			10
	Finalize report	5	
		Total cost	**315 hours**

15.1. Phase 1: Systems overview document

15.1.1. Activity 1

Email exchange. The initial team members from the group of 5 exchanged email addresses. The group grew larger afterwards, so only 5 of 9 could be reached at the given time.

15.1.2. Activity 2

Individual work on the systems overview document. As we did not know each other or our experience yet, we simply assigned tasks to those who were present at the time.

Output: Parts of the systems overview document.

15.1.3. Activity 3

Collate and assemble completed tasks into one formal document. All assigned tasks were submitted to one member to assemble into a single cohesive document.

Output: Systems overview document.

15.2. Phase2: First team meeting

15.2.1. Activity 4

Introductions and set up communications. We introduced ourselves to each other with the goal of determining what kind of experience and knowledge we can pool together. Then we exchanged email addresses and contact numbers between key team members. We also settled on a team leader and established a Google group to share data and work.

Output: Google Group, communications channel, human resources.

15.2.2. Activity 5

Set up group framework. Assigning temporary positions based on experience, program, knowledge and availabilities of team members, and organizing future team meetings based on everyone’s availability. Amine volunteered to set up an SVN server for the group, and Jenia chose to work on setting up the database.

Output: Tasks assignment and availability schedule.

15.2.3. Activity 6

Discussion of project requirements and brainstorming. We discussed ideas, and sorted through those that can be handled and implemented as soon as possible. We also started our preliminary design on what our end product should do.

Output: Preliminary design.

15.2.4. Activity 7

Decide on technical resources and generalized scope. We chose to use PHP, MySQL and JavaScript as most team members are familiar with the above, making it easier for others to work together.

Output: Determined scope.

15.3. Phase 3: Second team meeting

15.3.1. Activity 8

Finalized design. Put the finishing touches on our preliminary design and decided on the functional requirements and qualities of the application.

Output: Design prototype.

15.3.2. Activity 9

Dividing the work for the Scope and Plan Document. Assign sections of the Scope and Plan document that needs to be done amongst those not primarily engaged in the creation of SVN server, the data bases and the preliminary design.

Output: Work division.

15.3.3. Activity 10

Completed SVN server. Walk-through of the capabilities and functions of our SVN server and tutorial.

Output: SVN server.

15.3.4. Activity 11

Completed databases. Completion of the various databases we will require in order to properly run the application as well as a rundown of the uses for each database and how to use them within our application.

Output: Student database, course list database, program requirement database.

15.4. Phase 4: Coding the application

15.4.1. Activity 12

Implementing basic application interface. Starting to code some of the functional requirements of the application. Mostly the displays for the main page, including the course display, term toggle button, constraints window, course load and login/logout.

Output: Basic application framework completed.

15.4.2. Activity 13

Implementing classes and functions. Start coding the functional parts of the application. Ultimately should be able to generate at least one schedule.

Output: Completed application, but untested.

15.5. Phase 5: Third deliverable

15.5.1. Activity 14

Completion of detailed design. Completion of the in depth class diagrams and the various explanations and clarifications, as well as the unit description of every class, and the dynamic design scenarios.

Output: Detailed design portion complete.

15.5.2. Activity 15

Completion of architectural design. Completion of the architecture diagram and the rationale behind it, as well as the subsystem interfaces specifications and the description for their parameters.

Output: Architectural design portion complete.

15.5.3. Activity 16

Completion of dynamic design. Completion of the dynamic design diagrams, which are the system sequence diagrams and sequence diagrams. List of operational contracts of each scenario.

Output: Detailed design portion complete.

15.5.4. Activity 17

Prototype and risk report and completed report. Complete the report of prototype functions and whether or not they will be scoped out ultimately by the end of the project given current time constraints. Completion and collation of the third deliverable.

Output: Completed architecture and design document.

15.6. Phase 6: Product testing

15.6.1. Activity 18

Testing the final application. Design suitable test cases. Hard testing the application's functional requirements. Verify implementation versus requirements.

Output: Requirements and function progress.

15.6.2. Activity 19

Product debugging. Testing application for errors and problems that prevent normal use of the application and fixing them.

Output: Error Log.

15.6.3. Activity 20

Quality testing. Testing non-functional requirements like usability, reliability, performance and security.

Output: Quality assurance log.

15.7. Phase 7: Final report

15.7.1. Activity 21

Finalized report. Unifying all parts of the final submission worked on by the various members and combining all previous submissions into a cohesive and complete document. User manual and installation manual included.

Output: Final Document.

Faculty of Engineering and Computer Science Expectations of Originality

This form has been created to ensure that all students in the Faculty of Engineering and Computer Science comply with principles of academic integrity prior to submitting coursework to their instructors for evaluation: namely reports, assignments, lab reports and/or software. All students should become familiar with the University's Code of Conduct (Academic) located at http://web2.concordia.ca/Legal_Counsel/policies/english/AC/Code.html

Please read the back of this document carefully before completing the section below. This form must be attached to the front of all coursework submitted to instructors in the Faculty of Engineering and Computer Science.

Course Number: SOEN341 **Instructor:** Terry Fancott

Type of Submission (Please check off reponses to both a & b)

a. [✓ Report] Assignment Lab Report Software

b. Individual submission [✓ Group Submission (All members of the team must sign below)]

Having read both sides of this form, I certify that I/we have conformed to the Faculty's expectations of originality and standards of academic integrity.

Name: Astou Chanel ab Sene (please print clearly)	ID No: 9557679	Signature:	Date: 02/12/2011
Name: Jean-Pierre Escalante (please print clearly)	ID No: 9652264	Signature:	Date: 02/12/2011
Name: Ryan Held (please print clearly)	ID No: 9639985	Signature:	Date: 02/12/2011
Name: Jenia Ivlev (please print clearly)	ID No: 6191134	Signature:	Date: 02/12/2011
Name: Denis Lau (please print clearly)	ID No: 9746757	Signature:	Date: 02/12/2011
Name: Marwa Malti (please print clearly	ID No: 1916262	Signature:	Date: 02/12/2011

Do Not Write in this Space – Reserved for Instructor

1/2

Form ENCS-SAS (03/04)

Faculty of Engineering and Computer Science Expectations of Originality

This form has been created to ensure that all students in the Faculty of Engineering and Computer Science comply with principles of academic integrity prior to submitting coursework to their instructors for evaluation: namely reports, assignments, lab reports and/or software. All students should become familiar with the University's Code of Conduct (Academic) located at http://web2.concordia.ca/Legal_Counsel/policies/english/AC/Code.html

Please read the back of this document carefully before completing the section below. This form must be attached to the front of all coursework submitted to instructors in the Faculty of Engineering and Computer Science.

Course Number: SOEN341 **Instructor:** Terry Fancott

Type of Submission (Please check off reponses to both a & b)

a. ☑ Report ☐ Assignment ☐ Lab Report ☐ Software

b. ☐ Individual submission ☑ Group Submission (All members of the team must sign below)

Having read both sides of this form, I certify that I/we have conformed to the Faculty's expectations of originality and standards of academic integrity.

Name: Amine Najahi (please print clearly)	ID No: 9474102	Signature:	Date: 02/12/2011
Name: Rolf Schmidt (please print clearly)	ID No: 9538267	Signature:	Date: 02/12/2011
Name: (please print clearly)	ID No:	Signature:	Date:
Name: (please print clearly)	ID No:	Signature:	Date:
Name: (please print clearly)	ID No:	Signature:	Date:
Name: (please print clearly	ID No:	Signature:	Date:

Do Not Write in this Space – Reserved for Instructor

1/2

EXPECTATIONS OF ORIGINALITY & STANDARDS OF ACADEMIC INTEGRITY

ALL SUBMISSIONS must meet the following requirements:

1. The decision on whether a submission is a group or individual submission is determined by the instructor. Individual submissions are done alone and should not be identical to the submission made by any other student. In the case of group submissions, all individuals in the group must be listed on and must sign this form prior to its submission to the instructor.
2. All individual and group submissions constitute original work by the individual(s) signing this form.
3. Direct quotations make up a very small proportion of the text, i.e., not exceeding 5% of the word count.
4. Material paraphrased from a source (e.g., print sources, multimedia sources, web-based sources, course notes or personal interviews) has been identified by a numerical reference citation.
5. All of the sources consulted and/or included in the report have been listed in the Reference section of the document.
6. All drawings, diagrams, photos, maps or other visual items derived from other sources have been identified by numerical reference citations in the caption.
7. No part of the document has been submitted for any other course.
8. Any exception to these requirements are indicated on an attached page for the instructor's review.

REPORTS and ASSIGNMENTS must also meet the following additional requirements:

1. A report or assignment consists entirely of ideas, observations, information and conclusions composed by the student(s), except for statements contained within quotation marks and attributed to the best of the student's/students' knowledge to their proper source in footnotes or references.
2. An assignment may not use solutions to assignments of other past or present students/instructors of this course or of any other course.
3. The document has not been revised or edited by another student who is not an author.
4. For reports, the guidelines found in Form and Style, by Patrick MacDonagh and Jack Borden (Fourth Edition: May 2000, available at http://www.encs.concordia.ca/scs/Forms/Form&Style.pdf) have been used for this submission.

LAB REPORTS must also meet the following requirements:

1. The data in a lab report represents the results of the experimental work by the student(s), derived only from the experiment itself. There are no additions or modifications derived from any outside source.
2. In preparing and completing the attached lab report, the labs of other past or present students of this course or any other course have not been consulted, used, copied, paraphrased or relied upon in any manner whatsoever.

SOFTWARE must also meet the following requirements:

1. The software represents independent work of the student(s).
2. No other past or present student work (in this course or any other course) has been used in writing this software, except as explicitly documented.
3. The software consists entirely of code written by the undersigned, except for the use of functions and libraries in the public domain, all of which have been documented on an attached page.
4. No part of the software has been used in previous submissions except as identified in the documentation.
5. The documentation of the software includes a reference to any component that the student(s) did not write.
6. All of the sources consulted while writing this code are listed in the documentation.

Important: Should you require clarification on any of the above items please contact your instructor.