Project Description

What problems does the project solve

Course Timetabling

- A university can have different lecture halls with different capabilities, capacities and purposes.
- It can also have students that are taking different combination of courses.
- Therefore, an intelligent solution is required to optimize the usage of halls and period times.
- Course timetabling a course schedule that doesn't create any conflicts to the students attending the course.
- It can simulate a variety of situations like fewer classrooms or changes in course class requirements.

Course Management

- Manages timetable modifications.
- Searches for alternatives that have a minimal impact on the current timetable
- Communicates these changes to affected students and other systems.
- Each alternative produced by the solver provides information on any additional student conflicts created and any other preferences that may be violated.
- Changes can be customized to allow room-only, time-only swaps, or both.

Event Management

- A university can host more then just classes and lectures like Guest speakers, club meetings study sessions.
- UniTime handles the creation of class meetings and examinations in the events calender Automatically.
- Requesting and searching for events can be handled via the Web Interface that is available for all student and staff members.

Examination Timetabling

- Builds a complete exam schedule.
- Minimizes number of conflicting exam placements for students.
- Limits the number of exams per day for students.
- Creates schedules for midterm and final exams.

Student Scheduling

- UniTime supports 2 kinds of student scheduling.
- Batch Scheduling: UniTime automatically enrolls pre-registered students to their suitable classes.

- Online Scheduling: UniTime supports immediate real-time requests to change or apply to classes.
- UniTime's automatic enrollment is optimized to respect student preferences and course structure.

Techniques used to obtain information

We take an opportunistic approach (combinations of top-down and bottom-up approaches) to program comprehension, namely, the Integrated Metamodel for Program Comprehension[1].

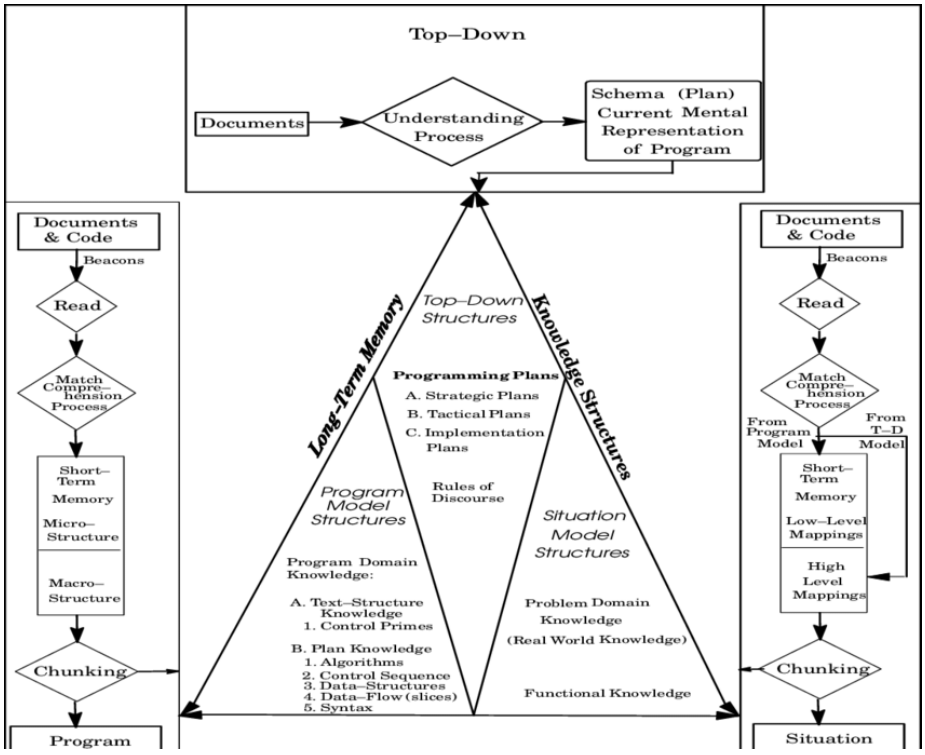

Figure 1: The integrated metamodel for program comprehension

Top down model

- Establish a partial top-down model by reading the help manuals of UniTime
- Read the provided documentation on the course timetabling component
- Read the course timetabling manual

[1]Mayrhauser, Anneliese & Vans, A. Marie. (1995). Program comprehension during software maintenance and evolution. Computer. 28. 44 - 55. 10.1109/2.402076.

- Log in with an admin account, and test Courses > Course Timetabling > Timetable Grid
- Log in with a manager account, and test Courses > Course Timetabling > Solver
- Read the scheduling assistant manual
- Log in with a student account, and test Scheduling Assistant
- And so on...

Situational model

- Establish the situational model by cloning the source code repo, examining the directory and packages structure, building it, and running it locally
- Examine `JavaSource/org/unitime/timetable/server/solver/SolverPageBackend.java` to get an idea of how the solver is called in the back-end
- Examine `JavaSource/org/unitime/timetable/server/solver/TimetableGridBackend.java` to get an idea of how the timetable grid is created in the back-end
- And so on...

Program model

- One way to determine where a feature is implemented is by hypothesizing how it is implemented and what syntactical keywords and beacons[2] might exist in the code.
- Then searching the code for these beacons to reach the target component.
- From the target component, cross-referencing can be done to check what other components are called by that component.
- For example, in order to find the back-end code that calls the solver, we search the code for the word "load" which exists in the solver page.
- We then find out that `JavaSource/org/unitime/timetable/gwt/client/solver/SolverPage.java` is responsible for adding the load button in the solver page (vague but helpful understanding).
- We find out that this page uses an enum called `SolverOperation`.
- We hypothesize that this enum will also be used by the back-end to determine what solver operation should be done.
- So we use cross-referencing to find out all of the components that use this enum
- And we end up at `JavaSource/org/unitime/timetable/server/solver/SolverPageBackend.java`
- We also learn that solver back-end code will most likely be `org.unitime.timetable.server.solver` which is helpful for further refining the situational model.
- And so on...

[2] Beacons are discussed in the integrated metamodel paper[^integrated-metamodel]