TEAM 2 FURPS++
FINAL SUBMISSION

Austin Cory Bart
Etornam Banini
Michael Chinn
Andrea Macartney
William Greenhalgh

CISC 475 Fall Semester
Professor Andrew Roosen

December 5, 2011

TABLE OF CONTENTS

VISION DOCUMENT

Currently at the University of Delaware there is a strong need for an automated way to assign Teaching Assistants (TA's) and Lab Assistants (LA's) to class sections. The most cumbersome part of this task is to apply numerous constraints to these assignments, where the constraints must be satisfied in order for a viable schedule to be produced. The current scheduling process is done by hand, and because of the overwhelming data involved, this process is both extremely time-consuming and also prone to human error. Having an efficient system which could aid in this process would give the domain expert- Dr. Daniel Chester- more time to spend on other tasks. Further, such a system could have applications in other departments and schools that have similar TA scheduling needs.

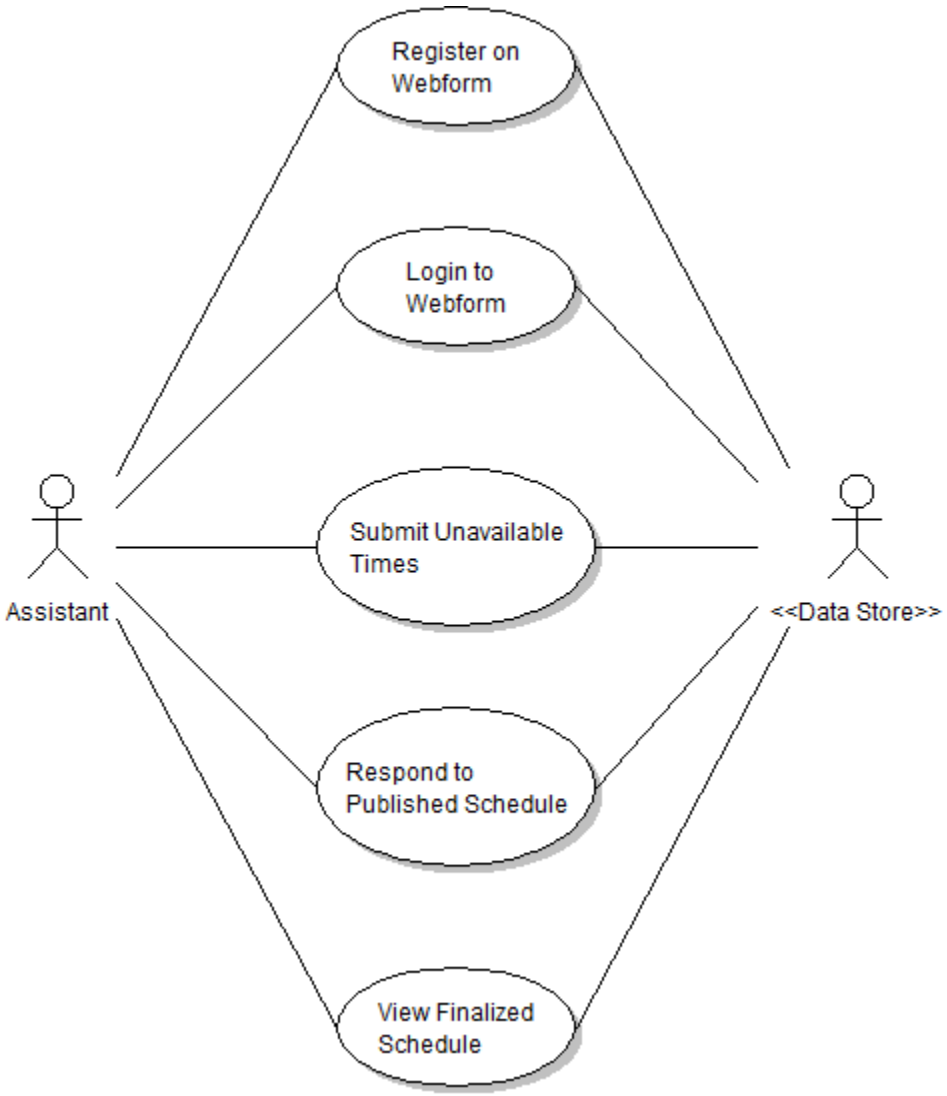
Our solution is a software suite, consisting of two different pieces, our Assistant Scheduling System and Webform. The Assistant Scheduling System is a Java-based application which aids in the scheduling of TA's and LA's. The system will produce schedules by performing constraint satisfaction on a wide variety of constraints, such as balancing the time between engagements, ensuring the TA has proper background knowledge of the course subject, and maintaining a manageable class size. The system will be extensible so that future scheduling constraints can be added without too much difficulty. There is a Graphical User Interface so that the user can easily enter in the required information and view the results of the proposed solution.

The second part of our software suite is a PHP-based Webform with which assistants will interact. The Webform was created because assistants need a way to enter in their information from anywhere, not just on one local computer like the Scheduling System can. It allows for assistants to add and edit their data, such as unavailable times, and then review and respond with either a rejection or acceptance of a particular schedule they were assigned to.

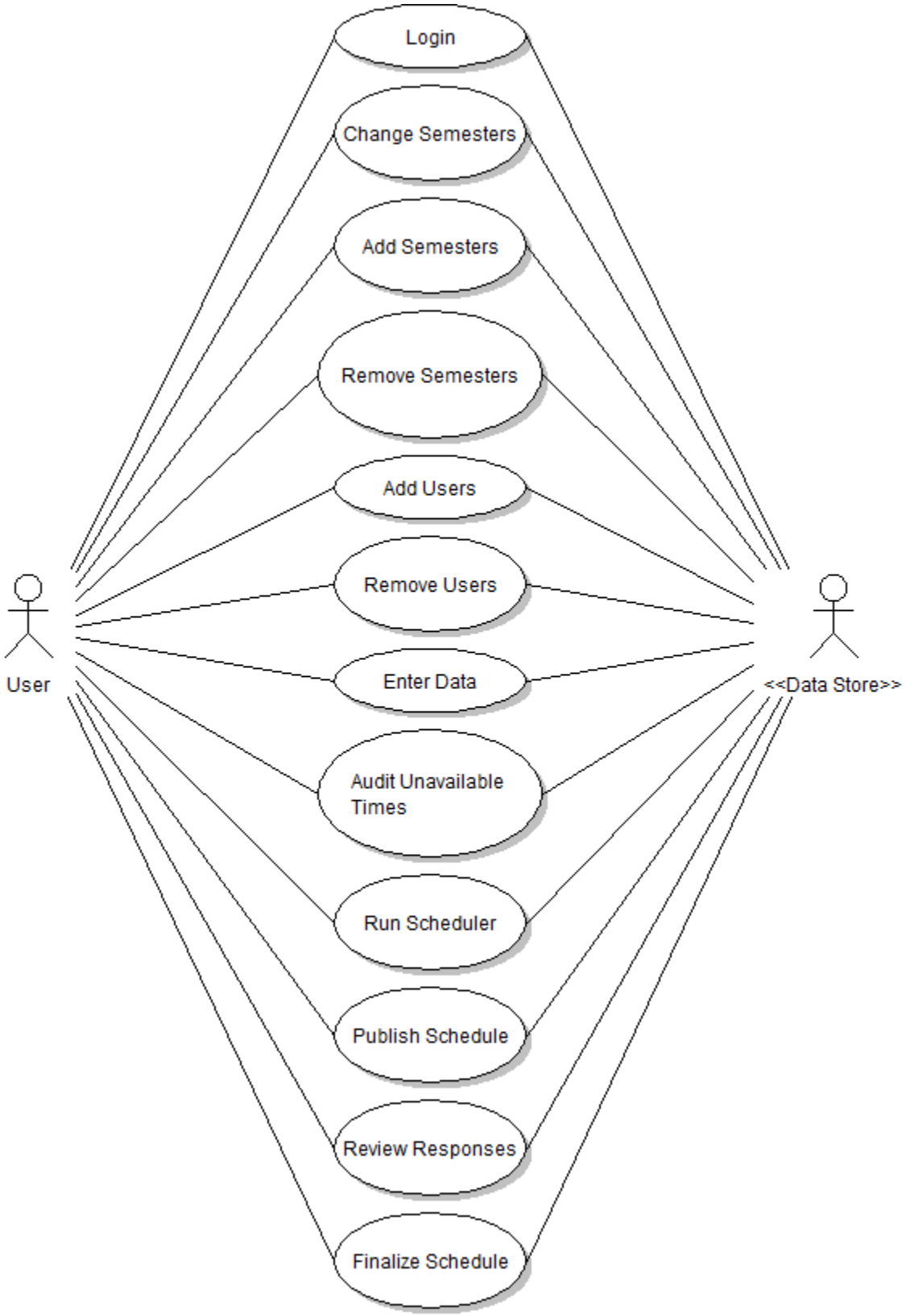
The success of the Assistant Scheduler System will be demonstrated by the production of a valid schedule that abides by the provided constraints. The system needs to give the result in a timely manner. It should report a viable schedule in minutes, not hours. It needs to consume significantly less time and effort than the old by-hand method when being used in subsequent semesters to justify the increased set-up and training time associated with using a new system.

USE CASE MODELS

ASSISTANT TO DATA STORE



USER TO DATA STORE



FORMALIZED USE CASES

(1) LOGIN

Actors:

1. User
2. Data Store

Trigger:

- The user signals to the system the desire to log in

Precondition:

- None

Post-condition:

- The user has successfully logged into the system

Normal Flow:

1. The user enters their username and password
2. The username and password are verified on the server
3. The user is logged into the system

Alternate Flow:

2b: The username is not found or the password is invalid

2b1: The user is not logged into the system

(2) CHANGE SEMESTERS

Actors:

1. User
2. Data Store

Trigger:

- The user signals to the system the desire to change semesters

Precondition:

- The user is logged into the system

Post-condition:

- The current semester has changed

Normal Flow:

1. The system presents all available semesters
2. The user selects a semester

3. The system loads the selected semester

Alternate Flows:
2a: The user wants to duplicate an old semester
2a1: The user selects a semester
2a2: The user enters a name for the duplicate semester
2a3: The system adds the the new semester based on the old one
2a4: The user selects a semester
2a5: The system loads the selected semester

(3) Add Semesters

Actors:
1. User
2. Data Store

Trigger:
- The user signals to the system the desire to add semesters

Precondition:
- The user is logged into the system

Post-condition:
- The current semesters in the system have been changed

Normal Flow:
1. The system presents all available semesters
2. The user enters a name for the semester
3. The system adds the new semester

(4) Remove Semesters

Actors:
1. User
2. Data Store

Trigger:
- The user signals to the system the desire to remove semesters

Precondition:
- The user is logged into the system

Post-condition:

- The current semesters in the system have been changed

Normal Flow:

1. The system presents all available semesters
2. The user selects a semester
3. The system removes the semester

(5) ADD USERS

Actors:

1. User
2. Data Store

Trigger:

- The user signals to the system the desire to add users

Precondition:

- The user is logged into the system

Post-condition:

- The current users in the system have been changed

Normal Flow:

1. The system presents all existing users besides the logged-in user
2. The user enters information about a new user
3. The system adds that user from the data store

(6) REMOVE USERS

Actors:

1. User
2. Data Store

Trigger:

- The user signals to the system the desire to remove users

Precondition:

- The user is logged into the system

Post-condition:

- The current users in the system have been changed

Normal Flow:

1. The system presents all existing users besides the logged-in user
2. The user selects a user to remove
3. The system removes that user from the data store

(7) ENTER DATA

Actors:

1. User
2. Data Store

Trigger:

- The user signals to the system the desire to enter data

Preconditions:

1. The user is logged into the system
2. The user has chosen a semester

Post-condition:

- The model in the data store has changed

Normal Flow:

1. The system presents the current semester's model data (assistants, sections, constraints, instructors, schedules, and skill sets) from the data store
2. The user manipulates the data present
3. The system stores the changes to the data store

Alternate Flow:

2a: The user enters incorrect data

2a1: The system rejects the data

(8) AUDIT UNAVAILABLE TIMES

Actors:

1. User
2. Data Store
3. Assistant

Trigger:

- The user signals to the system the desire to audit assistant's unavailable times

Preconditions:

1. The user is logged into the system
2. The user has chosen a semester

3. Assistants have submitted unavailable times associated with the semester to the data store

Post-condition:

- There are no more unavailable times associated with the semester in the data store

Normal Flow:

1. The system presents a list of assistants who have unavailable times
2. The user chooses an assistant
3. The user reviews all the unavailable times associated that assistant
4. All acceptable unavailable times are marked as such
5. All marked unavailable times are added to the data in the data store
6. Unmarked times are removed from the system

(9) RUN SCHEDULER

Actors:

1. User
2. Data Store

Trigger:

- The user signals to the system the desire to obtain a schedule

Preconditions:

1. The user is logged into the system
2. The user has chosen a semester
3. The user has entered any data they want to base the schedule on

Post-condition:

- A new schedule is added to the data store

Normal Flow:

1. The system performs constraint satisfaction to find a valid schedule
2. The user reviews the generated schedule and makes individual modifications as they desire
3. The schedule is marked as “unpublished” and added to the data store

Alternate Flow:

2a: The system cannot find a valid schedule

2a1: The system notifies the user that it is unable to find a valid schedule

2a2: The system suggests to the user what data and constraints are preventing a valid schedule

(10) PUBLISH SCHEDULE

Actors:

1. User
2. Data Store
3. Assistant

Trigger:

- The user signals to the system the desire to publish a schedule

Preconditions:

1. The user is logged into the system
2. The user has chosen a semester
3. The user has run the scheduler to create a schedule

Post-condition:

- A schedule is marked as published on the data store

Normal Flow:

1. The system presents the user with a list of schedules
2. The user chooses a schedule to publish
3. The schedule is marked as published on the data store
4. Any assistants who are affected by the published schedule are notified via email

(11) REVIEW RESPONSES

Actors:

1. User
2. Data Store
3. Assistant

Trigger:

- The user signals to the system the desire to review assistant responses

Preconditions:

1. The user is logged into the system
2. The user has chosen a semester
3. The user has published a schedule
4. Assistants have submitted responses to the published schedule

Post-conditions:

1. Assistant responses are removed from the data store
2. Assignments in the schedule are marked as accepted, rejected, or overridden

Normal Flow:

1. The system presents a list of assistant responses to the user
2. If there are no rejections, the reviews are removed and the schedule is marked as fully accepted

Alternate Flows:

2a: A rejection is present for an assignment and the user would like to modify the schedule

2a1: The user modifies the schedule and has it republished

2a2: Any reviews that included rejections are purged

2b: A rejection is present for an assignment and the user would like to override this rejection

2b1: The user overrides the rejection

2b2: If all rejections are accepted or overridden, the reviews are removed and the schedule is marked as fully accepted

(12) FINALIZE SCHEDULE

Actors:

1. User
2. Data Store

Trigger:

- The user signals to the system the desire to obtain a schedule

Preconditions:

1. The user is logged into the system
2. The user has chosen a semester
3. The user has published a schedule
4. The schedule has been marked as fully accepted

Post-condition:

- The schedule is marked as finalized on the data store

Normal Flow:

1. The system presents a fully accepted schedule to the user
2. The user approves it
3. The system marks the schedule as finalized on the data store
4. All assistants are notified of the finalized schedule via email

(13) REGISTER ON WEBFORM

Actors:

1. Assistant

2. Data Store

Trigger:
- The assistant signals the data store that they wish to register

Precondition:
- None

Post-condition:
- The assistant is registered for the webform

Normal Flow:
1. The webform requests the udel username and a non-udel password from the assistant
2. The assistant provides their udel username and a non-udel password
3. The assistant is sent an email asking them to verify their account
4. The user verifies their account
5. The webform registers the assistant in the data store

(14) LOGIN TO WEBFORM
Actors:
1. Assistant
2. Data Store

Trigger:
- An assistant signals the webform that they desire to login to the system

Precondition:
- The assistant is registered in the system

Post-condition:
- The assistant is logged into the webform

Normal Flow:
1. The assistant provides their login credentials to the webform
2. The webform accepts the login credentials
3. The assistant is logged into the webform

(15) SUBMIT UNAVAILABLE TIMES
Actors:
1. Assistant
2. Data Store

Trigger:

- An assistant signals the webform that they desire to submit unavailable times

Precondition:

- The assistant is logged into the system

Post-condition:

- Unavailable times are added to the current semester for the assistant on the data store

Normal Flow:

1. The webform provides a blank list of unavailable times
2. The assistant fills out the list with times they are unavailable and the reason they are unavailable at that time
3. The assistant submits the information and the webform queues this information in the data store

(16) Respond to Published Schedule

Actors:

1. Assistant
2. Data Store
3. User

Trigger:

- An assistant signals the webform that they desire to review their assignments

Preconditions:

1. The assistant is logged into the system
2. A user has published a schedule

Post-condition:

- Responses are added to the current semester for the assistant on the data store

Normal Flow:

1. The webform presents a list of assignments to the assistant
2. The assistant accepts the assignment
3. The acceptance is queued on the data store

Alternate Flow:

2a: The assistant rejects the assignment

2a1: The assistant provides a reason for this rejection on the webform

2a2: The rejection is queued on the data store

(17) VIEW FINALIZED SCHEDULE

Actors:

1. Assistant
2. Data Store
3. User

Trigger:

- An assistant signals the webform that they desire to see their finalized assignments

Precondition:

- The assistant is logged into the webform

Post-conditions:

- None

Normal Flow:

1. The webform presents the assistant with a list of their assignments

SUPPLEMENTARY REQUIREMENTS

Operational Concerns:

1. The server system needs to be running at all times that a client system could be running. The reverse is not necessary.
2. All assistants must be able to access and modify the webform.

Domain Rules of the Constraint Satisfaction System:

1. The input of the system is the following
    a. Let *assistants* be the set of assistants
    b. Let *sections* be the set of sections
    c. Let *positions* be the set of positions
    d. Let *instructors* be the set of instructors
    e. Let *skills* be the set of all possible skills
    f. Let *times* be the set of all possible times
2. The output of the system is the set of binary predicates *assigned(x,y)* such that *a* is an assistant assigned to a position *p*.
3. Constraints should be addable via source code modification. The following initial, minimal constraints have been identified for the system by Dr. Chester:
    a. All sections must be filled.

```
for all p in positions:
    exists an a in assistants:
        assigned(p, a)
```

    b. Some assignments are specifically forbidden for a section:

```
for all s in i.SECTIONS_TAUGHT:
    for all p in s.OPEN_POSITIONS:
        for all a in assistants:
            if a in s.FORBIDDEN_ASSISTANTS:
                ¬assigned(p,a)
```

    c. Some assignments are specifically required for a position by a section:

```
for all s in i.SECTIONS_TAUGHT:
    for all p in s.OPEN_POSITIONS:
        for all a in assistants:
            if a in s.REQUIRED_ASSISTANTS:
                assigned(p,a)
```

    d. Some professors specifically forbid a student from working with them:

```
for all i in instructors:
    for all s in i.SECTIONS_TAUGHT:
        for all p in s.OPEN_POSITIONS:
            for all a in assistants:
                if a in i.FORBIDDEN_ASSISTANTS:
                    ¬assigned(p,a)
```

    e. An Assistant cannot be assigned a position that would exceed their student limit

```
for all a in assistants:
```

```
        exists p in positions:
            a.STUDENT_LIMIT ≤ (sum({b|for all b in positions,
    assigned(a,b)}) + p.STUDENT_LIMIT) ∧ ¬(assigned(p,a)
```

f. An Assistant cannot be assigned to a position that meets during one of their unavailable times:

```
for all p in positions:
    for all a in assistants:
        if s.TIME ∈ a.UNAVAILABLE_TIMES:
            ¬assigned(p,a)
```

g. Two sections cannot be assigned to the same assistant if their times intersect

```
for all p1 in positions:
    for all p2 in positions:
        if p1 ≠ p2 ∧ intersects(p1.TIME, p2.TIME):
            assigned(p1,x) ≠ assigned(p2,x)
```

h. Require Skills: Forbid an assignment if the assistant doesn't have all the skills required for it

```
for all p in positions:
    for all a in assistants:
        if p.SKILLS ⊈ a.SKILLS:
            ¬assigned(p,a)
```

Packaging

- The client system should be installable with no hassle. The server system's requirements are less stringent, but should still be clear and direct.

Security

1. Any private assistant data should be stored encrypted in the Data Store, including
    - Skills
    - Unavailable Times
2. Any private instructor data should be stored encrypted in the Data Store including
    - Forbidden Assistants
3. Any private section data should be stored encrypted in the Data Store including
    - Required Assistants
    - Forbidden Assistants
4. Users must login with a username and password
5. Assistants must login with a username and password
6. All passwords must be stored encrypted.

GLOSSARY

Assignment: a single matching/relation of an Assistant to an Position.

Assistant: Student who will be assisting the professor in a Section of a Course. This could either be a Teaching Assistant (TA) or a Lab Assistant (LA). Their information includes their Skill Set, Time Blocks in which they are not available, their student ID number, and username (from their UD email address).

Assistant Scheduling System (aka Scheduler): The system that uses Assistant, Section, and Instructor data to generate a satisfactory schedule.

Constraint: Predicates/conditions which are applied to the relations (Assignments) of Assistants to Sections before the output of a Schedule can be produced.

Course: a class at UD, e.g. "CISC-106", which may consist of multiple Sections.

Data Store: The location that sessions, login information, and webform data is stored in.

Finalized Schedule: a schedule that has been sent to the webform for Assistants to view. Assistants are notified and only allowed to look at the schedule.

Hard Constraint: A constraint which is absolute, and cannot be compromised.

Instructor: teaches a particular Course. Can forbid an Assistant to assist any of their Courses. Can also request an Assistant to fill a specific Position.

Position: A position that needs to be filled by an Assistant for a section, e.g. "LA Position #1 of CISC-106-030L" and "LA Position #2 of CISC-106-030L". A section may have multiple Positions associated with it.

Published Schedule: a schedule that has been sent to the webform for Assistants to review. Assistants are notified if they are affected by the new Schedule, and must submit an acceptance or rejection of their assignments via the webform.

Response: An acceptance or rejection by an Assistant of a published schedule. These responses are queued in the data store, until they are reviewed by a User. If an Assistant rejects an assignment, they must provide a reason for doing so. If all responses report acceptable, then the schedule may be finalized.

Schedule: A set of assignments between Assistants and Positions. A schedule must be satisfactory, that is, it conforms to all Constraints specified. After a schedule has been produced, it can be published and ultimately finalized.

Section: A particular meeting of a course, e.g. “CISC-106-030L” or “CISC-106-031L” Information includes the location (building), enrollment limit, current enrollment, Time Block, Course in which the Section is associated with, a set of skills required to assist it, and one or more Positions.

Semester: A saved snapshot of the model at a particular time that correlates to a semester that a schedule is needed for.

Session: a snapshot of the model, saved to and loaded from the Data Store between runs of the program.

Skill Set: The skills which an Assistant has, or that a section requires. These are represented as a set of “tags”, keywords that include items like “software engineering”, “english proficiency”, or “graduate level”.

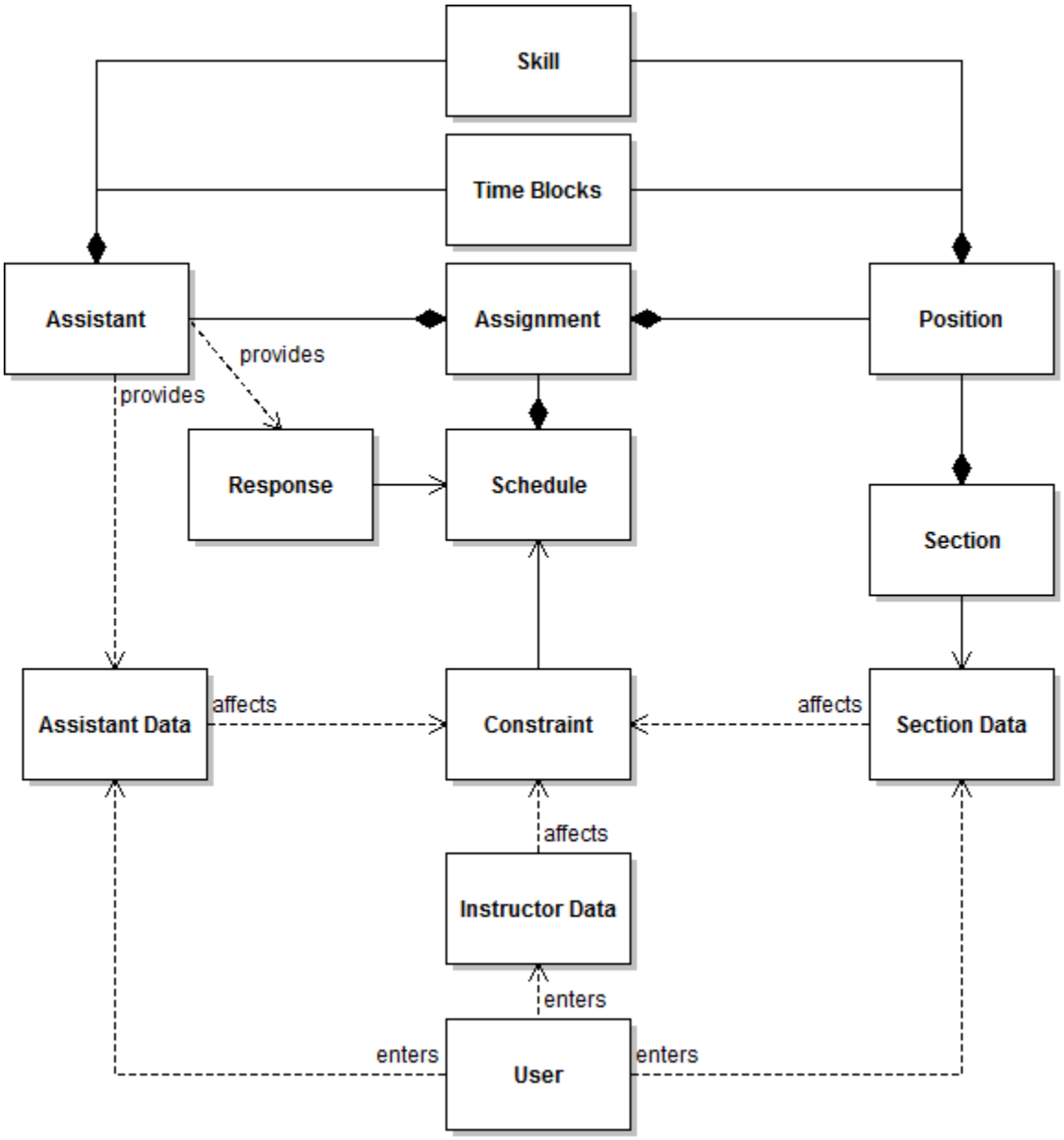
Soft Constraint: A constraint which is a preference. This would be taken into consideration when calculating an optimal schedule, but is not an absolute requirement. Our system does not implement soft constraints but is built such that they may be added in the future.

Time Blocks: A range of time (having a start and end time) associated with a description. Assistants have a set of Time Blocks that they are unavailable for. Sections have a Time Block that they meet during.

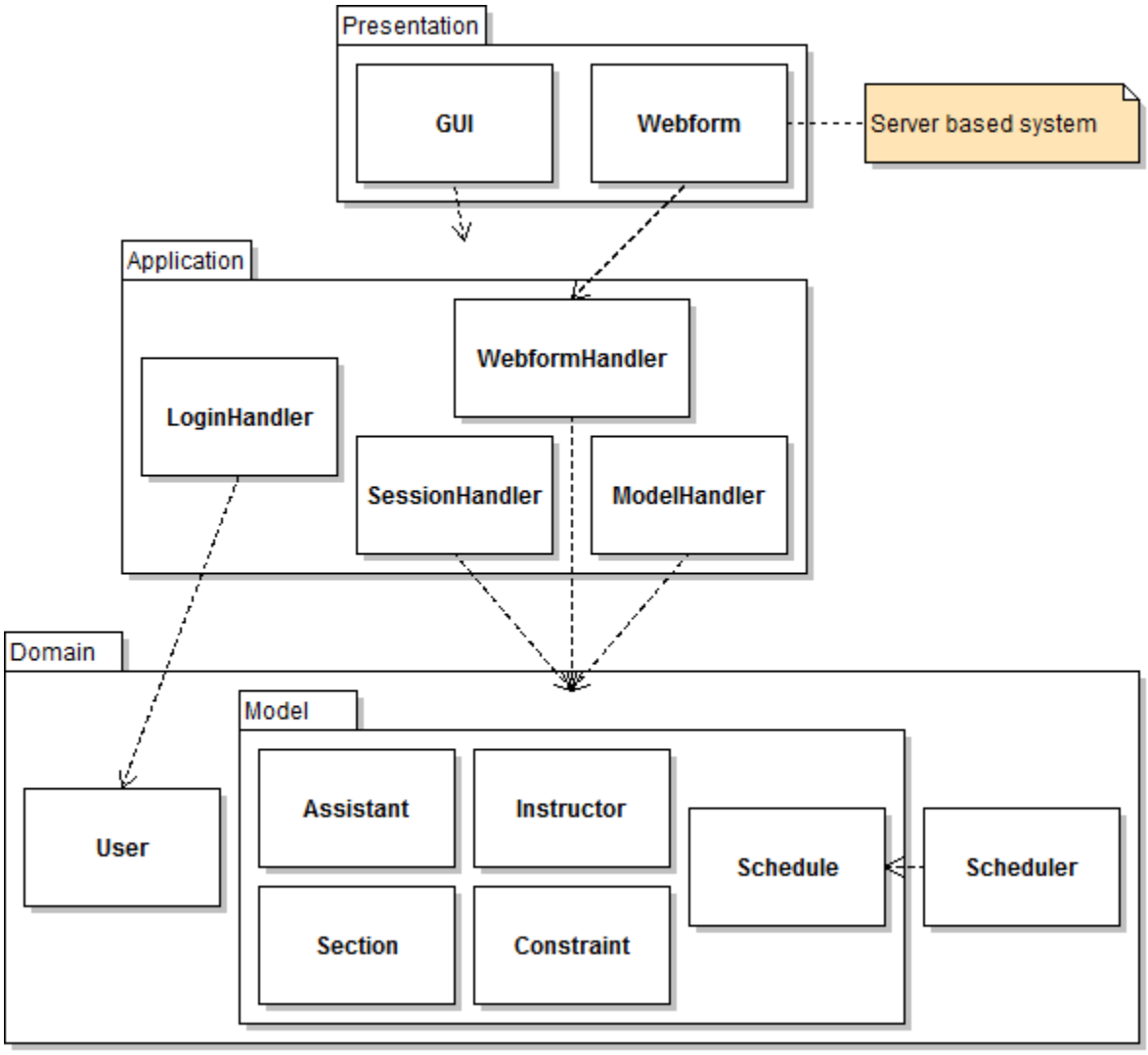
User: One who interacts and runs our Assistant Scheduling System and has access to all functions of the program. They can add, remove, or modify all data in the system including but not limited to Assistants, Courses, Sections, Instructors, and Constraints. Based on their input they will run the Assistant Scheduling System to produce a Schedule. They can access the System a GUI.

Webform: Adjunct system to the Scheduler, accessed via a web browser by an Assistant. It allows for input of Assistant Time Blocks without entry via the Scheduler. Assistants can also use it to review and accept Schedules, and see the finalized schedule for the semester. Assistant Data and schedule accepts/rejects are queued in the system.

DOMAIN MODEL

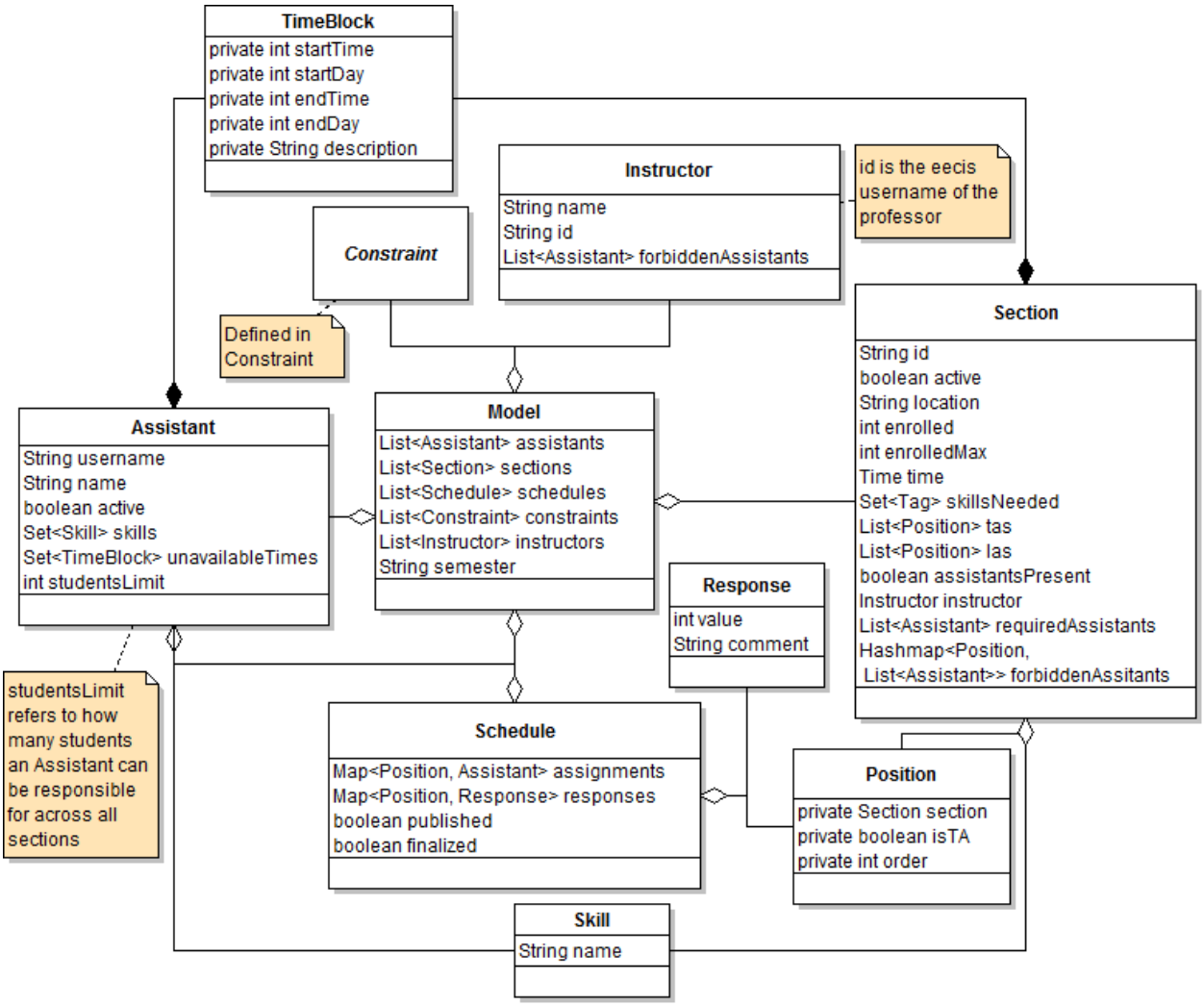


SYSTEM ARCHITECTURE

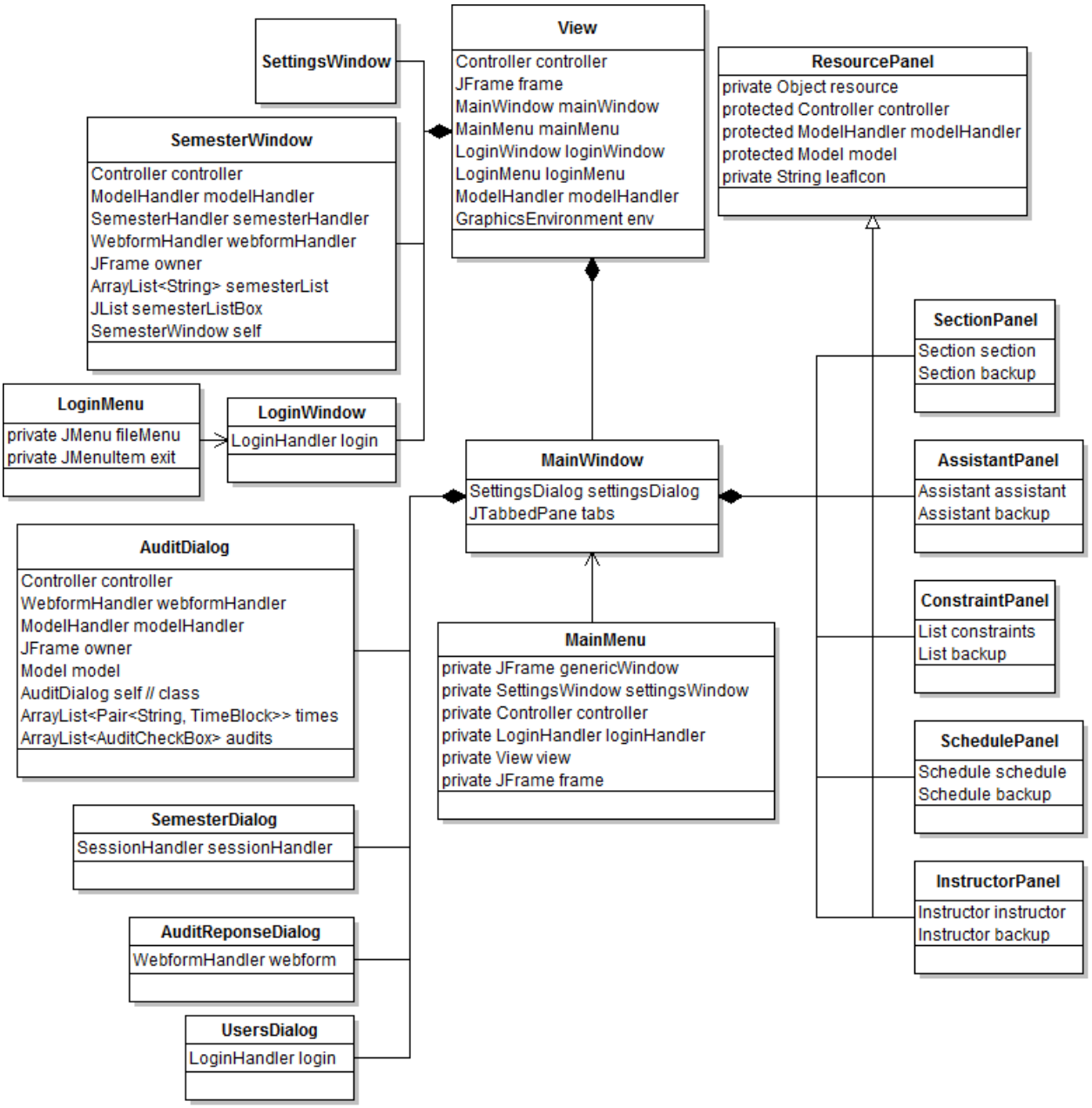


CLASS DIAGRAMS

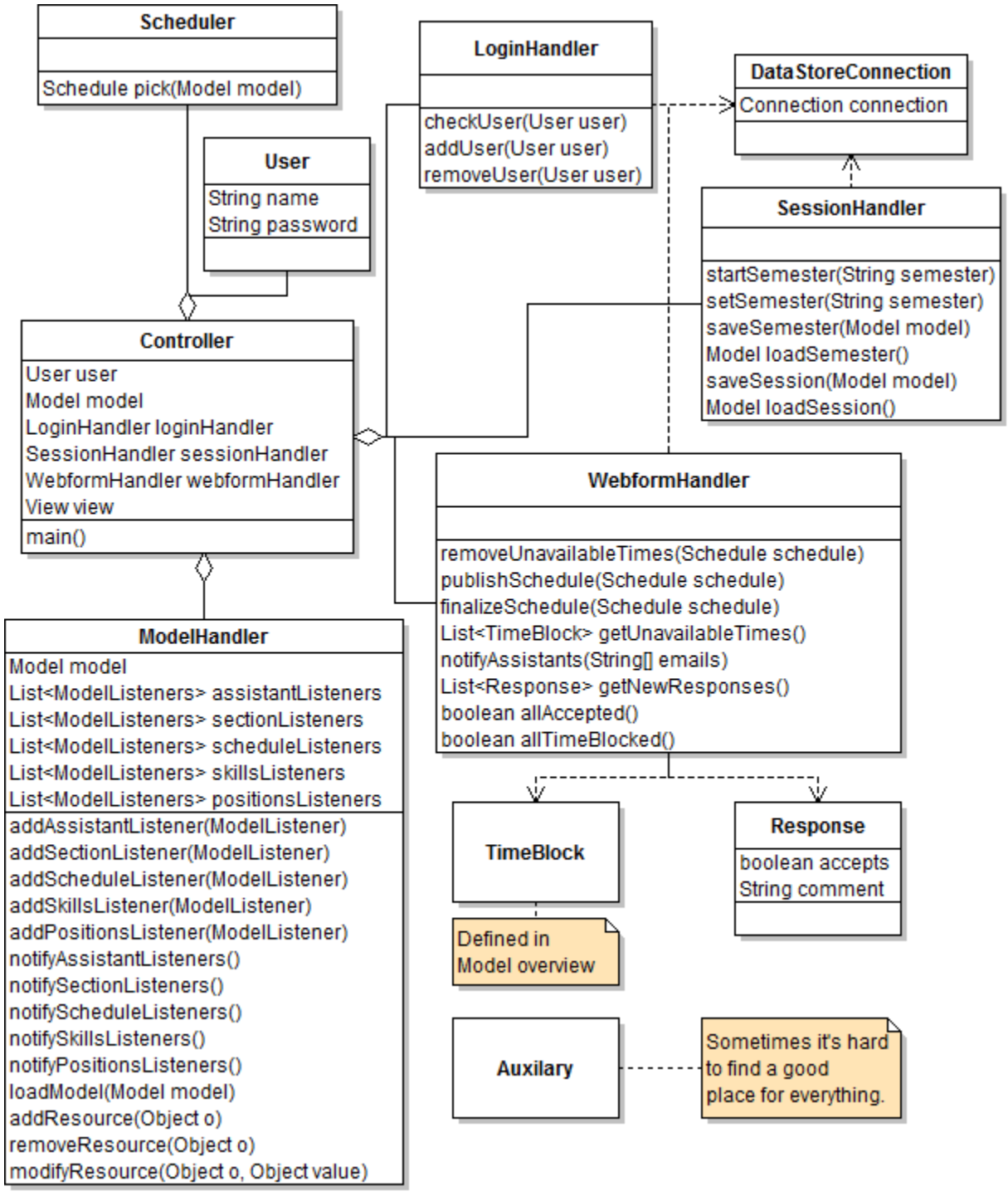
MODEL



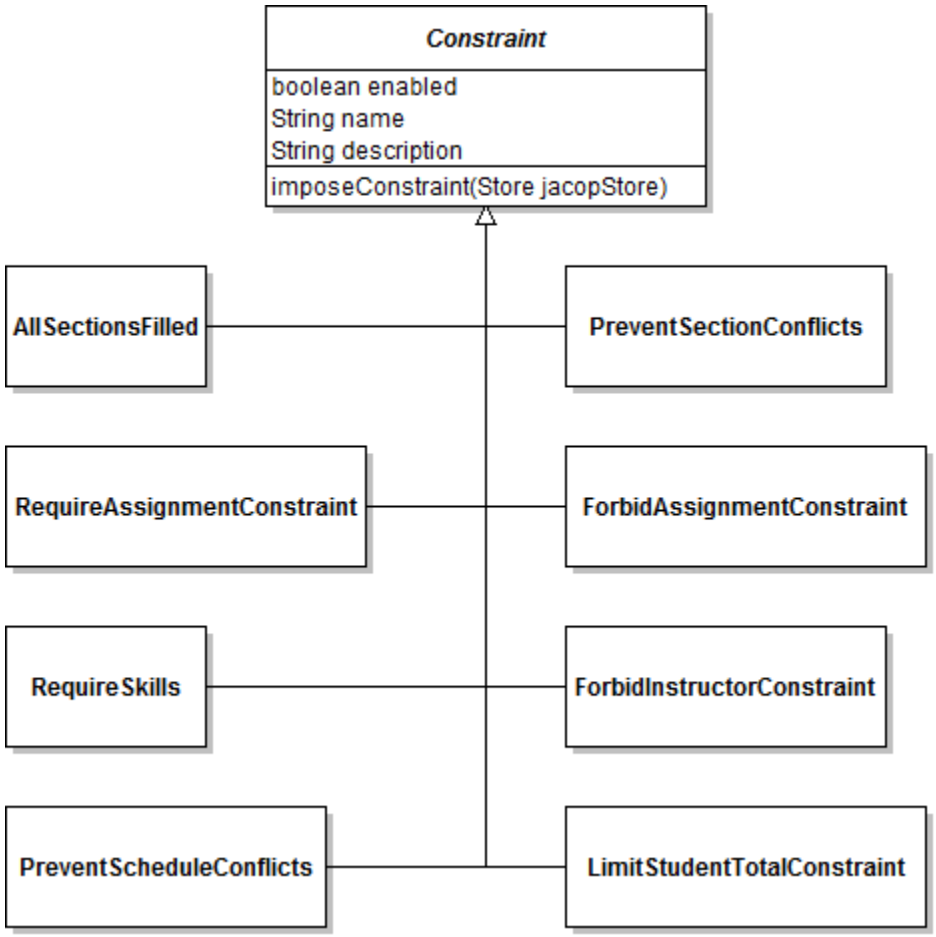
VIEW


SettingsWindow
View
Controller controller
JFrame frame
MainWindow mainWindow
MainMenu mainMenu
LoginWindow loginWindow
LoginMenu loginMenu
ModelHandler modelHandler
GraphicsEnvironment env
ResourcePanel
private Object resource
protected Controller controller
protected ModelHandler modelHandler
protected Model model
private String leafIcon
SemesterWindow
Controller controller
ModelHandler modelHandler
SemesterHandler semesterHandler
WebformHandler webformHandler
JFrame owner
ArrayList<String> semesterList
JList semesterListBox
SemesterWindow self
SectionPanel
Section section
Section backup
LoginMenu
private JMenu fileMenu
private JMenuItem exit
LoginWindow
LoginHandler login
MainWindow
SettingsDialog settingsDialog
JTabbedPane tabs
AssistantPanel
Assistant assistant
Assistant backup
AuditDialog
Controller controller
WebformHandler webformHandler
ModelHandler modelHandler
JFrame owner
Model model
AuditDialog self // class
ArrayList<Pair<String, TimeBlock>> times
ArrayList<AuditCheckBox> audits
ConstraintPanel
List constraints
List backup
MainMenu
private JFrame genericWindow
private SettingsWindow settingsWindow
private Controller controller
private LoginHandler loginHandler
private View view
private JFrame frame
SchedulePanel
Schedule schedule
Schedule backup
SemesterDialog
SessionHandler sessionHandler
InstructorPanel
Instructor instructor
Instructor backup
AuditReponseDialog
WebformHandler webform
UsersDialog
LoginHandler login

CONTROLLER

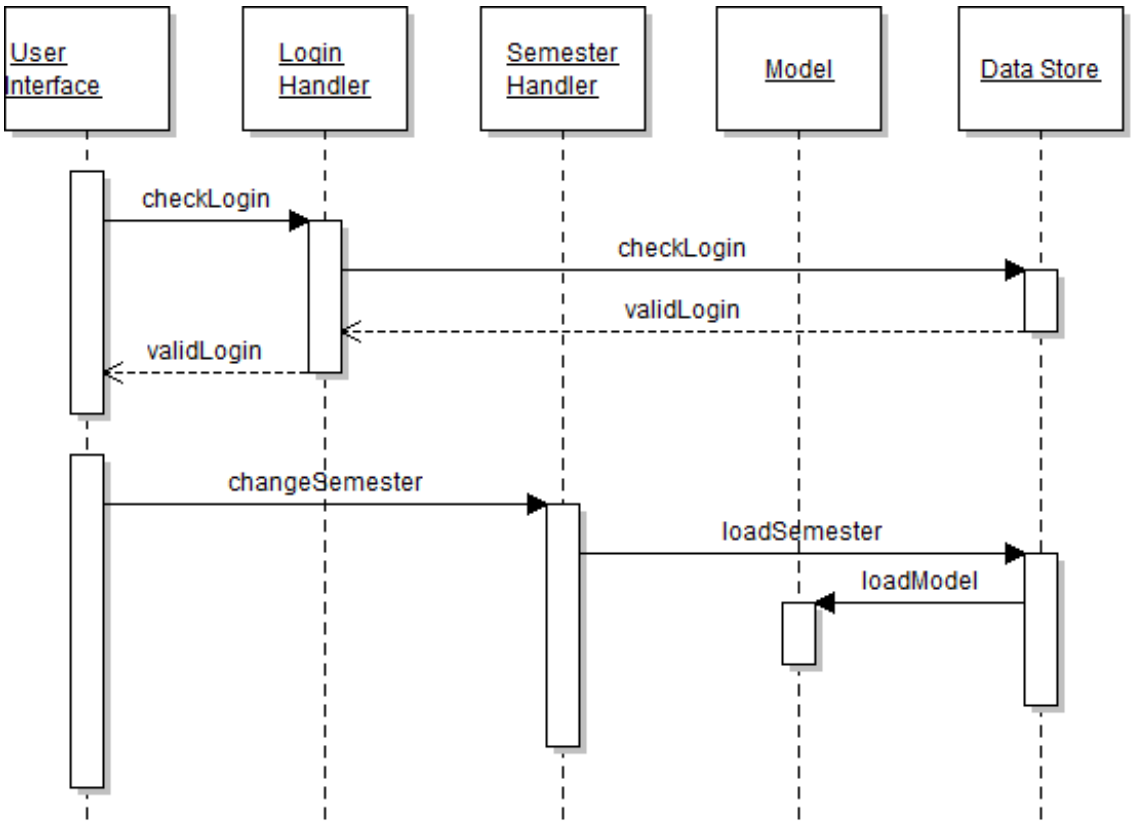

Scheduler
Schedule pick(Model model)
LoginHandler
checkUser(User user)
addUser(User user)
removeUser(User user)
DataStoreConnection
Connection connection
User
String name
String password
SessionHandler
startSemester(String semester)
setSemester(String semester)
saveSemester(Model model)
Model loadSemester()
saveSession(Model model)
Model loadSession()
Controller
User user
Model model
LoginHandler loginHandler
SessionHandler sessionHandler
WebformHandler webformHandler
View view
main()
WebformHandler
removeUnavailableTimes(Schedule schedule)
publishSchedule(Schedule schedule)
finalizeSchedule(Schedule schedule)
List<TimeBlock> getUnavailableTimes()
notifyAssistants(String[] emails)
List<Response> getNewResponses()
boolean allAccepted()
boolean allTimeBlocked()
ModelHandler
Model model
List<ModelListeners> assistantListeners
List<ModelListeners> sectionListeners
List<ModelListeners> scheduleListeners
List<ModelListeners> skillsListeners
List<ModelListeners> positionsListeners
addAssistantListener(ModelListener)
addSectionListener(ModelListener)
addScheduleListener(ModelListener)
addSkillsListener(ModelListener)
addPositionsListener(ModelListener)
notifyAssistantListeners()
notifySectionListeners()
notifyScheduleListeners()
notifySkillsListeners()
notifyPositionsListeners()
loadModel(Model model)
addResource(Object o)
removeResource(Object o)
modifyResource(Object o, Object value)
TimeBlock
Defined in
Model overview
Response
boolean accepts
String comment
Auxilary
Sometimes it's hard
to find a good
place for everything.

CONSTRAINT

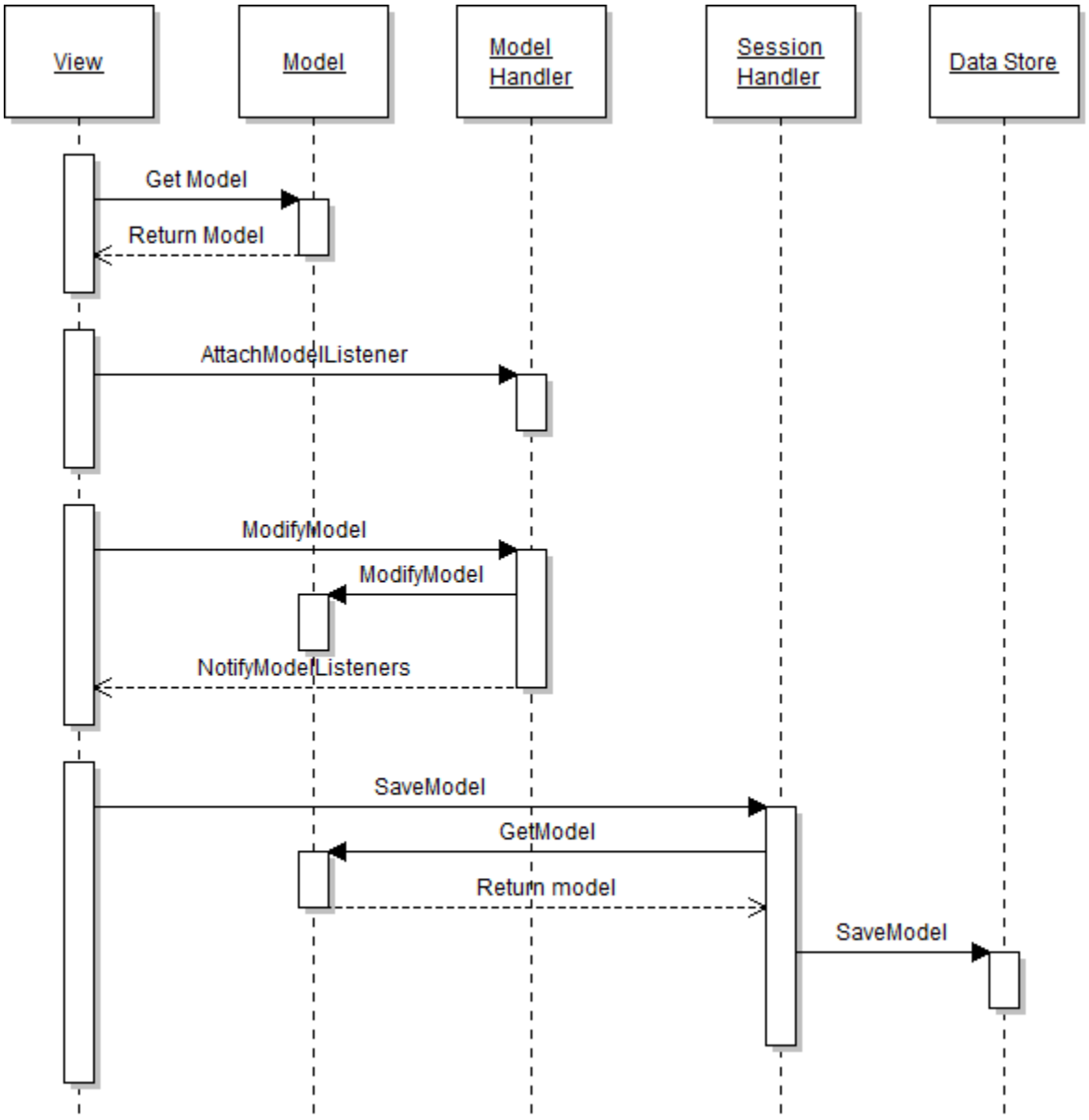

Constraint
boolean enabled
String name
String description
imposeConstraint(Store jacopStore)
AllSectionsFilled
PreventSectionConflicts
RequireAssignmentConstraint
ForbidAssignmentConstraint
RequireSkills
ForbidInstructorConstraint
PreventScheduleConflicts
LimitStudentTotalConstraint

SEQUENCE DIAGRAMS

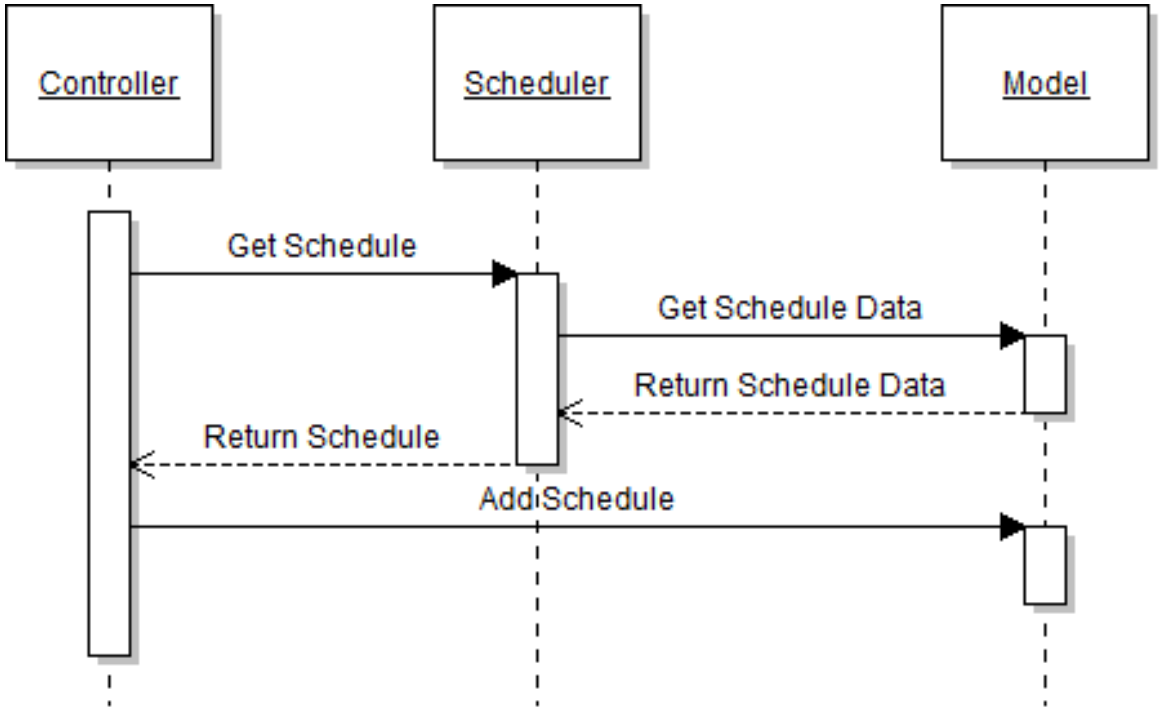
LOGIN



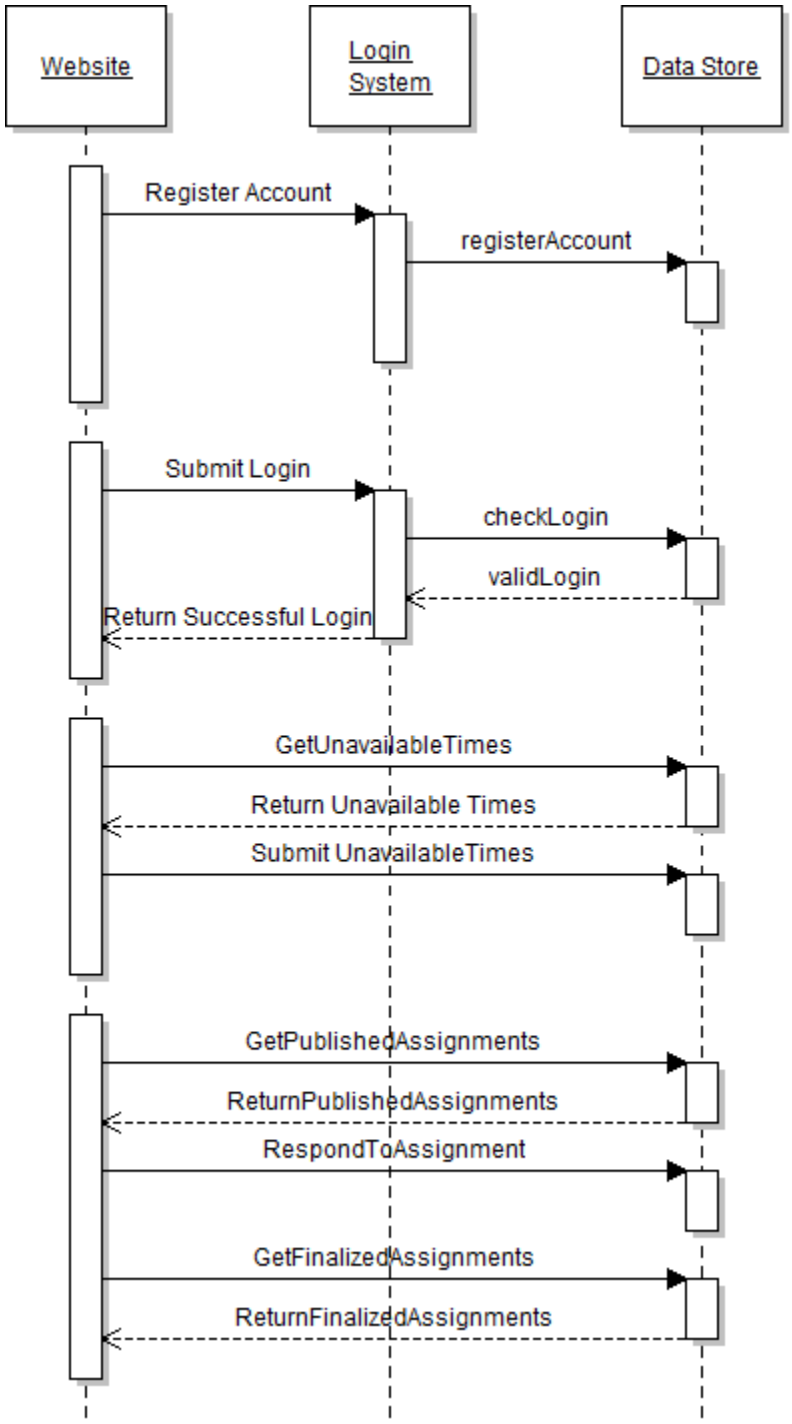
SAVE SESSION



PICK SCHEDULE

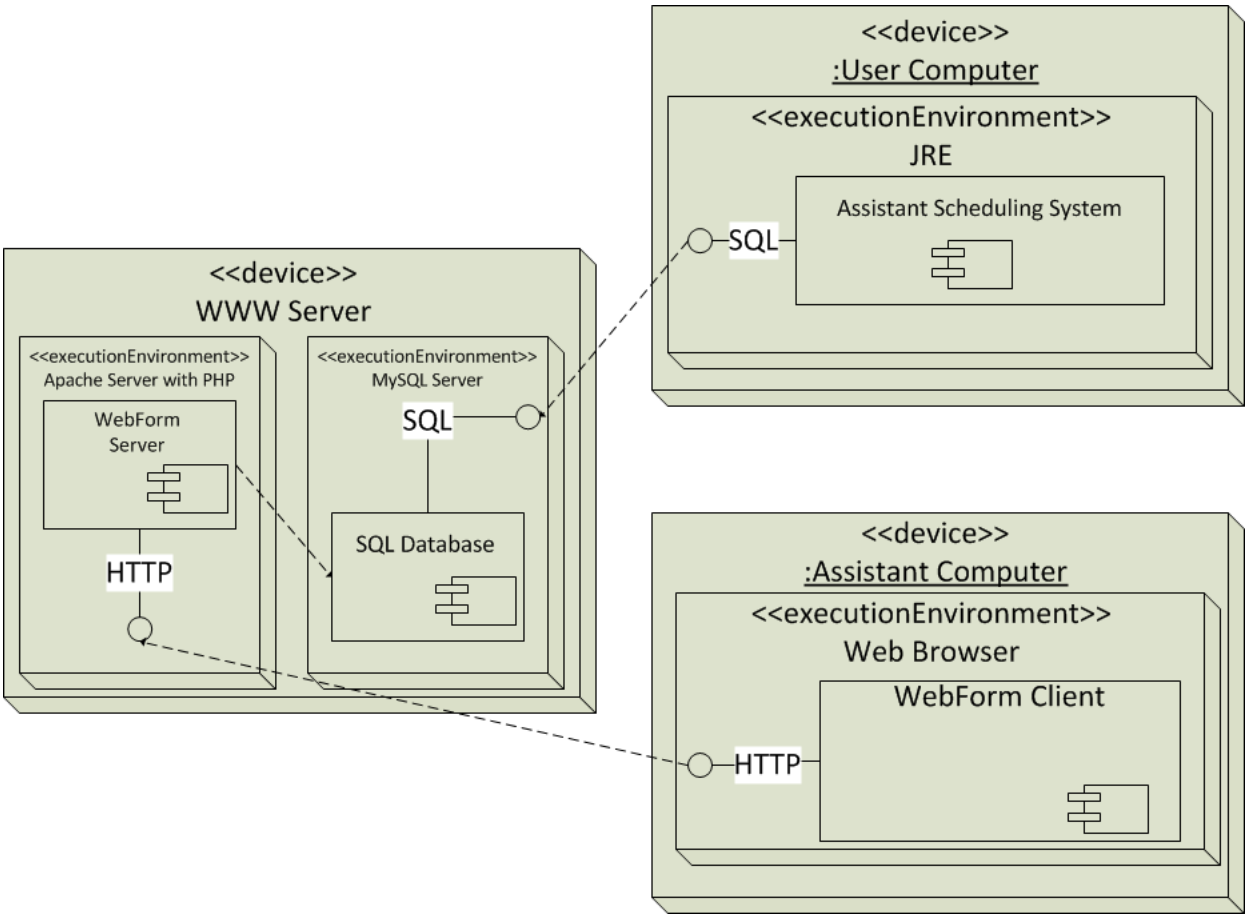


USE WEBFORM


Website
Login System
Data Store
Register Account
registerAccount
Submit Login
checkLogin
validLogin
Return Successful Login
GetUnavailableTimes
Return Unavailable Times
Submit UnavailableTimes
GetPublishedAssignments
ReturnPublishedAssignments
RespondToAssignment
GetFinalizedAssignments
ReturnFinalizedAssignments

DEPLOYMENT ARCHITECTURE

DEPLOYMENT DIAGRAM



DEPLOYMENT LOGISTICS

What gets installed where:

1. Server
    a. Install Apache
    b. Enable PHP in Apache
    c. Install MySQL Server
    d. Setup SSL Certificate on Apache if a signed one is being used.
    e. Open port 80, 443 and 3306 in the firewalls protecting the server.
    f. Place everything from php folder in the root of the Apache webserver.
    g. Run sql command line application and run the following command:
        i. "@<Directory where kiwi.sql is located>/kiwi.sql"
2. Assistant Computer
    a. If one is not already installed, install one of the listed modern browsers.
    b. Go to the web address of the Apache server.
3. User Computer
    a. Install the Java Runtime Environment
    b. Run Kiwi Assistant Assignment JAR

Required supporting software:

1. Server
    a. Apache Web Server
    b. Apache PHP Support
    c. PHP LDAP Extension
    d. PHP MySQL Extension
    e. MySQL Server
2. Assistant Computer
    a. Modern Web Browser (Firefox 3.6+, Chrome, IE 8+, Safari, Opera 10+)
        i. Others may work but they are not supported
3. User Computer
    a. Java Runtime Environment

TEST MODEL

To test our Assistant Scheduling System we will first create JUnit test files for all classes in our project. We will write thorough JUnit tests for all methods in the java classes from trivial "getters" and "setters" to long methods which perform many tasks. For user interface tests, we will also run thorough our use cases to make sure that interacting with the program's user interface acts the way we intended the system to work. All of our JUnit tests can be found in the kiwi.unitTests package. In the unitTests package there is a test suite which will run all JUnit tests however each JUnit test can be run by themselves. This allows us to quickly test changes and do a full regression test depending on what we decide to run.

For the webform we plan to do user completed tests, checking values in the mysql database and visual elements in the webform itself.

PROBLEMS AND LESSONS LEARNED

Over the course of the design and implementation of this project, a number of challenges were met. Some of these problems were technical in nature, such as the ones involving constraint satisfaction and third party libraries. Some of the problems were more human-oriented, typical of a project of this scope. All the difficulties encountered were overcome with collaboration from the group.

Perhaps the most central issue encountered in creating this system was performing the constraint satisfaction. A considerable amount of research had to be done to explore the different ways that the root problem (known in the literature as the "Nurse Scheduling Problem") could be solved. We explored a stochiastic search approach, but instead opted for a more traditional Constraint Satisfaction system. Eventually, in the interest of time and simplicity, the group decided to use a third-party system named JaCoP, chosen for it's documentation, acceptable benchmarks, and comparitively well-maintained code base.

JaCoP was not the only source of contention. As part of the developing requirements of the system, it was decided that the system should involve a Data Store. Initially, this system was to be done in MySQL. Discussions with the professor and teaching assistant led the group to decide that MySQL would be too heavy for our purposes, and we switched to a folder-based structure with data files. Eventually, however, problems with concurrent file access and connection protocols led us back to MySQL.

The final major technical problem that occurred in the system was working with Java Swing libraries. The group found the library to have a complex data model and complicated layout engine. Ironing out errors in component placing took up an unfortunate amount of time and distracted the group from work on internal systems. The only solution was to spend more time learning and testing the systems in order to get the kinks out.

As previously mentioned, not all the problems were strictly technical. For a large portion of the semester, there was limited communication with our client. This lack of dialogue eventually resulted in a grave misunderstanding of the requirements of the system, causing us to waste a large amount of time and energy in creating an over-generalized constraint specification system. Ultimately, contact was reestablished and the mistake corrected.

The final difficulty we had was handling the task management. Group members had a number of issues adjusting to TRAC, and in the end, it was decided to post work assignments via Google Docs and use it to collaborate on documents. This allowed us to forgo many group meetings and complicated email chains. Google Docs also had the added advantage of keeping a revision history and cloud storage, ensuring recovery of lost work and ease of access

PRIMARY AUTHORS

Austin Cory Bart:

- Supplementary Requirements
- Problems and Lessons
- Domain Model

Andrea Macartney:

- Formalized Use Cases
- Use Case Models
- Glossary
- Vision

Michael Chinn:

- Deployment Architecture
- Test Model

Etornam Banini:

- Sequence Diagrams
- Class Diagrams
- System Architecture

Will: