WPI SCHEDULER IMPROVEMENT PROJECT

Interactive Qualifying Project Report completed in partial fulfillment
of the Bachelor of Science degree at
Worcester Polytechnic Institute, Worcester, MA

Submitted to:
Professor Michael Ciaraldi (advisor)

Ryan Danas

Douglas Lally

Henrique Polido

(Date of submission)

_____
Advisor Signature

Abstract

The WPI Scheduler Improvement Project intends to improve the class scheduling process for students at Worcester Polytechnic Institute. Existing scheduling methods, including the WPI Scheduler (http://wpischeduler.org), have inadequacies which lead to incorrect behavior in the application. The need for a specialized application for scheduling classes at WPI was identified, and provided the basis for this project.

Table of Contents

Authorship Page

Ryan Danas assisted with group research and design, acted as co-programmer[1], and co-authored the project report. Douglas Lally handled meeting minutes and agendas, assisted with the preliminary design, acted as auxiliary programmer[2], and co-authored the project report. Henrique Polido acted as lead programmer[3] and designer[4], as well as managing the development cycles and task delegation.

[1] Co-programmer: Secondary coder who implemented a portion of the application
[2] Auxiliary Programmer: Primary implementer of middleware needed to properly serve data in the format the application expects
[3] Lead Programmer: Primary coder who implemented the most significant portion of the application
[4] Lead Designer: Most influential role in planning the overall application design, as well as deciding the best option when design changes are needed

I. Introduction

Scheduling classes at WPI has always been a complicated process. Until recently, the only option for choosing classes used to be the Banner Web Information System (Bannerweb). Through this website, students can access class information, which included dozens of sections for most classes in every department, usually taught by different professors at varying times. Even without factoring in waitlists, cross listed courses, experimental class offerings, and degree requirements, sifting through all this data was a time consuming task. Many students had issues just coming up with a schedule without conflicting classes, let alone one that they actually enjoy.

> "The (original) WPI Scheduler Project aimed to develop a new course scheduling system for the students of Worcester Polytechnic Institute by modifying and extending the open source University Scheduler originally developed by Keith Lea. This course scheduling system will provide students and their advisors an easier, more convenient method of creating course schedules for an entire academic year." (Hawkins, Lucia and Rubio)

However, the needs of WPI students have changed since the development of the original WPI Scheduler in 2007. In order to address these issues, the team planned on creating a new scheduling application which improved upon the previous WPI Scheduler's design. Our approach was to develop this application based on student feedback and research into modern technologies. Employing modern development libraries and compilers helped us create an application that was simple to access, easy to use, and platform independent.

The previous scheduling application is a program adapted to work at WPI from University Scheduler[5], an open source program. University Scheduler was written in Java using IntelliJ[6], which is a pay-to-use Integrated Development Environment (IDE). Our group identified issues in the previous

[5] http://sourceforge.net/projects/unischeduler/
[6] http://www.jetbrains.com/idea/

application, including program crashes, and creation of schedules with conflicts. University Scheduler was designed to create schedules for two-semester schools, and as WPI uses a four term system. Therefore, the previous project team had to modify the code to work with WPI's four term system. Modifying the code complicated the program structure, making it difficult to debug. Any further changes to the original project risked introducing new issues to the program, which require additional time and effort to fix. Furthermore, advancements in software have provided other avenues of development for a scheduling application. Our group elected to create a totally new scheduling program, tailored to work at WPI in order to address all of the identified inadequacies.

II. Methodology

Identifying Inadequacies in Previous Scheduling Methods

In order to identify which issues were most prevalent in the average user's experience, with the previous WPI Scheduler, our group sent a survey to the student body at WPI (See Appendix A). Out of the 114 responses, 71 reported troubles saving and later re-opening a schedule, 61 had experienced unexpected crashes, and 48 generated a schedule that had time conflicts. Respondents also overwhelmingly indicated that they expected any scheduling solution to be able to create a schedule without conflicts, and be able to save and later re-open a schedule. Based on these results from the survey, this group identified program crashes, valid schedule generation, and save-load functionality as our three critical areas that must be improved in a new application.

Designing the Solution

Our group spent the first term of our project collecting information and planning the implementation of our program. We drafted user stories (Appendix B) and use cases (Appendix C) to describe the desired behavior of our program. The team then used this behavior to plan the technical structure of our program through Unified Modeling Language (UML) class diagrams (Appendix D). These diagrams depict the logical structure of our program, and how different parts interact. By carefully creating these UML diagrams, the members of the team were able to work independently on different areas of the program with minimal time needed to integrate the different components.

Selecting a Programming Language

After the planning of our application was complete, our group needed to select which language would be used to implement our design. Our application also needed to work regardless of operating

system. This led us to consider Java[7], the language used by the previous WPI Scheduler. Java is perhaps best known for being an object-oriented cross platform language. However, it is also notorious for using a large amount of memory on the user's computer, as well as being computationally slower than other languages.

We then turned our considerations to JavaScript, which although similar in name, has many differences from Java. JavaScript programs are typically accessed by navigating to a webpage, where the client downloads the JavaScript program from the server, and then runs it in the web browser. JavaScript can also be an object-oriented language, and it offered one additional benefit which our team had not previously considered: Program updates and maintenance would be very simple to implement.

A JavaScript program would be re-downloaded every time a user accessed a web page where it was stored. Therefore, by updating the program on the web server, anyone using the program would always have the newest version. However, the major drawback of JavaScript is a difficulty in creating a Graphical User Interface (GUI). A text-based application would be simple to develop, but not user friendly. Conversely, a graphical program would be inherently more user friendly, but difficult to implement in JavaScript. Furthermore, our group was not as experienced in writing JavaScript as we were with Java.

Additional research led us to Google Closure[8], a JavaScript optimizer and library which together provide performance improvements and additional functionality. This tool would not only simplify the creation of a GUI, but also recompile any JavaScript code that we wrote to be as efficient as possible. Unfortunately, using Google Closure meant the application would still be written in JavaScript.

Finally, the team discovered Google Web Toolkit[9] (GWT). Again, a tool created by Google, this

[7] http://www.java.com
[8] https://developers.google.com/closure/
[9] https://developers.google.com/web-toolkit/

toolkit operates as a plugin for Eclipse[10], a popular open-source Java IDE. GWT takes code written in Java, and compiles it into a JavaScript program with identical functionality. This compromise would allow us to comfortably use Java to write our program, but maintain the speed and updatability of JavaScript. GWT also provides libraries to aid in the creation of a GUI, which would remove the need to manually create one in JavaScript.

Creating the New Scheduler

Over two terms, the team worked on implementing our design in Java, with the aid of GWT. The team worked mostly autonomously, and by conforming to our design and using a remote Git[11] repository[12] for version control, we were able to work separately towards the same goal.

By the end of B-Term 2012, our application was functional at a basic level. It was still missing certain crucial functionalities, including saving and loading, as well as restricting schedules by class time. The user interface was also basic, and confusing to use. C-Term 2013 was used to make small but critical improvements to the application. The user interface was slowly improved, and missing functionality was added.

Application Deployment

A copy of the source code, a computer, and the desired deployment webserver are needed to properly deploy the application. Firstly, the source code must be compiled, preferably using the latest version of Google Web Toolkit. For ease of use, using the GWT Eclipse Plugin[13] to compile the code is recommended. As the GWT Eclipse tutorial explains, the web application deployment code can be retrieved from the "WebAppName/war" directory. Simply copy the compiled code, which consists of

[10] http://www.eclipse.org/
[11] http://git-scm.com/
[12] https://github.com/Nican/wpischeduler
[13] http://www.gwtproject.org/usingeclipse.html

several javascript files, to the desired web server location, just like you would host any standard HTML document. You should now be able to reach the WPI Scheduler web application by browsing to the web server location. For the application to produce proper schedules, an up-to-date copy of the WPI schedule data must be provided. Included with our report is a standalone java JAR file, xmltojson.jar which was created to restructure the data provided by the Banner Team at WPI. It parses the XML data, and outputs a single JSON file containing all the same information. The source code of this program and a readme are also included. JavaScript works natively with JSON data, so by transforming from XML to JSON, the load time of the application is significantly reduced. This JSON file must be placed in the same directory where the compiled JavaScript application was placed. It is recommended that this be set up to run as an automatic job, as Banner will provide an updated file fairly frequently. The full data flow for loading class information starts from the WPI Banner Database, which generates an XML output file, which is used as input to the xmltojson utility, which outputs a schedb.json, a file properly formatted to work with our application. This process is visualized in Appendix E.

Saving and Loading is handled by the client automatically using Local Storage, a feature introduced with HTML 5, and as such, no server interaction is required for saving and loading.For detailed instructions on application usage and deployment, please see the included Appendicies G and H.

III. Results

During D-Term of the 2012-2013 academic year, we ran a closed beta testing demo of our application by circulating it to WPI students who willingly volunteered to use it instead of the previous WPI scheduler. Many students ended up sharing the location with their friends, which resulted in hundreds of unique visitors within the two weeks before course registration. Overall, students enjoyed using the program over the previous iteration due to its quick accessibility and ease of use. Since then, we have received inquiries from several students asking where the final version of our application will be hosted.

IV. Conclusions

Recognition

Towards the end of C-Term 2013, our group was approached by members of WPI IT. They expressed interest in adopting our application and providing formal support. In the beginning of the following academic year, we were contacted by the Dean of Undergraduate Studies, Arthur Heinricher, who similarly expressed interest in our application. We met with the Dean and representatives from the IT department on September 3, 2013 to discuss our project and the requirements for continuing to support it. At the end of our project, WPI was still reviewing their options and deciding the best way to move forward. Our group would much prefer that WPI take over support of our program so we are not indefinitely responsible for its maintenance, and it can be properly supported after we graduate from WPI.

Future Work

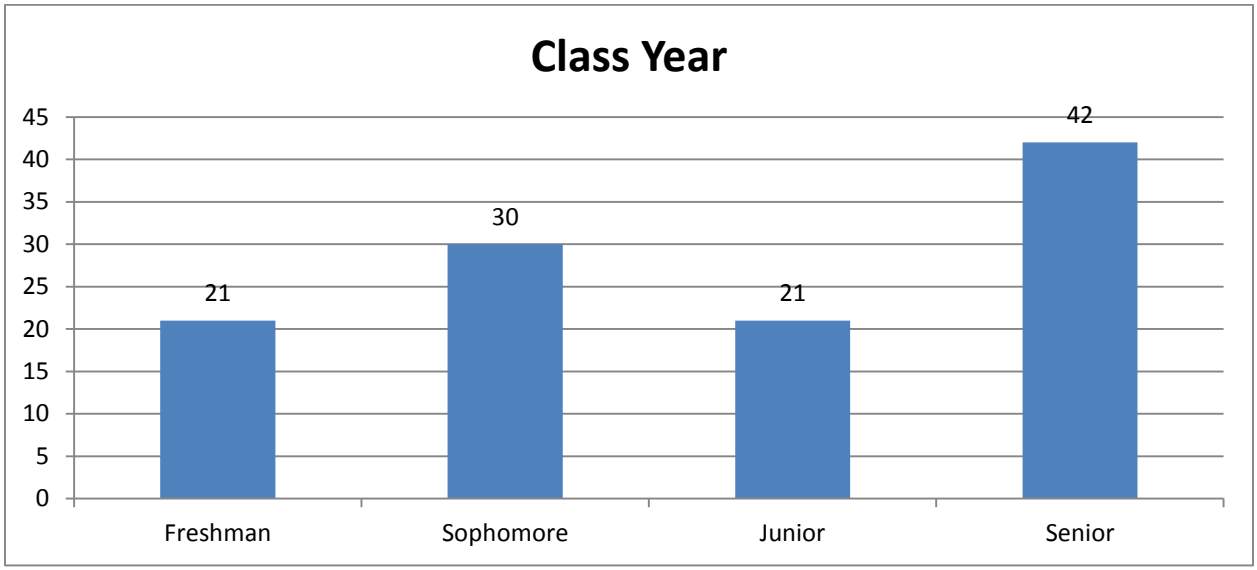
Overall, our group was pleased with the state of our application at the end of the project. We worked with John Plunkett, part of WPI's Banner team, to update the XML file that they generate. The new version includes additional information for each class. Ideally, this new version should be generated on a regular basis and made available for our application to use.

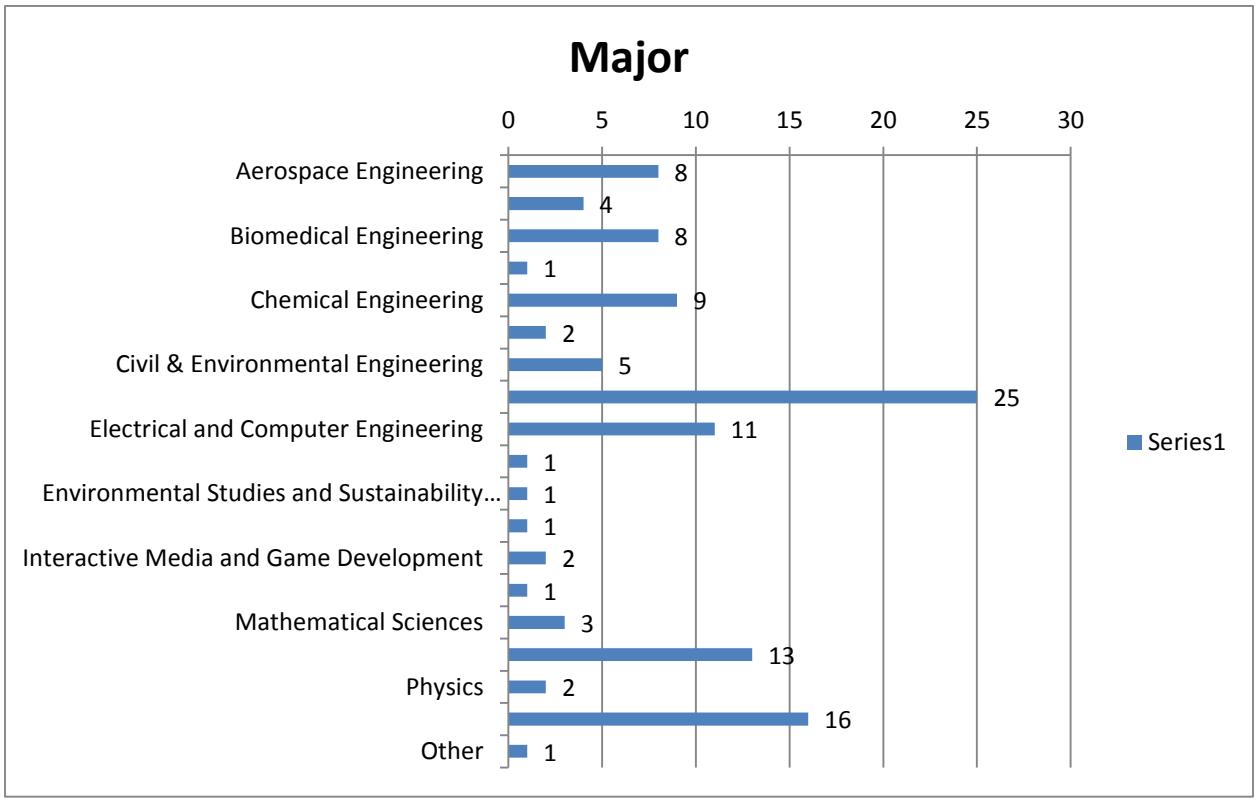
Furthermore, if WPI does adopt our application, we recommend extending it so it can connect directly to the Banner database and retrieve class information in real-time. This would prevent many of the issues that arise due to the current information being up to four hours out of date. Furthermore, it would remove the need to run the xmltojson utility. Direct database access would impose an extra load on the database servers, which would have to be carefully tested to make sure the infrastructure can handle the additional requests.

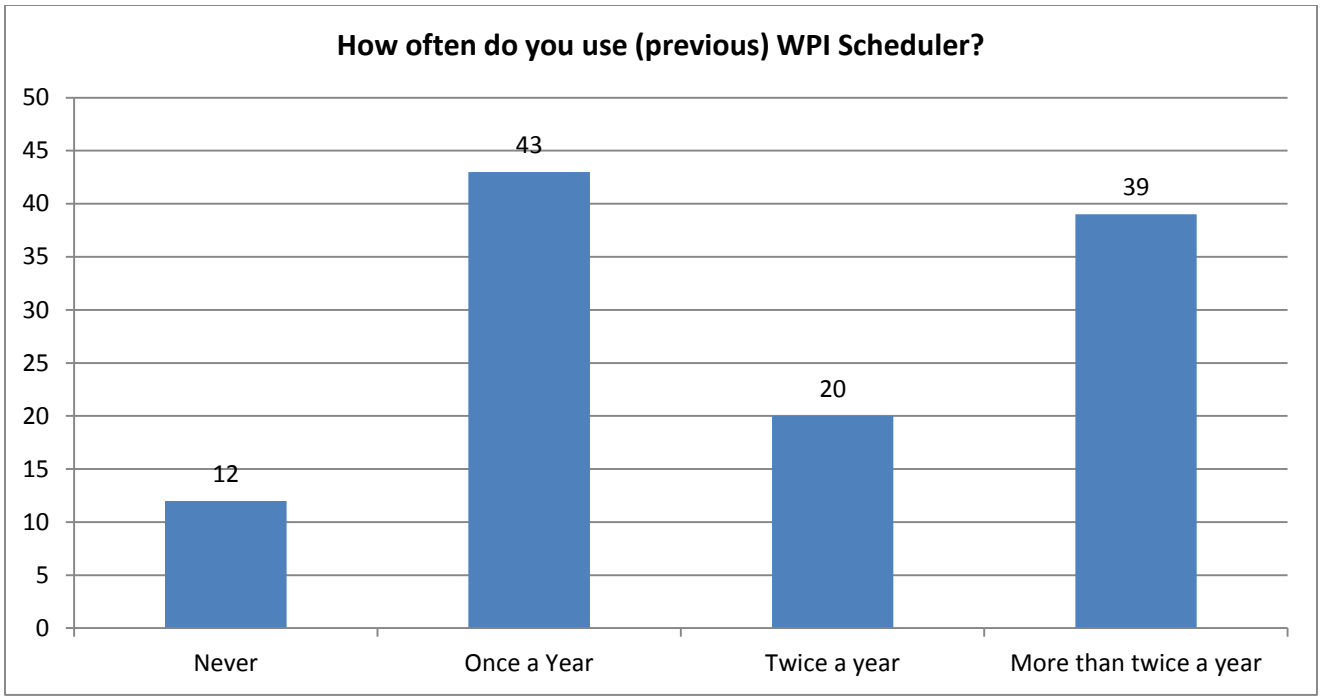
Finally, it may be worth investigating other means of saving and sharing schedules. Our group initially planned on adding a mechanism to allow users to share their created schedules on Facebook, or through some other social media, so their friends could easily see what classes they had in common. Due to time constraints, this was not possible during this project.

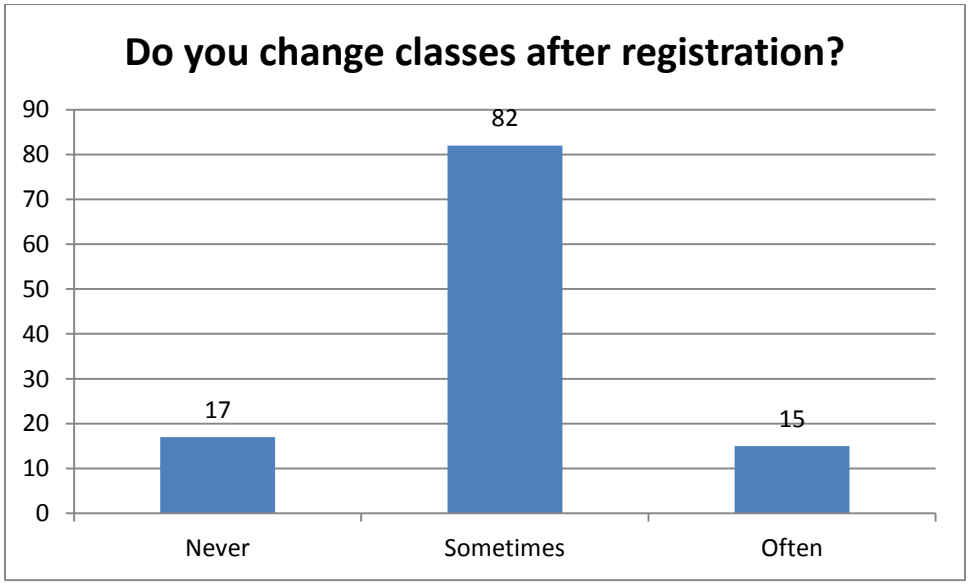
V. Appendices

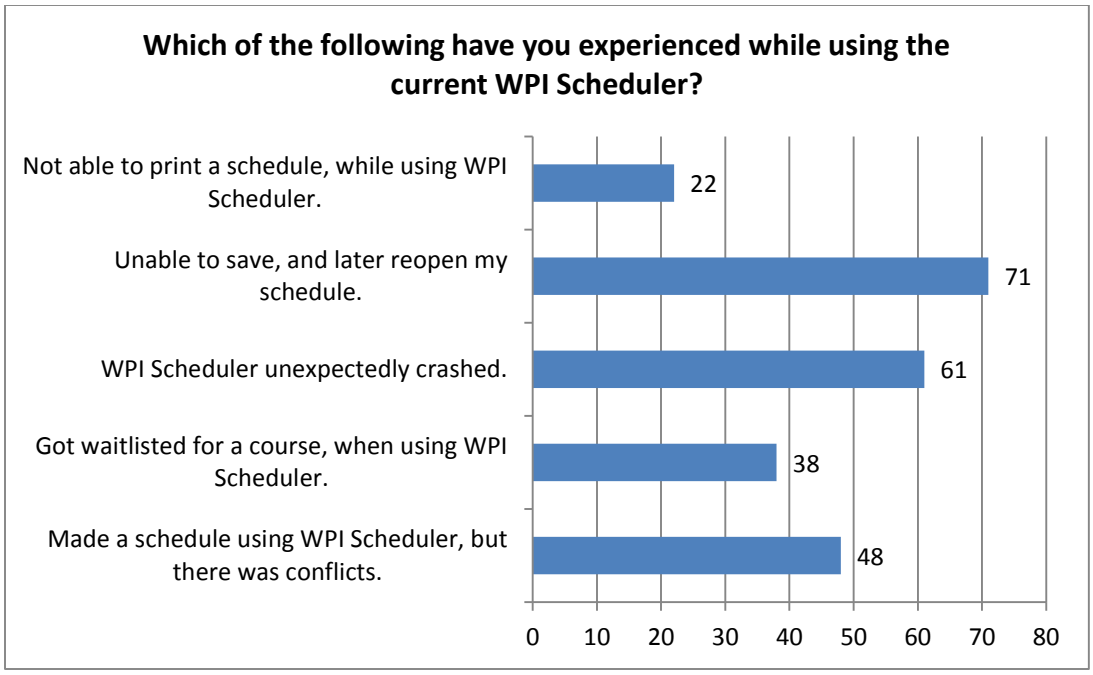
Appendix A – Survey Results

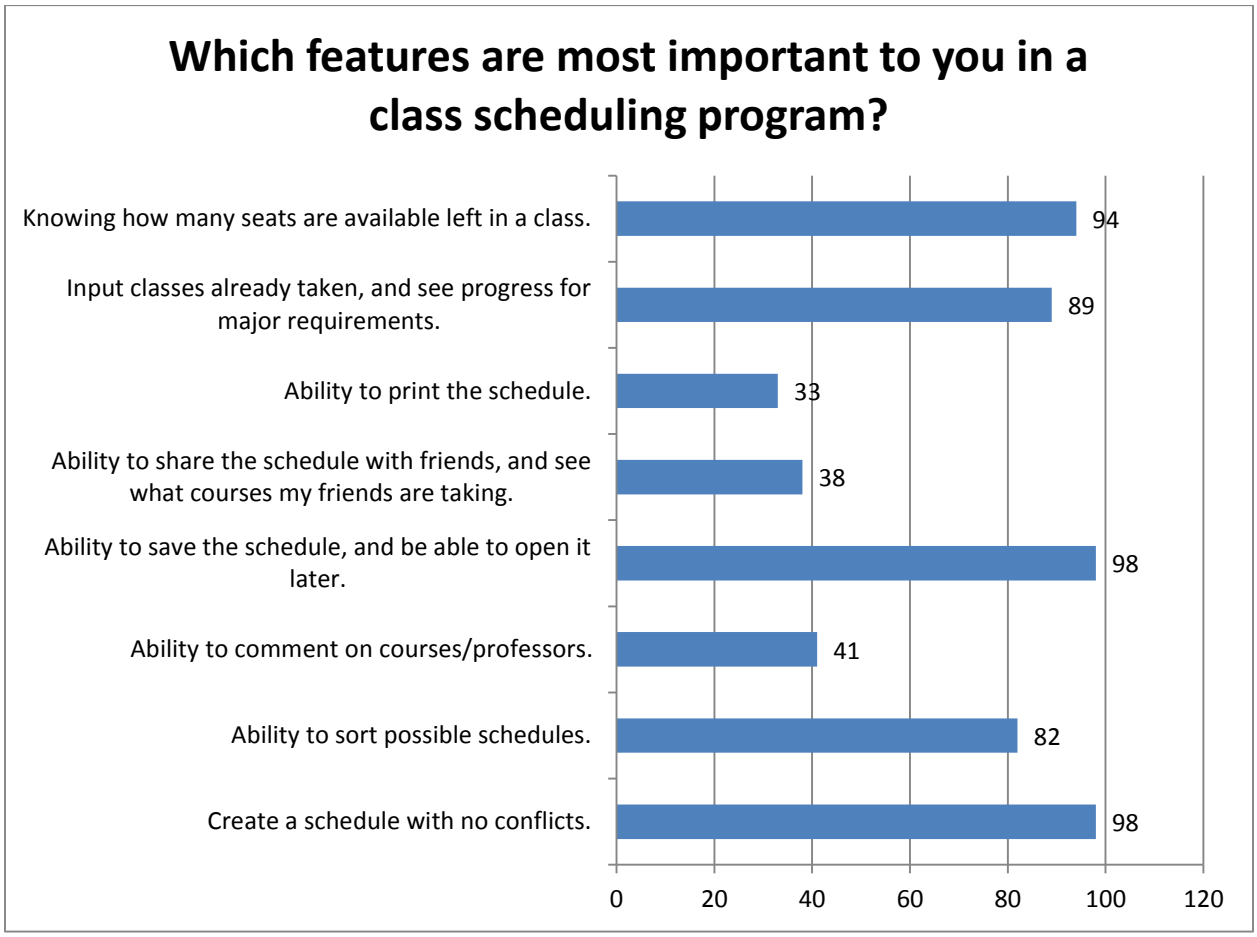





How often do you use (previous) WPI Scheduler?
50
45
40
35
30
25
20
15
10
5
0
12
43
20
39
Never
Once a Year
Twice a year
More than twice a year


Do you change classes after registration?
90
80
70
60
50
40
30
20
10
0
17
82
15
Never
Sometimes
Often


Which of the following have you experienced while using the current WPI Scheduler?
Not able to print a schedule, while using WPI Scheduler.
22
Unable to save, and later reopen my schedule.
71
WPI Scheduler unexpectedly crashed.
61
Got waitlisted for a course, when using WPI Scheduler.
38
Made a schedule using WPI Scheduler, but there was conflicts.
48
0 10 20 30 40 50 60 70 80


Which features are most important to you in a class scheduling program?
Knowing how many seats are available left in a class.
94
Input classes already taken, and see progress for major requirements.
89
Ability to print the schedule.
33
Ability to share the schedule with friends, and see what courses my friends are taking.
38
Ability to save the schedule, and be able to open it later.
98
Ability to comment on courses/professors.
41
Ability to sort possible schedules.
82
Create a schedule with no conflicts.
98
0 20 40 60 80 100 120

Appendix B – User Stories

Title: Base Case
User: Wants to create a schedule for the upcoming year
Story: The user opens WPI Scheduler, chooses the classes he wants to take, and have a list of CRNs ready to be put into WPI Scheduler.

Title: Updating schedule
User: User has an existing schedule, and wants to update it
Story: The user has an existing schedule inside of the WPI scheduler; The user loads the previously made schedule, add/drop classes as needed, and generates a new list of CRN numbers.

Title: Finding now non-existent classes
User: User loads the schedule, and is notified that some previously chosen classes no longer exists.
Story: User has the choice to see a list of a list of the classes that no longer exist, remove them, and choose new classes.

Title: Discovering newly added classes/sections
User: New classes are added into bannerweb, in the student's interested major, and the user is notified.
Story: New classes/sections are added to bannerweb, and the students are notified that the new classes exist.

Title: Saving/Loading a schedule
User: User is working on a schedule, and he wants to save it for editing it later.
Story: User saves the schedule through the WPI Scheduler, and he is able to edit it later. The saved schedule should have the classes picked and the year.

Title: Sharing the schedule with adviser
User: The user has create a new schedule, and now he wants to discuss it with his adviser.
Story: All students have meeting with their advisers to discuss about their schedule. The user has the option to share the schedule, and the adviser is able to leave comments on the scheduler.

Title: Sharing the schedule with friends
User: User has chosen his classes, and now wants to share the schedule with his friends
Story: The user has the option to share the schedule with social networks, such as facebook, google+, etc..., so other friends can see it.

Title: Finding full/waitlisted classes
User: User logins into the scheduler, and see that some of the previously chosen classes are now full.
Story: The user login into scheduler, and finds that some of the classes are full, or are going to be waitlisted, and has the option to choose other alternative classes.

Appendix C – Use Cases

Name: Search Class

- Participating Actor: Initiated by Student
- Entry Condition
    - Scheduler has been started, and the student has started a brand new schedule
- Exit Criteria
    - Scheduler displays a list of classes that meet the search criteria
- Flow of Events
    1. The student enters information they would like to search by
    2. Scheduler displays the classes that fit the information the student entered

Name: Add Class

- Participating Actor: Initiated by Student
- Entry Condition
    - Scheduler has been started, and the student has searched for a certain class they would like to add
- Exit Criteria
    - Scheduler adds the class to the list of classes the student would like to take this year
- Flow of Events
    1. The student selects a class they would like to add
    2. Scheduler adds that class to the separate list of chosen classes

Name: Choose Terms

- Participating Actor: Initiated by Student
- Entry Condition
    - Scheduler has been started, and the student has chosen the classes they wish to take
- Exit Criteria
    - Scheduler updates the terms the student would like to take that specific class
- Flow of Events
    1. The student requests to take a certain class during a subset of the available terms
    2. Scheduler reflects that term selection for the class requested

Name: Choose Times

- Participating Actor: Initiated by Student
- Entry Condition
    - Scheduler has been started, and the student has chosen the classes they wish to take
- Exit Criteria
    - Scheduler updates the times classes are allowed to occur in order to be considered
- Flow of Events
    1. The student requests to take classes in a certain set of timeframes
    2. Scheduler updates the time frame in which selected classes are allowed to occur

Name: Choose Section

- Participating Actor: Initiated by Student
- Entry Condition
    - Scheduler has been started, and the Student has chosen the classes they wish to take
- Exit Criteria
    - Scheduler selects/deselects that specific section, affecting if it will be a part of the final schedule generation process
- Flow of Events
    1. The student requests a specific section
    2. Scheduler shows the student the specific time details of that section
    3. The student selects/deselects that section
    4. Scheduler removes/adds that section to the list of sections to be included in the final schedule generation process

Name: Generate Schedules

- Participating Actor: Initiated by Student
- Entry Condition
    - Scheduler has been started, and the Student has chosen sections that they would like to be a part of their final schedule choices
- Exit Criteria
    - Scheduler displays every possible schedule that qualifies as a “necessary and sufficient” WPI schedule
- Flow of Events
    1. The student requests for schedules to be generated over the sections chosen
    2. Scheduler generates every possible schedule to the Student to choose from

Name: Sort Schedules

- Participating Actor: Initiated by Student
- Entry Condition
    - Scheduler has been started, and the possible schedules have been generated
- Exit Criteria
    - Scheduler displays the schedules in a certain order that follows the specified sort criteria
- Flow of Events
    1. The student requests for schedules to be sorted by a certain criteria
    2. Scheduler sorts the current schedules by the given criteria

Name: Finalize Schedule

- Participating Actor: Initiated by Student
- Entry Condition
    - Scheduler has been started, and the Student has generated all possible schedules
- Exit Criteria
    - Scheduler confirms the final schedule and provides several end options to the student
- Flow of Events
    1. The student chooses a final schedule out of the list of possible schedules
    2. Scheduler displays the final schedule to confirm their choice is correct, with final options to be used upon the final schedule

Name: Print Schedule

- Participating Actor: Initiated by Student
- Entry Condition
    - Scheduler has been started, and the Student has chosen their final schedule
- Exit Criteria
    - Scheduler gives the student several printer friendly visual formats
- Flow of Events
    1. The student requests for their final schedule to be printed
    2. Scheduler generates visual formats that can be printed using their web-browser

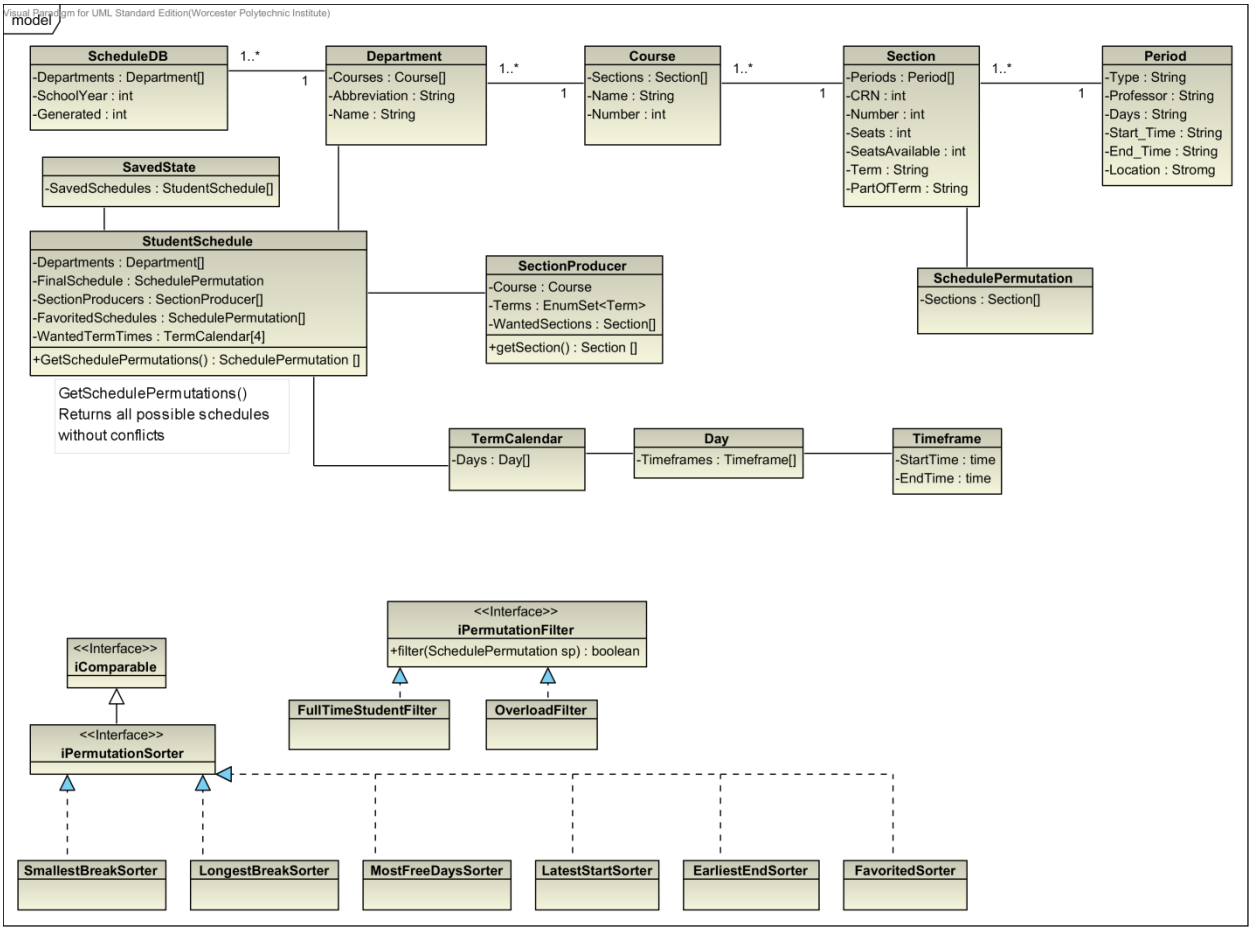
Name: Save Schedule

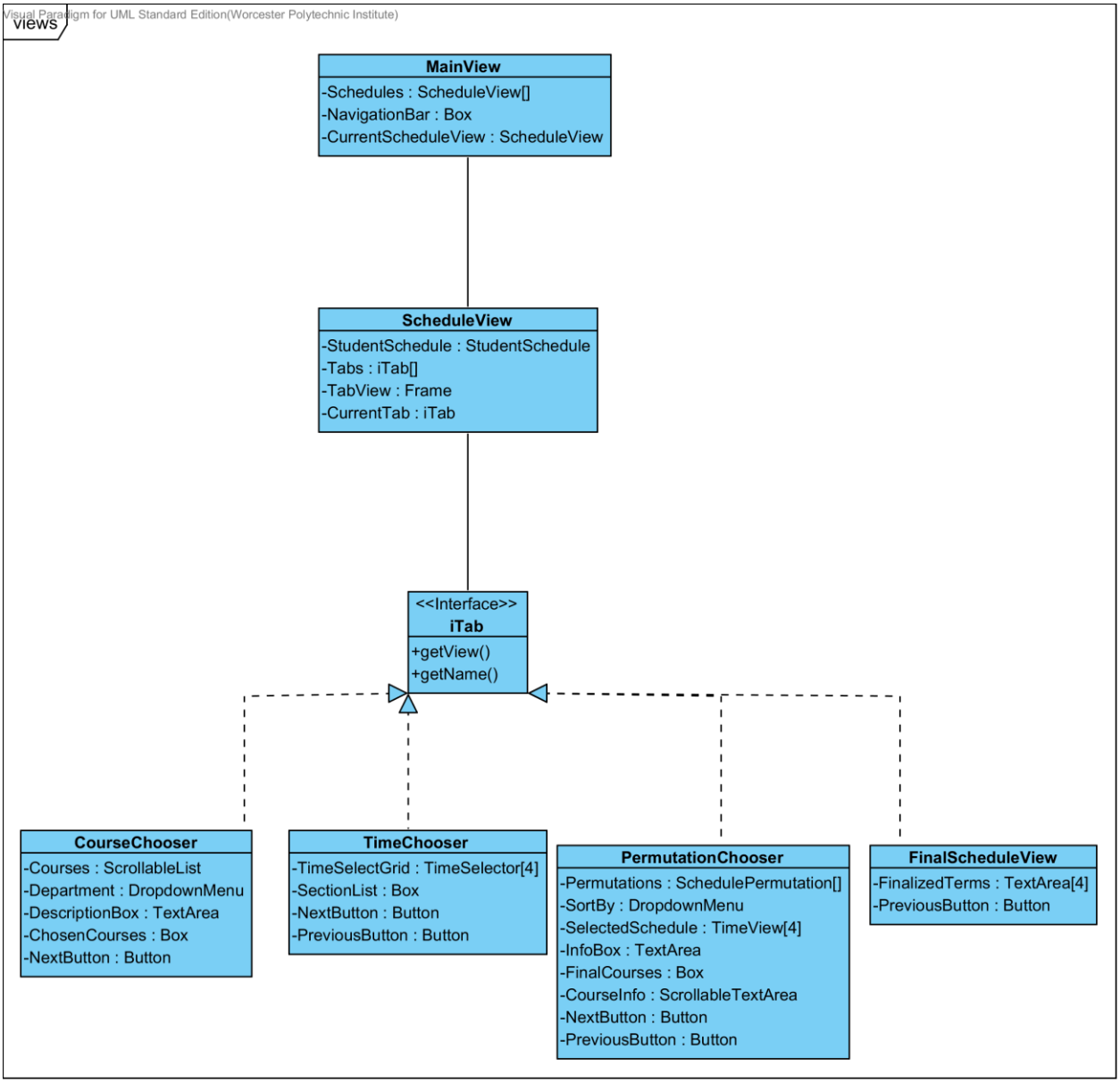
- Participating Actor: Initiated by Student
- Entry Condition
    - Scheduler has been started, and the Student has chosen their final schedule
- Exit Criteria
    - Scheduler saves the schedule so it can be opened at a later date
- Flow of Events
    1. The student requests for the final schedule to be saved
    2. Scheduler notifies the user that the schedule has been saved and how to open it later

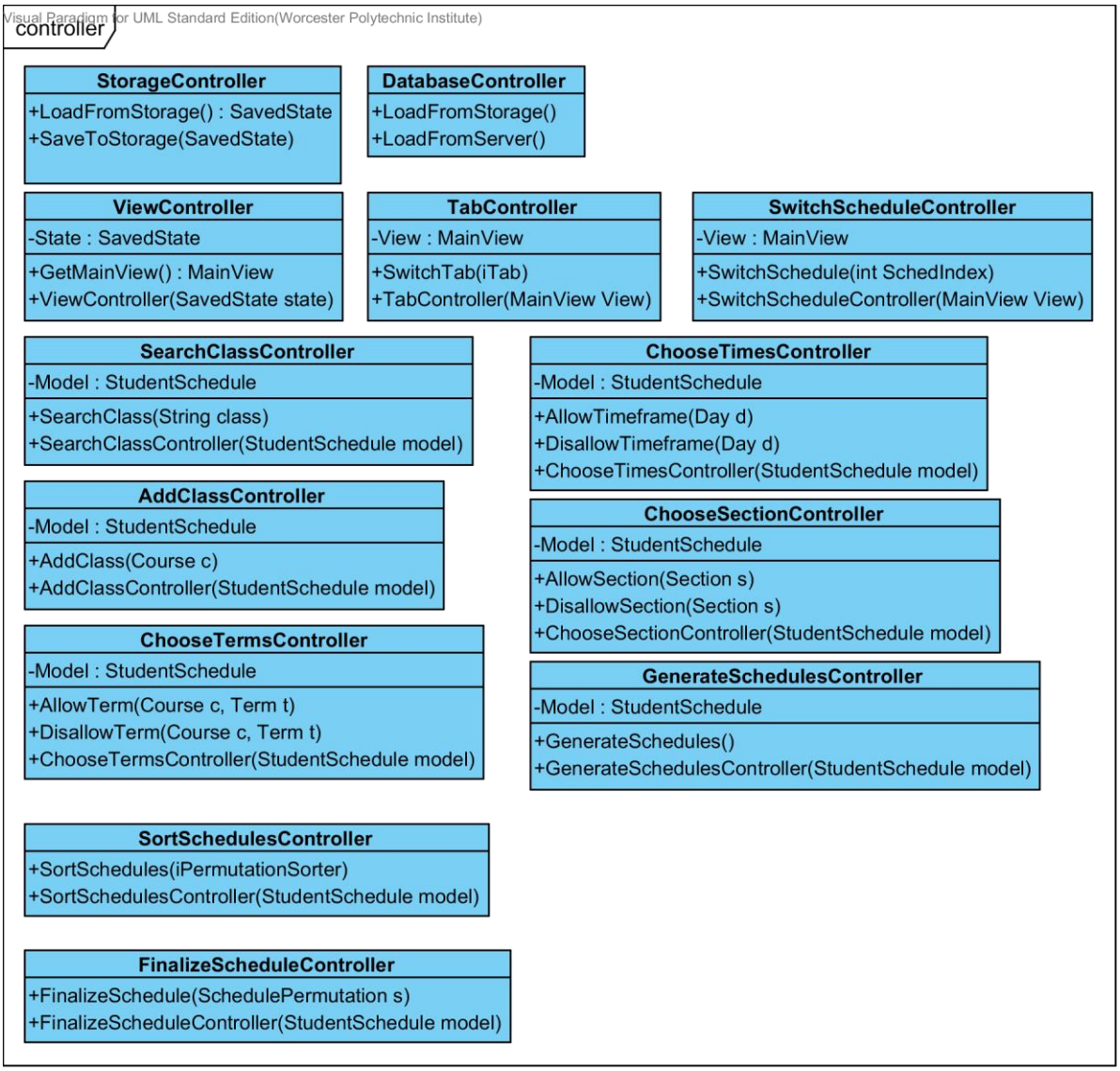
Name: Load Schedule

- Participating Actor: Initiated by Student
- Entry Condition
    - Scheduler has been started
- Exit Criteria
    - Scheduler opens the schedule given
- Flow of Events
    1. The student requests to open the saved schedule
    2. Scheduler opens the schedule and returns the student to where they left off
    3. Scheduler cannot fully restore the schedule given, and returns the student to the farthest possible state of their application usage

Appendix D – UML Class Diagrams




Visual Paradigm for UML Standard Edition(Worcester Polytechnic Institute)
views
MainView
-Schedules : ScheduleView[]
-NavigationBar : Box
-CurrentScheduleView : ScheduleView
ScheduleView
-StudentSchedule : StudentSchedule
-Tabs : iTab[]
-TabView : Frame
-CurrentTab : iTab
<<Interface>>
iTab
+getView()
+getName()
CourseChooser
-Courses : ScrollableList
-Department : DropdownMenu
-DescriptionBox : TextArea
-ChosenCourses : Box
-NextButton : Button
TimeChooser
-TimeSelectGrid : TimeSelector[4]
-SectionList : Box
-NextButton : Button
-PreviousButton : Button
PermutationChooser
-Permutations : SchedulePermutation[]
-SortBy : DropdownMenu
-SelectedSchedule : TimeView[4]
-InfoBox : TextArea
-FinalCourses : Box
-CourseInfo : ScrollableTextArea
-NextButton : Button
-PreviousButton : Button
FinalScheduleView
-FinalizedTerms : TextArea[4]
-PreviousButton : Button


Visual Paradigm for UML Standard Edition(Worcester Polytechnic Institute)
controller
StorageController
+LoadFromStorage() : SavedState
+SaveToStorage(SavedState)
DatabaseController
+LoadFromStorage()
+LoadFromServer()
ViewController
-State : SavedState
+GetMainView() : MainView
+ViewController(SavedState state)
TabController
-View : MainView
+SwitchTab(iTab)
+TabController(MainView View)
SwitchScheduleController
-View : MainView
+SwitchSchedule(int SchedIndex)
+SwitchScheduleController(MainView View)
SearchClassController
-Model : StudentSchedule
+SearchClass(String class)
+SearchClassController(StudentSchedule model)
ChooseTimesController
-Model : StudentSchedule
+AllowTimeframe(Day d)
+DisallowTimeframe(Day d)
+ChooseTimesController(StudentSchedule model)
AddClassController
-Model : StudentSchedule
+AddClass(Course c)
+AddClassController(StudentSchedule model)
ChooseSectionController
-Model : StudentSchedule
+AllowSection(Section s)
+DisallowSection(Section s)
+ChooseSectionController(StudentSchedule model)
ChooseTermsController
-Model : StudentSchedule
+AllowTerm(Course c, Term t)
+DisallowTerm(Course c, Term t)
+ChooseTermsController(StudentSchedule model)
GenerateSchedulesController
-Model : StudentSchedule
+GenerateSchedules()
+GenerateSchedulesController(StudentSchedule model)
SortSchedulesController
+SortSchedules(iPermutationSorter)
+SortSchedulesController(StudentSchedule model)
FinalizeScheduleController
+FinalizeSchedule(SchedulePermutation s)
+FinalizeScheduleController(StudentSchedule model)

Appendix E – Data Flow Diagram

WPI Scheduler Data Flow Diagram

Appendix F – Files Included With Project Submission

1) WPI SCHEDULER IMPROVEMENT PROJECT REPORT.docx – Word format of the PDF report
2) xmltojson.zip – Zip containing compiled JAR as well as source code and README for the xmltojson utility
3) wpischeduler-master.zip – Zip file containing source code for the scheduler as it existed at the conclusion of this project
4) wpi.schedbR – Sample XML file generated by the Banner team at WPI
5) schedb.json – Sample JSON output from the xmltojson utility
6) Appendix G – User Guide.pdf
7) Appendix H – Deployment Guide.pdf