CSCI 2194 (Professional, Ethics, and Design Seminar), Spring 2009

Design Project

Note: The HTML version of this document may contain hyperlinks. In this version, hyperlinks are represented by showing both the link text, formatted like this, and the full URL as a footnote.

1 Project description

1.1 Overview

A perennial problem for our department is scheduling classes in available rooms, because there are lots of constraints to satisfy:

- Some are obvious — for example, two courses can't meet in the same room at the same time, one person can't teach two (different) courses at the same time, not every instructor can teach every course, etc.
- Some are not so obvious — for example, we don't want to schedule two courses likely to have overlapping enrollments at the same time, and some courses may need equipment/software only available in one room.
- Some are more preferences than requirements, but still useful to consider — for example, which courses an instructor prefers to teach, or instructors' scheduling preferences (T/R classes only, e.g.).

At present those responsible for making up the schedule mostly use trial and error, sometimes guided by schedules used in previous semesters. Your mission for this course is to design a system to help automate this process (aiming at a design that could be reused for other similar processes). Some things the system should do:

- Display a schedule in different forms (complete list of courses, similar to what appears in the registrar's schedule of courses; teaching schedules for professors; schedule showing room use).
- Allow authorized users to change a schedule.
- Allow authorized users to enter, view, and edit constraints as appropriate (some constraints — e.g., not scheduling two classes at the same time in the same room — should be built-in and not subject to change).
- Confirm that a schedule meets a set of constraints.
- Ideally, produce a complete schedule given lists of courses, time slots, classrooms, instructors, and constraints. (This may not be feasible, but is worth trying for.)
- Support more than one schedule (to make it possible to consider/compare different options).

You have some flexibility in deciding exactly what functionality to provide; that's part of the design problem. In a real-world situation, you would have a "customer" whose needs you are trying to meet, and part of the design problem might be to turn a vague initial problem description into something more specific. For this class, I will play the role of customer, so part of the analysis phase of the project might be asking me questions to help you clarify requirements. Class time will be available for this purpose, or you may ask me in office hours or via e-mail (being sure to coordinate with other members of your group.)

1.2 Constraints

Customers typically also impose constraints (how much a solution can cost, what hardware/software platforms it must run on, etc.). For this project, the following constraints apply.

- Your solution should be as cross-platform and portable as possible — i.e., it should not constrain users to a particular platform.
- Your solution should not require spending money — e.g., if you use existing products/programs, they must be public-domain / free.

1.3 Project phases

A project such as this one can be broken down into three phases:

- Requirements analysis. In this phase, you analyze the problem and come up with specific requirements that a solution/implementation should meet. During this phase you may want to try to research other programs/products that solve similar problems. We will discuss in class an approach to this phase based on "use cases", which you should follow.
- Implementation design. In this phase, you design an implementation that meets the requirements laid out in the first phase. You should be familiar with object-oriented programming from CSCI 1321, so we will focus on using object-oriented design in this phase.
- (Prototype) implementation. In this phase, you produce a prototype implementation — one that is mainly intended to show how your solution looks to users and therefore does not need to fully implement the parts of the design that don't show. For this project, however, I think you do need to build as much as you can of the part of the program that actually figures out the schedule.

2 Deliverables

The following table lists "deliverables" on which you will be graded. Items with a "Y" in the column labeled "Group?" are to be turned in by group leaders (and should represent the combined efforts of the members of the group); other items are to be turned in by individual students (and should represent individual efforts). Notice that the points add up to the 75 points specified in the syllabus for the project.

What	*Group?*	*Due*	*Points*
Preliminary requirements analysis	Y	February 23	5
Midterm report	Y	March 30	10
Midterm group evaluation	N	March 30	5
Presentation	Y	April 27	15
Final report	Y	April 27	20
Prototype code	Y	April 27	15
Final group evaluation	N	April 27	5

2.1 Preliminary requirements analysis

Your first deliverable is a short report presenting the results of your requirements analysis — a list of use cases, a UML use-case diagram, and a short text description of each use case.

2.2 Midterm report

Your next deliverable is another short report, incorporating the previous report (possibly with improvements or corrections) and adding a sketch of your implementation design using UML diagrams.

2.3 Midterm group evaluation

Each student will also evaluate the performance/contribution of members of his/her group. I will use this information in determining grades. Forms will be provided.

2.4 Presentation

Each group will present its analysis of the problem, its proposed solution, and a prototype implementation. The presentation should include UML diagrams of your requirements analysis and your design and a brief demonstration of your prototype implementation.

2.5 Final report

Your final report should be an expanded version of your midterm report, incorporating any changes you have made (particularly to the implementation design) and also describing your prototype implementation.

2.6 Prototype code

You are not required to come up with a complete implementation of your design; this course is meant to be about design more than implementation. However, before starting a complete implementation it can be extremely helpful to develop a prototype that shows your "customer" what you have in mind, to be sure that what you plan to implement really meets his/her needs. Thus, you will develop a prototype implementation you can demonstrate. Prototype implementations generally focus more on the parts of the system that "show" — that is, the user interface. For this project, ideally you would build as much as you can of the part of the program that actually figures out the schedule, subject to time constraints.

2.7 Final group evaluation

See "Midterm group evaluation."

3 Groups

- Group 1: Ogba, Onyekwere M.; Repsher, Philip A.; Swords, Cameron G.; Tupper, Zachary L.; Wolf, Douglas G. (group leader)
- Group 2: Gunadi, Christopher; Jensen, Philip R. (group leader); Perez, Patricia L.; Thrasher, Elise M.; Welch, Donald A.
- Group 3: Cavin, Christopher E.; Chiu, Ansell L.; Shepherd, William R.; Thornton, William O. (group leader)