S.C.O.R.E Report for MyCourses A Course Scheduling System UCSC Team

Ben Ross (benr22@gmail.com)
Erik Steggall (eqstegall@gmail.com)
Justin Lazaro (jlazaro@ucsc.edu)
Sabba Petri (sabbap@gmail.com)
Will Crawford (wacrawfo@ucsc.edu)

1 Development Process

This instance of the MyCourses course scheduling system was created as part of a class instructed by Professor Linda Werner here at UCSC. As such, our instructor required several deliverables from us apart from the requirements of the SCORE project - primarily presentations and the delivery of documents - as part of a modified Unified Process development process.

Participation in this class required us to approach design very seriously before we started coding. This was a new approach to us; most of us were tempted to go directly for some sort of rapid prototype.

After gathering the requirements from the SCORE project description, we analyzed them, immediately noticing that the problem space for scheduling classes would probably be NP-complete. Much of our time in this phase was spent shopping around for an appropriately similar problem that we could model our actual problem on.

During the design phase,

2 Requirements: Problem Statement

3 Requirements: Specification

This section is designed to provide an overview of S.C.O.R.E. Scheduling based upon the requirements above. Within this section, one will understand the power behind the technology used for this system: the web server technology and the algorithm used to schedule the classes. To begin, the web server for the S.C.O.R.E. Scheduling system is Django, strictly because of its ability to ease the creation of database-driven websites. It provides an abstracted framework for dynamic websites that is easy to learn and easy to integrate. The available components also will help us throughout this design process, such as the clever template system interfacing with the user interface. Django is written in Python, which is the language of choice for this project, as well as the language of our algorithm.

When deciding upon an algorithm, we needed to decide whether to go with a constraint-based solver, or to design an algorithm in-house to fulfill out needs. Because the scheduling problem is an NP-Hard problem, by using a constraint-based solver, it would ease the programming end of the algorithm team; however, learning to use a constraint-based solver would require more time to learn. On the other side to this decision, designing an algorithm in-house would provide full control of the algorithm in how we decide to integrate it with the web server framework.

Looking at figure three, the class diagram depicts an overall outlook of our in-house algorithm, of which will be the heart and engine of our Scheduling system. This class diagram shows the relationship between the configuration Python file with the different database attributes, such as professor, course, room, schedule, and course class. The configuration class is the primary driver that weaves all the user groups and database attributes together to function as a single entity.

Figure 1: Class Diagram of Algorithm

Once one understands the algorithm above, then it can then be applied to the multiple use cases that it will be serving for. The interaction diagrams below contain every interaction of each user: student, lecturer, program manager, and program administrator. Each diagram details a pictorial representation of the users interaction with the system, such as the HTML website, the Django engine, the database thats handled, or the algorithm itself. In all these diagrams, the HTML portal, or front-end website, is an input/output object wherein users will input items and receive output from Django. From behind the front-end website, Django serves as a control object that handles all aspects of user, algorithm, and database interactions. The database itself is an entity that stores information from user input processed by Django, and it also provides output to the user. Lastly, the algorithm is a control entity that manipulates the database courses, but all functionality of the algorithm remains hidden from the entire system.

Figure 2: Student interaction diagram

1. The student will login with their ID and password on the portal. Django will read the ID and password and match it with the database.
2. Once the database confirms user, the database will return the correct UI for the student.
3. The student will want to get the schedule of classes to sign up for and makes a request to Django and the database to display the available courses. By making a request, this is done invisible to the user and the user will simply click on the courses tab. It will open automatically for the user.
4. The student can now select courses and store them into their profile on the database.

Figure 3: Program Admin enters courses interaction diagram

1. The program administrator will login with their ID and password on the portal. Django will read the ID and password and match it with the database.

2. Once the database confirms user, the database will return the correct UI for the program administrator.

3. The program administrator will want to know the status of the schedule, to see if it is finished or not, and will make a request to both Django and the database to find out the status.

4. The database will return the status.

5. The program administrator will begin filling out course information through a courseInfo object that will contain various data about a given course.

6. When the program administrator feels done, they can choose to submit the course and lock the course in the database.

Figure 4: Program Administrator runs algorithm interaction diagram

1. The program administrator will login with their ID and password on the portal. Django will read the ID and password and match it with the database.

2. Once the database confirms user, the database will return the correct UI for the program administrator.

3. The program administrator will want to know the status of the schedule, to see if it is finished or not, and will make a request to both Django and the database to find out the status.

4. The database will return the status to see if the program manager has completed the schedule.

5. If the schedule is complete, the program administrator will run the algorithm that will be passed through Django and straight into the algorithm engine.

6. When the engine has completed, it will return a message that it is complete to the HTML website.

7. Once the program administrator reviews the schedule, the program administrator can submit and lock the schedule and provide it to students

Figure 5: Lecturer interaction diagram

1. The lecturer will login with their ID and password on the portal. Django will read the ID and password, and match it with the database.
2. Once the database confirms user, the database will return the correct UI for the lecturer.
3. The lecturer will want to find out the constraint form has been completed. Upon logging in, Django will send a message on the status of his constraint form, based upon a status in the database.
4. The lecturer will fill out the constraint form and store it in the database.
5. Upon storing it in the database, the lecture status will be updated and confirmed.
6. The lecturer can now formally submit his constraints form to the database.

Figure 6: Program Manager interaction diagram

1. The program manager will login with their ID and password on the portal. Django will read the ID and password and match it with the database.
2. Once the database confirms user, the database will return the correct UI for the program manager.
3. Upon logging in, Django will automatically send a request to find out the department status of the courses.

4. If the status is not yet been completed, it will send a message to the program manager.

5. The program manager can modify the courses that was made available from the program administrator and placed in the database.

6. Upon modifying the courses and the course info, the program manager can confirm and submit the finalized course.

4 Architectural Design

Our architecture is broken into three main categories:

1. User interface or UI (The front end)

2. Django (The back end)

3. The algorithm (The engine)

4.1 The User Interface

There are four different user interfaces, each is assigned to one of the four roles that a user could be: Program administrator, program manager, lecturer, student. The students UI is the most complex and offers the most customization.

The program administrators user interface is fairly simple, they are given fields such as courses, rooms, and professors to fill out, once they fill out the fields to the proper specifications for the quarter then they have an option to run the algorithm. The algorithm will run, and then post up the results back to the program manager.

The program manager's interface will hold all of the courses and lecturers for their department. The program manager will have the option of selecting the lecturers and courses that they want to be active for the quarter.

The lecturer will have an interface that will hold the courses that they can teach and the current courses that they are teaching. The student will have a customizable interface which will hold their past, present and future schedules. It will also have other widgets that they will be able to add to their profile.

4.2 Django

Django connects the user interface, the database and the algorithm. Django manages the user interface by handling the user requests and providing the appropriate response. In the case of the algorithm it takes the request from the program administrator to run the algorithm and creates a request object and sends it through multiple middlewares (Python functions). Django then checks the URLs to decide which function it must send the request object to. The function in this case is the algorithm, which takes the information stored in the program administrators database and runs. The output of the algorithm is placed into a second database and the request object is sent back up through the middleware in the reverse order of the way it came in. The second database is then displayed to the program administrator.

4.3 The Algorithm

5 Project Plan

The project plan will describe the date and deliverables of our development process. The SCORE project followed a software methodology course and many of the deliverables are similar to those in the SCORE requirements.

Instruction and classes began on September 23, 2010 and would continue for 10 weeks until the end of the quarter. Further development of our scheduling system will continue on to the winter quarter starting January 4, 2011 through March 18, 2011.

Week 1: Initial Presentation

For the first week, we had to gather in groups of 5 to form teams for our project and give an initial presentation on a brief overview of our project.

We covered the following items:

- Team name
- Team members
- User experience
- Core functionality
- Wish list
- Possible project risks

Week 2: Requirements - Scenarios

During this week, our team designed several user scenarios related to our project. The scenarios will be a significant part of our requirement document and will be important in defining various aspects of our systems functionality. We came up with 5 user scenarios: (2) program administrators, program manager, lecturer professors, and students.

Each scenario outlined and described the end users interaction with the system, the data flows of information, and the interaction with different functions and systems as the user performed a particular action.

Week 3: Requirements - Complete

After defining our scenarios, we created a complete and defined requirements document which will be necessary for implementing our system in a clear and concise matter. It details the specification of the functionality and constraints on that functionality for our scheduling system.

The requirements we cover include the following:

- Functional - A specific and detailed description and list of what our system is
- Performance - Specific characteristics of the performance of our system
- Usability - How users interacts with our system
- Wish list - System features we hope to implement if time permits

- Coding Standards - Rules for organizing and formatting code
- Preliminary User interface - Preliminary sketch drawings of what our user interface will look like.

In this week, we also had to give a Design Presentation which will basically outline everything in our requirement documents and how it is done.

Week 4: Architecture and Design Document

After defining our requirements and scenarios, we create an architecture and design document that will document the high-level architecture and the detailed design of our project. It will have a detailed description of the objects that we will use as well as relationships between objects.

We created the following items:

- Overview - The overview document will provide context and an overview of our diagrams. It will include description of our major design decisions and modularization criteria.
- Architecture diagram - High-level overview of the components in our system and how they co-operate.
- UML Structure diagrams - Diagrams of our class objects in the form a UML structure diagrams that describe the attributes and operations.
- UML Interaction Diagrams - A developed UML diagram based on several of our scenarios we produced for our requirement documents.

Week 5-6: Development and user manual

During this phase of our course, implementation of our system was to commence and continue for the next several weeks. This includes implementing our design and requirement documents. As we were creating this system, we were also simultaneously required to create a user manual that details how our system is to work. The user manual was broken down into the following parts:

- Purpose - Description of the UI and functionality

- How do you do this? - Defining user classes, convey the user experience level at each of the classes, and provide computerized help.

Week 7: Software inspection

A software inspection is an in-class presentation where our group simultaneously and systematically reads source code and identifies and classifies defects as they are encountered. For our software inspection, we chose our algorithm class that will be the primary engine that will run our course scheduling system.

It is also important to note that a software inspection is strictly a time to catch defects and not make an attempt to discuss how to fix it or what to do to fix it. There is no discussion of the problem at this time, it is simply to define the problem and make not of it to the developer.

Week 8: Unit Test

Our group created several unit tests that tested various classes and modules of our system. These would be simple script code that automated and simplified testing for us. Since our project was primarily written in Python and Django, we created very simple Python scripts that quickly tested how each class and the expected responses.

Week 9-10: Acceptance Test

With our development, testing, and inspection complete, we demonstrate our system to our primary stake-holder (our professor). An acceptance test is generally a basic run-through of our entire end-to-end user experience of our system. At this point, the quarter is about to end and all our project deliverables should have been completed and ready to be delivered as a final product. Due to the scale of our particular project, we unfortunately did not have a complete end-to-end system, however, bits and pieces of our system worked well when isolated.

Due to our system at this point of development was not complete; we would spend the next academic quarter completing development of our project. The next academic quarter at UC Santa Cruz would not begin until January 4, 2011 and will end on March 18, 2011. The following is our tentative plan for the quarter to complete this Scheduling system.

Week 1: Review

Our plan is to review our system and deliverables. We define items we need to complete or improve on. We also make the decision to revise our life cycle model from Unified Process to an Agile SCRUM software methodology. We plan on hosting small update meetings every evening to inform each others progress with development, along with weekly major meetings where we sit down and discuss items as necessary.

Week 2: Documentation improvement

With our SCORE submission coming near, we focus this week on improving and revising any necessary documents needed to be submitted under SCORE requirements.

NOTE: As of now, we have not yet defined the rest of our continued development after our SCORE submission.

6 Management Plan

Because the skill sets of our team members were remarkably heterogenous, proper management of our team was crucially important as it allowed us to delegate work that we considered specialized to the proper specialist on our team in order to complete work in an efficient manner.

6.1 An Overview of the Management Process

6.2 Channels of Communication

Subversion (SVN)

As part of the class, we were required to use Subversion as a project repository for our code, as well as other deliverables for the class. Subversion is a source control mechanism that allows multiple people to remotely collaborate on the same codebase. We used Subversion to transfer code and miscellaneous files between team members. We also used it to automatically merge source code files.

Google Groups
: For the purposes of communicating by e-mail, we created an e-mail list with Google Groups; it allowed us to automatically archive our e-mail conversations and bounce all communication to each team member. Scheduling meetings was done primarily via this e-mail list. It also initially served as a rudimentary feature tracker.

Daily Skype Meetings
: After the conclusion of the class, we continued to meet via Skype every night for 15 minutes to check in with each other. These meetings were intended to briefly answer three questions: “What did you work on today?” “What are you working on tomorrow?” and “Is there anything preventing you from making progress?”

Git & Codaset
: After the class concluded, we switched from Subversion to Git as more of our team members were comfortable with it and preferred it. Git is another source control mechanism, but we found its granularity regarding which files it committed to be superior to that of Subversion. We also found its merging algorithm produced fewer conflicts during our usage. The website Codaset (`http://codaset.com/`) served as a central repository for the highly mobile team, as well as a GUI for viewing the repository's metadata, such as commit histories, volume, etc.

7 Implementation

Implementation of the MyCourses scheduling system was a multifaceted problem. The overall system was broken down into four main parts to implement: Web Interface, Database, Algorithm, and Security. Our implementation strategy was to find the best way to implement the parts separately, and then collectively integrate them together. This would allow us to selectively work on each part of the combined system separately.

The implementation of the Web Interface is done in HTML5, with a Django Server backend. HTML5 was selected over previous versions of the markup language because it allows development with less JavaScript and Flash. Even though these technologies are very popular on the Internet, staying away from them allows us to increase the browser compatibility of our system. As newer browsers are released, HTML5 support increases, which permits us to use newer features of the language to add newer features to the system as the client desires.

The Django Server backend was selected for a few reasons. Django is a free Web Server backend that allows for easy control of databases, as well as Pythonic ways to display information on varying web pages. This allows us to integrate the algorithm with the database seamlessly. This was to be a major problem facing our system, but was quickly alleviated earlier in the process. Django uses forms and templates to display information from the database to web pages and this allows us to create a few templates for each of the authentication levels, all of which are able to display information directly from the database with no SQL queries. Because Django performs the CRUD on the database, Python queries are performed with great speed, letting pages to load quicker.

The Algorithm went through three different implementations before the best implementation was selected. The first implementation was a slow Algorithm designed by one of the team members. It was very inefficient, but allowed the program administrator complete history of the courses that were scheduled at a given time, in a given room. This allowed for easier manual edits of the proposed schedule. The second implementation was using a FLOSS constraint-solver Minion. This solution was very fast, but needed a complicated script to translate from the database to the solver and back again. The third implementation was a scheduling algorithm using a genetic Algorithm. This Algorithm was based off of a freeware genetic algorithm scheduler. The Algorithm was written in Python, allowing it to be fast and easily modifiable. The genetic Algorithm was selected as the best implementation solution, because it allowed seamless integration with the database, and it proves incredibly fast, even for larger solutions.

Security for our system was not something that came up in our initial discussions of the system. However, security is one of the most important aspects of a web application. Also, universities generally have lots of sensitive data displayed on similar applications and thus, security needs to be paramount. The security system we designed into our system also integrates with the privacy settings for the application. HTTPS is used instead of the HTTP; this allows all traffic coming to and from the system to be encrypted. This is a step above popular web applications like Facebook, and Myspace, providing students, faculty members, and administrators peace of mind, knowing their traffic will be very difficult to decrypt. Our second important security feature is passwords. Our minimum password length is 8, with a minimum of one capital letter, one symbol, and one letter. This allows users increased security from brute force password attacks. Another security feature involving passwords is similar to what some banks do. We assign each user a sitekey, which is a random picture out of 50. The user is then requested to name the sitekey. When the user enters their password, if an incorrect sitekey is displayed, the user is made aware of an attempt to compromise their information.

The other security features deal with the privacy of the user once they are in the system. These features are designed for the student role only. The integration of Facebook with our system provided another way for our system to lose security. When the student first logs into the system, a tutorial walks them through setting up their account and linking it with their Facebook. The student has the option to turn off sharing of information during this setup, as well as, anytime through the account settings page. This feature gives permission to students for complete access as to what information they share through our system, and through Facebook. Allowing students this control will decrease the chances of a student sharing more information than they previously wanted to.

8 Validation and Verification

Our test framework for our implementation has two forms. Testing for the algorithm and testing for the web UI. The testing for the algorithm is done by scripts, and the web UI testing is done through a software suite.

The algorithm was first tested by running the algorithm and then manually checking the output. The output was checked to see if the proposed schedule was a valid schedule and to see if all of the constraints were met. This became very tedious for large class schedules, so a script was written that validates the output based on the constraints given in the database. This allowed large testing to be completed much faster.

The web UI testing is done using a framework called Selenium. This open source tool allows automatic testing of the web UI. Two methods were used for writing tests for the web UI. Selenium has an IDE that allows you to record an interaction with the web UI and run that recording as a test. This was done for basic actions, such as logging in and out. The other way are python scripts that use Seleniums built in libraries to interact with the web interface. This allows the tester to check for output on a given page and to perform more complex actions. This was done for advanced actions, such as signing up for classes, or adding students to the database.

9 Outcomes and Lessons Learned