FINAL PROJECT

Scheduling system

DEGREE: BSc COMPUTER SCIENCE
DEPARTMENT: INFORMATICS
PROJECT SUPERVISOR: DR. BERNHARD REUS

CANDIDATE NO. 53742
UNIVERSITY OF SUSSEX
25th APRIL | 2009

1 Introduction

The system to be implemented will be an extension to the already existing system, containing all information regarding students' final projects, a list of supervisors, a list of students, a list of projects to choose from and the function to enable students to propose the project. The extension will be in terms of helping to generate a timetable (see functional requirements section) and assigning appropriate faculty/staff as 2nd markers of students' presentations of the final project. The old system has different functionality from the system planned to be implemented; only data will be shared or imported to the new system from the old one. This means that all data regarding students, their projects and faculty staff will be imported. The main functionality of the new system is to produce a timetable, which meets all of the hard constraints (see explanation below).

Explanation of Hard constrains:
Hard constraints determine the constraints that must be fulfilled in order to consider a generated solution/timetable to be acceptable/valid. If a generated solution does not fulfil hard constraints then the solution will not be accepted as a valid timetable.

Producing timetables or scheduling is a general problem and applies to many real life problems in industry or other places. Scheduling is a NP-complete problem (NP standing for Nondeterministic Polynomial time). The difficulty with NP problems is that they cannot be solved fast. As the size of the problem or the size of the input rises, the time required to find a solution can exceed into billions of years using the current computation power and most efficient algorithms. At present, to solve NP problems, many techniques are used such as approximation, randomization, restriction and heuristic.

Definition of scheduling:
"Scheduling concerns the allocation of limited resources to tasks over time. It is a decision-making process that has as a goal the optimization of one or more objectives." [Pinedo, 1995]

"Scheduling problem events must be arranged around a set of timeslots so as to satisfy a number of hard constraints and optimize a set of objectives. Types of scheduling problem differ in terms of the kinds of constraints and objectives involved." [Corne, Fang, Mellish]

2 Professional considerations

The interface created must follow regulation and be designed in such a way that it takes on board the needs of the users along with their abilities and consequently not discriminate against sex, ethnic origin and disability.

The key issues to take into consideration within this project are as follows:

To make sure that the needs and requirements of the user are fully understood and agreed before starting the project.

To make sure that agreement is made on the requirements so the user knows the realistic end product before agreement.

To make sure to complete thorough research in order to find previous pitfalls in similar work and develop my product so that it has a few errors as possible.

To make sure that requirements, building and testing are effectively carried out in order to produce a sound finish product.

To "Produce design specifications that clearly state the objectives, scope, features, facilities, reliability, resilience, constraints, environment, system functions, information flows and traffic volumes as well as identifying requirements not met and scope for improvement." [code&conduct]

To "Strive to achieve well-engineered products that demonstrate fitness for purpose, reliability, efficiency, security, safety, maintainability and cost effectiveness" [code&conduct]

3 ANALYSIS

This section describes the requirements for the scheduling system to be implemented, generating a timetable for faculty staff and students.

The formal computing approach described here will be used to model the currently informal process carried out to negotiate dates for project meetings between students and project supervisors. As the process is presently informal, the workflow currently used will be a template for the resulting computer system, and will be largely user-driven in terms of selecting dates and input of project data.

3.1 Problem specification

3.1.1 General explanation

The objective or main task is to find an optimal timetable (see definition below), if a solution is feasible (possible to be computed/generated) with the respect to the given constraints. All hard and soft constraints are listed in the requirements.

Definition of optimal timetable:
An optimal timetable is a timetable that fulfils all hard constraints and as many soft constraints as possible when a solution is feasible.

Definition of timetable:
A timetable is a set of time slots representing a particular time frame where events are allocated by meeting all of the constraints.

Definition of event:
In terms of this project, an event is a student presentation that takes a certain amount of time (e.g. 20 minutes), conducted in a university room available at that time. The event is attended by one student (student presenting a final project), supervisor of the project and a faculty member assigned to the project as a second examiner.

Definition of faculty member project load:
A faculty member project load is an ability to supervise or mark the maximum amount of student projects as a second examiner.

Another task is to assign a faculty member as a second examiner to a student's final project presentation. Second examiners are selected on their project loads (see the definition above). A special set of projects, for example music informatics projects, are assigned second examiners only from music informatics faculty members. The same rule applies to multimedia projects. Only music informatics projects are conducted in a music lab.

The system relies on the data imported from the already existing system such as information regarding students, their projects and faculty. This data can be imported from the existing system and university timetable, before faculty start inserting their own time constraints.

The format of the timetable will be calendar based and can be emailed to faculty and students in a readable format or a URL link will be provided.

Personal timetable is a particular timetable, whose contents apply only to particular person (faculty or students).

3.1.2Hard Constraints

See the definition of hard constraints in the introduction of this report.

Definition of time constraints:
A time constraint stands for a time that faculty staff/student is physically able to attend the student's presentation.

Definition of room time constraints:
A room time constraints specifies when university rooms are available for students' presentations.

All faculty, student or room time constraints are hard constraints, among others stated below.

List of hard constraints:

- Faculty member and second marker can only be physically involved in one project at one particular time.
- Faculty member cannot be assigned to the project he supervises as a second examiner.
- Second examiners are assigned to student projects on the basis of loads (see definition of load in the section 3.1.1)
- Faculty, student and room time constraints (see definition above)
- Timeslot is 20 minutes long.

3.1.3 Soft constraints

Definition:
Soft constraints are constraints that do not have to be fulfilled to consider a generated timetable as valid. These constraints are optional, and help to determine an optimal solution (see definition of optimal solution in the introduction)

All soft constraints can be considered as parameters that can be set up by the administrator.

List of soft constraints:

- For music and video projects, music and video faculty will be assigned as 2nd markers preferable
- Presentations will preferably be in chunks
- Time between 12 to 2pm is reserved for lunch
- Usually there are up to 4 days for presentations. Sometimes an extra day (spill over day) is needed when some member of faculty is away for the entire week. The system detects if faculty are away for the entire time period for which the presentations are scheduled (so that one can install an extra day).
- For Music Informatics second markers are problematic (this is because there are not many faculty members who can mark those as second examiners/markers). Music Informatics projects are demonstrated in the MI Lab Arun 221. This is used as a teaching Lab in the summer so its booking times have to be considered for the scheduling. Ideally this is again taken from the university online timetable.

3.2 Functional requirements

Functional requirements determine what the system can do, as well as the input and output of the system.

3.2.1 Administrator functionalities

Administrator can interact with the system as follows:

- Insert, delete:
  - Room available for a student presentation
  - Time constraints (see section 3.1.2)
  - Soft constrains
- Generate second examiners based on their loads (see section 3.1.1)
- Generate an optimal timetable (see section 3.1.1)
- Amend a generated timetable by using graphical user interface. An interactive user-friendly interface will be provided with extra functionality for swapping time slots (see section 3.2.2). This function will be provided so that when a user double clicks on a slot, it will highlight in green the other possible available time slots that can also be used.
- Import or use data of old system including university timetable, all information regarding students and faculty members
- Email or print a timetable for faculty members and students. Administrator can email a link to the system, therefore students or faculty members can see their personal timetable or timetable including all presentations. As describe in the previous section, the timetable is in calendar-based format
- Administrator inherits all faculty staff's functionalities.

3.2.2Faculty staff functionalities

Faculty Staff can interact with the system as following:

- Display a personal timetable.
- Display a timetable including all presentations
- Insert/delete their time constraints by using a graphical user interface

3.2.3 Student functionalities

Students can interact with the system as following:

- Display a personal timetable.
- Display a timetable including all presentations

3.3 Extended Functionalities

In order to generate the most optimal timetable (see section 3.1.1), other extra functionalities related to solving this constraint satisfaction problem (CSP) are provided to the administrator. One of the extra functionalities is to limit identical pairs of a student project's supervisor and a second examiner marking the final project, or vice versa. The system is extended to meet this extra requirement in the way that this feature is only available to the system administrator who can set up explicitly the limit of such pairs (default value of limit is 3 as discussed with the customer). For instance, if faculty members X and Y are involved in a project presentation as a supervisor or second marker/examiner or vice versa, they are both involved in the same project presentation at maximum another 2 times. In the analysis phase of the developing process, this extra requirement was not captured, but it was recognised at the early stage of the developing process after discussion with the supervisor and the customer.

Another extended functionality of the system is to recognize faculty members who are in some way involved in music informatics project presentation either as a supervisor or second marker/examiner, and provide them a sufficient time for relocation from the music lab to other places where the projects are presented. In this particular case, the assumption of time required for the relocation was made by the informatics department office and is equal to the length of one time slot (20 minutes) that should be adequate.

Also the system will need to be extended by an extra feature, helping to identify time slots which are available to swap for a particular slot. In other words, if anything unexpected happens, as we are not living in the ideal world, two time

slots can be dynamically swapped. To make it easier and prevent the administrator from making errors, all time slots that are safe to be swapped are marked with a different colour.

The last extra functionality enables the administrator to assign explicitly a second examiner to a particular project or all projects of a particular supervisor. Also a room can be assigned to a specific programme, for instance a music lab to music informatics projects.

3.4 Non-functional requirements

The system will be installed on the informatics department's Apache web server and will include a web interface to interact with users.

The timetable needs to be generated in reasonable time. The maximum time allowed for this task is one week, but preferably should take less than a day.

The administrator will set the deadline for inserting time constraints, therefore afterwards, inserting is disabled and no time constraint can be inserted into the system. Only the administrator can insert time constraints after the deadline and thus more alternatives arise on how the system would cope with this situation. One of the alternatives will be to run the system to generate the whole timetable again, or to just amend the timetable locally in case that generating the timetable is not affordable due to time limitation. After discussing the issue with the customer, the acceptable time for producing the timetable is approximately one day, but as the process runs a few times, the minor extension to the time is acceptable.

Non-functional requirements of interface:

- Easily usable by all university administrators and faculty
- Easy to insert time constraints for faculty
- Entering data to be quick
- Graphical user interface is calendar-based

3.5 Use cases

3.5.1 Use case diagram

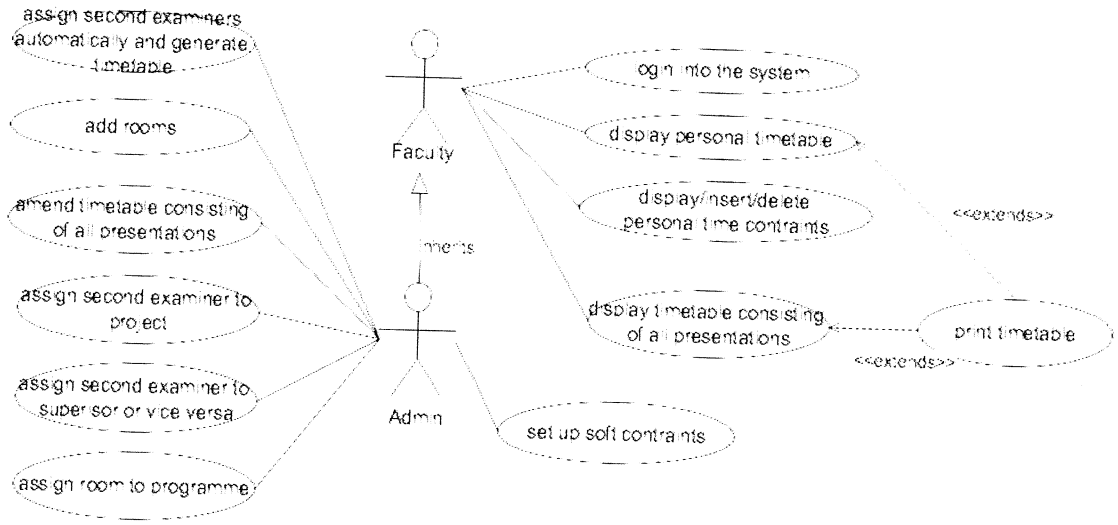

Figure 1 – Use case diagram

3.5.2 Use Case UC1: Login into the system

Primary Actor: Any user
Success guarantees: User is logged into the system.

Main Success Scenario (or Basic Flow):

1. User (Faculty, Administrator or student) provides login details.
2. The user is authenticated.
3. Homepage is displayed and the message to the user that the login has been successful.

Extensions (or alternative flows):

a* At any time system fails, user is informed about the error and the user can repeat the action.

1. a) Login details are invalid:
    1. Login details are validated and the details are detected to be invalid.
    2. The user is informed that login failed.

2. a) Process of login failed:
    1. An error is detected during the process
    2. The user is informed that a process failed

3.5.3Use Case UC2: Add personal time constraints

Primary Actor: Faculty member
Preconditions: User is identified and authenticated.
Success guarantees: Personal time constraints are saved.

Main Success Scenario (or Basic Flow):

1. Faculty member adds a time constraint by using calendar-based GUI.
2. The constraints are displayed.
3. User submits constraints.
4. The constraints are saved.
5. The message is shown to user that constraints have been saved successfully.

Extensions (or alternative flows):

a* At any time system fails, user is informed about the error and the user can repeat the action.

3. a) Constraint already exists:
    1. If GUI allows, insert the same time constraint, the constraint is validated
    2. System informs a user that the constraint was not inserted

4. a) Process of saving the constraints failed:
    1. An error is detected during the process
    2. The user is informed that a process failed
    3. The constraints are displayed

Special requirements
The graphic user interface is supposed to be user-friendly and easy to interact with. Adding constraints is required to be quick and straightforward and displaying data (constraints) clear to user.

3.5.4 Use Case UC3: Delete personal time constraints

Primary Actor: Faculty member
Preconditions: User is identified and authenticated.
Success guarantees: Personal time constraints are deleted.

Main Success Scenario (or Basic Flow):

1. Faculty member selects a time constraint to delete.
2. The constraint is marked and displayed to delete.
3. The user submits constraint to delete.
4. The constraint is deleted.
5. The message is shown to user that constraint has been deleted successfully.

Extensions (or alternative flows):

a* At any time system fails, user is informed about the error and the user can repeat the action.

3. a) Constraint cannot be deleted:
    1. Constraints are prevented from being deleted by user
    2. The user is informed that constraint is not deleted

    b) Process of deleting the constraint failed:
    1. An error is detected during the process
    2. The user is informed that the process failed
    3. The constraints are displayed

Special requirements
The graphic user interface is supposed to be user-friendly and easy to interact with. Deleting constraints is required to be quick and straightforward and displaying data (constraints) clear to the user.

3.5.5 Use Case UC4: Assign second examiners to projects

Primary Actor: Administrator
Success guarantees: Second examiner is assigned to the project.

Main Success Scenario (or Basic Flow):

1. Administrator selects an option to assign second examiners to projects.
2. GUI providing to accomplish this task is displayed to administrator.
3. Administrator, using the interface, assigns a second examiner to project.
4. Changes are automatically saved after every action/interaction.
5. The message is shown to user that the timetable has been saved successfully.

Extensions (or alternative flows):

a* At any time system fails, user is informed about the error and the user can repeat the action.

4. a) Process of assigning the second examiner failed:
    1. An error is detected during the process
    2. The user is informed that the process failed

Special requirements
The graphic user interface is supposed to be user-friendly and easy to interact with. Assigning is required to be quick and straightforward and displaying data is clear to the user.

3.5.6 Use Case UC5: Assign second examiners to supervisors and vice versa

Primary Actor: Administrator
Success guarantees: Second examiner or supervisor is assigned successfully.

Main Success Scenario (or Basic Flow):

1. Administrator selects an option to assign second examiners to supervisors.
2. GUI providing to accomplish this task is displayed to administrator.
3. Administrator, by using the interface, assigns a second examiner to the supervisor or vice versa.
4. Changes are automatically saved after every action/submit.
5. The message is shown to user that the timetable has been saved successfully.

Extensions (or alternative flows):

a* At any time system fails, user is informed about the error and the user can repeat the action.

4. a) Process of assigning the second examiner or supervisor failed:
    1. An error is detected during the process
    2. The user is informed that the process failed

Special requirements
The graphic user interface is supposed to be user-friendly and easy to interact with. Assigning is required to be quick and straightforward and displaying data clear to user.

3.5.7 Use Case UC6: Assign room to programme

Primary Actor: Administrator
Success guarantees: Room is assigned successfully.

Main Success Scenario (or Basic Flow):

1. Administrator selects an option to assign a room to the programme.
2. GUI providing to accomplish this task is displayed to administrator.
3. Administrator, by using the interface, assigns the room to the programme.
4. Changes are automatically saved after every action/submit.
5. The message is shown to user that the timetable has been saved successfully.

Extensions (or alternative flows):

a* At any time system fails, user is informed about the error and the user can repeat the action.

4. a) Process of assigning the room failed:
    1. An error is detected during the process
    2. The user is informed that the process failed

Special requirements
The graphic user interface is supposed to be user-friendly and easy to interact with. Assigning is required to be quick and straightforward and displaying data clear to user.

3.5.8 Use Case UC7: Display personal timetable

Primary Actor: Faculty, Student, Administrator
Success guarantees: Personal timetable is displayed to user.

Main Success Scenario (or Basic Flow):

1. User (Faculty, Student or Administrator) selects an option to display the personal timetable.
2. The appropriate calendar-based timetable is displayed.
3. User prints the print-friendly version of timetable.

Extensions (or alternative flows):

a* At any time system fails, user is informed about the error and the user can repeat the action.

3.5.9 Use Case UC8: Display timetable

Primary Actor: Faculty, Student, Administrator
Success guarantees: Timetable is displayed to user.

Main Success Scenario (or Basic Flow):

4. User (Faculty, Student or Administrator) selects an option to display the timetable.
5. The appropriate calendar-based timetable is displayed.
6. User prints the print-friendly version of timetable.

Extensions (or alternative flows):

a* At any time system fails, user is informed about the error and the user can repeat the action.

3.5.10 Use Case UC9: Manually edit timetable

Primary Actor: Administrator
Success guarantees: Timetable is updated successfully.

Main Success Scenario (or Basic Flow):

6. Administrator selects an option to edit the timetable
7. The timetable is displayed in an edit mode.
8. User edits the calendar-based timetable by interacting with it.
9. Changes are automatically saved after every action/interaction.
10. The message is shown to user that the timetable has been saved successfully.

Extensions (or alternative flows):

a* At any time system fails, user is informed about the error and the user can repeat the action.

4. a) Process of editing the timetable failed:
    1. An error is detected during the process
    2. The user is informed that the process failed

5. a) In case of a mistake caused by administrator, step back functionality is provided.

Special requirements
The graphic user interface is supposed to be user-friendly and easy to interact with. Editing the timetable is required to be quick and straightforward and displaying data clear to user.

3.5.11 Use Case UC10: Insert room

Primary Actor: Administrator
Preconditions: User is identified and authenticated.
Success guarantees: Room is inserted.

Main Success Scenario (or Basic Flow):

1. Administrator types required information and submits the data.
2. The message is shown to user that the process has been accomplished.

Extensions (or alternative flows):

a* At any time system fails, user is informed about the error and the user can repeat the action.

1. a) Process of inserting the room failed:
    1. The error is detected during the process
    2. The user is informed that a process failed

Special requirements
The graphic user interface is supposed to be user-friendly and easy to interact with. Inserting rooms is required to be quick and straightforward and displaying data clear to user.

3.5.12 Use Case UC11: Delete room

Primary Actor: Administrator
Preconditions: User is identified and authenticated.
Success guarantees: Room is deleted.

Main Success Scenario (or Basic Flow):

1. Administrator selects an option to delete a room.
2. The room is deleted.
3. Message is shown to the user that the process has been accomplished.

Extensions (or alternative flows):

a* At any time system fails, user is informed about the error and the user can repeat the action.

2. a) Process of deleting data failed:
    1. An error is detected during the process
    2. The user is informed that a process failed

Special requirements
The graphic user interface is supposed to be user-friendly and easy to interact with. Deleting a room is required to be quick and straightforward and displaying data clear to user.

3.5.13 Use Case UC12: Assign second examiners automatically and generate timetable

Primary Actor: Administrator
Preconditions: User is identified and authenticated.
Success guarantees: Second markers are assigned and timetable is generated.

Main Success Scenario (or Basic Flow):

1. Administrator selects an option to generate a timetable.
2. The second examiners are assigned and the timetable is generated.
3. The calendar-based timetable is displayed and alternatives if applicable.
4. The message is shown to user that the process has been accomplished.

Extensions (or alternative flows):

a* At any time, system fails: The current state (sessions) is deleted and the system asks for details to login.

1. a) Process of assigning the second examiners or generating timetable failed:
    1. An error is detected during the process
    2. The user is informed that a process failed

4 Design

In this section the requirements from the analysis phase of the software development process will be used to produce a design of this system. Unified Modelling Language (UML) will be used to describe the design in a clear and readable form.

Section 4.1 is the Overall structure. It informs about the architecture of the system in a high-level form and looks at the system from a general prospective.

Section 4.2 contains a database design. It describes every part of the architecture of the database designed for this system, along with the database of old system. Textual description of each table is also included.

Section 4.3 is written regarding "Backend engine", the application assigning second examiners and producing a timetable. It also describes an external library "choco" and constraint model of the CSP. Class and sequence diagrams of its packages are provided to explain their architectures in detail.

Section 4.4 contains the User interface design. It describes the PHP framework used to implement the interface, why it was used and its general features.

4.1 Overall structure

The whole structure of the system is divided in two main parts:

- "Backend engine", which its main objective is to do all computations related to generating a timetable based on the constraints provided. It relies on the external library named "Choco" (see section 4.2).

- Web-based user interface using the open source project, PHP framework "Symfony" (see section 4.4.1). "Servlet", PHP and AJAX technology are used to build an interactive and user-friendly interface.

Definition of AJAX:
"Ajax, sometimes written as AJAX (shorthand for Asynchronous JavaScript + XML), is a group of interrelated web development techniques used to create interactive web applications or rich Internet applications. With Ajax, web applications can retrieve data from the server asynchronously in the background without interfering with the display and behavior of the existing page. The use of Ajax has led to an increase in interactive animation on web pages.[1][2] Data is retrieved using the XMLHttpRequest object or through the use of Remote Scripting in browsers that do not support it." [ajax def]

Definition of Servlet:
"Java Servlet technology provides Web developers with a simple, consistent mechanism for extending the functionality of a Web server and for accessing existing business systems. A servlet can almost be thought of as an applet that runs on the server side--without a face. Java servlets make many Web applications possible." [servlet def]

Definition of PHP:
"PHP is a widely-used general-purpose scripting language that is especially suited for web development and can be embedded into HTML. It generally runs on a web server, taking PHP code as its input and creating web pages as output. It can be deployed on most web servers and on almost every operating system and platform free of charge." [php def]

4.1.1 Overall design structure diagram

The overall structure of the system is displayed in Figure 2. As mentioned in the previous section, it is divided into two parts, user interface and "backend engine" application. A user makes a request to the system by using a web browser. The request is accepted by the controller, which recognizes a module and action from the request and invokes an appropriate action. All computations and data processing happens in modules, which functions are called by the controller action. The code produced in the view is returned to the user browser (more detailed explanation of an MVC pattern and the framework provided in the following sections). Also the user can make a request to the "Backend engine" by using "AJAX" and "Servlets" (see definitions in previous section) to invoke a method producing a timetable (see section 4.3).

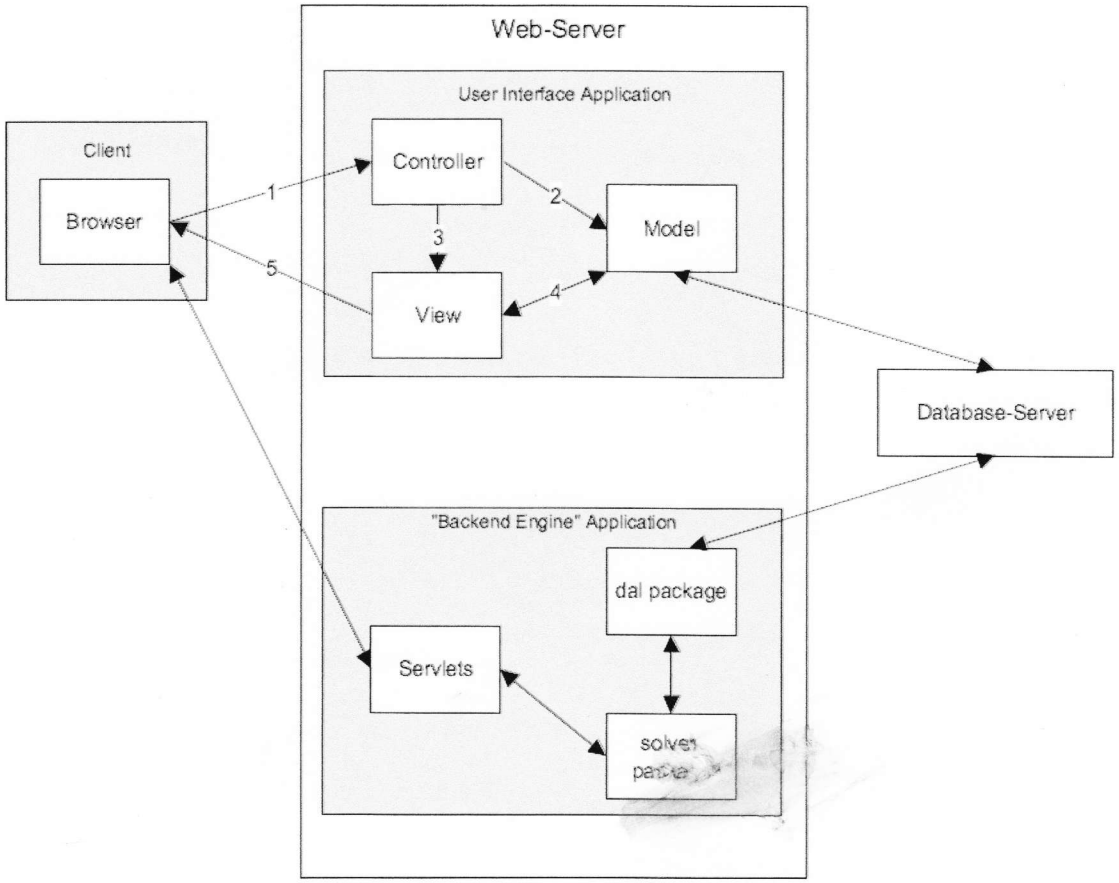

Figure 2. Overall structure diagram of the system

Note: "dal package" - database access layer package

4.2 Database design

The design of database structure for this project relies on the database implemented for the old system, used for managing final year projects. In order to prevent duplications of data the old system database will be used as part of a data source. The old system includes data regarding students, faculty members and projects, which are required to produce a timetable. To see the structure of the database the Enhanced Entity-Relationship (EER) model is provided below. The entities marked grey colours are entities of the old system database. The yellow entities are designed for this system database.

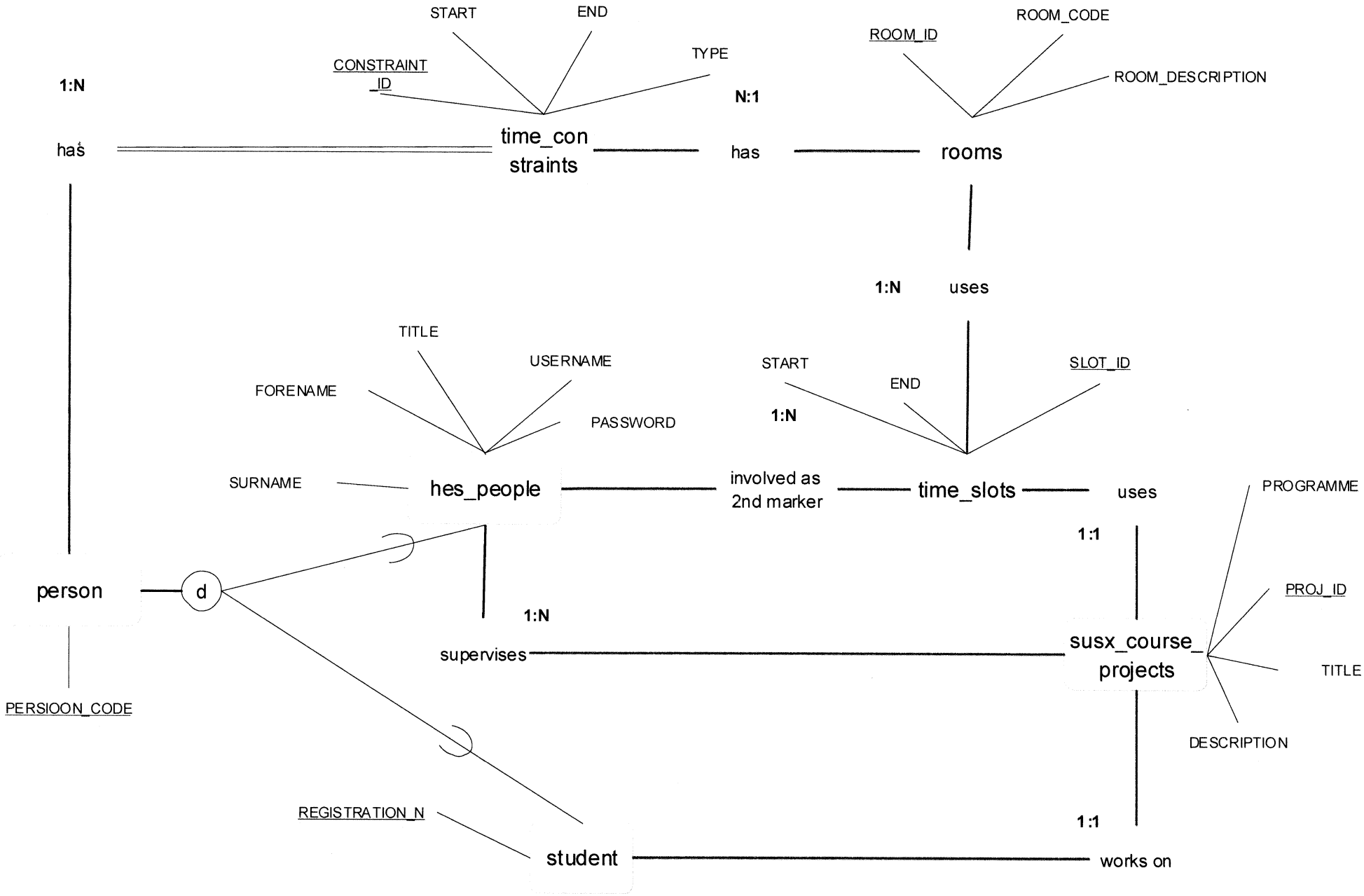

Figure 3. EER model of database (part1)

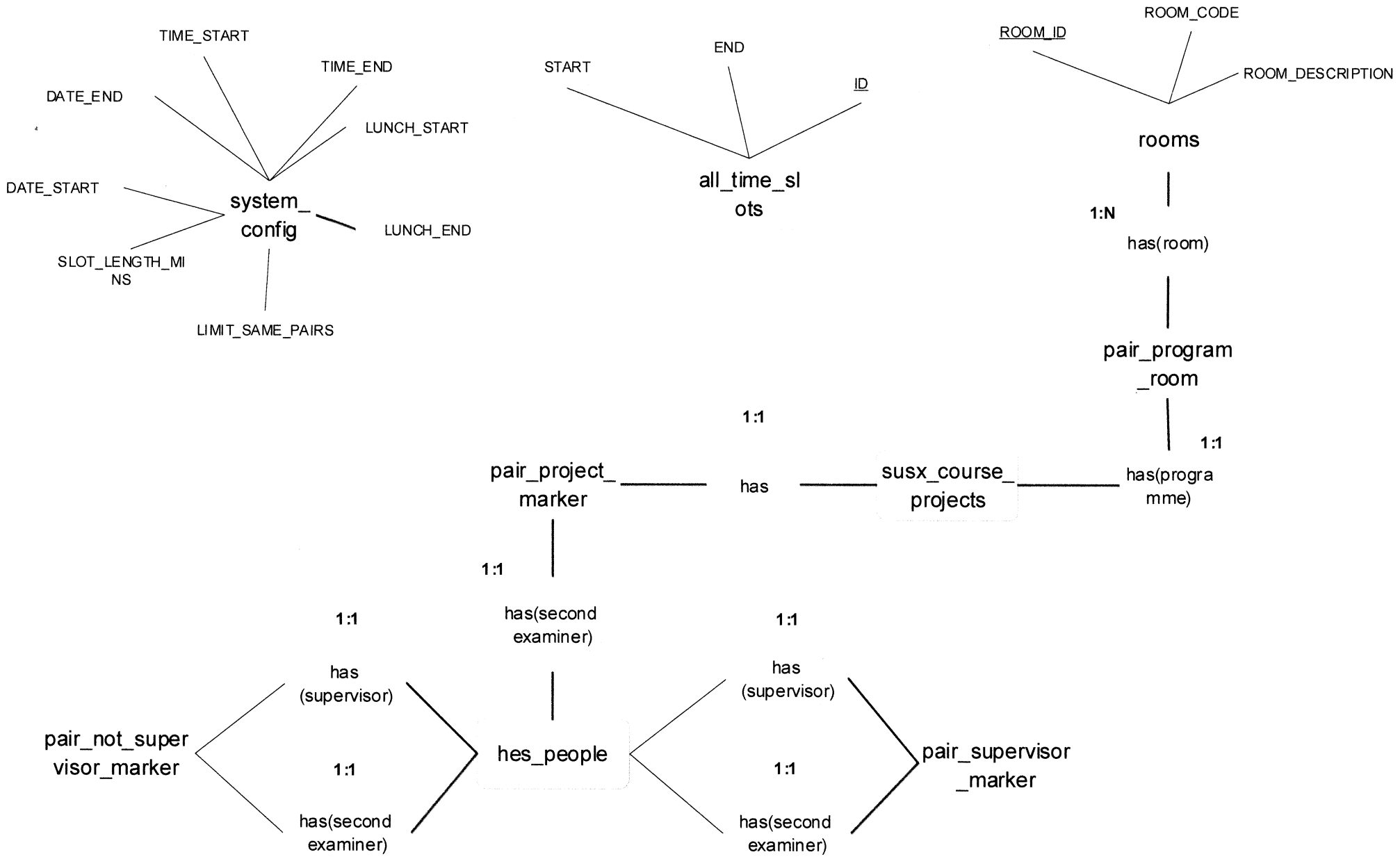

Figure 4. EER model of database (part2)

4.2.1 Description of external database tables (old system)

This section describes the database tables of the old existing system, their structures and columns. The column "Name" describes the names of attributes of database table. The type column corresponds to a type of the attribute. A primary key of table is marked as yellow key.

This table represents faculty members and students including personal data and data required for login.

Name	Type	Length	Decimals	Null
PERSON_CODE	bigint	22	0	
TITLE	varchar	8	0	✓
FORENAME	varchar	30	0	✓
SURNAME	varchar	30	0	
USERNAME	varchar	20	0	
PASSWORD	varchar	100	0	✓

Figure 5. Table "hes_people"

This table includes extra information about students such as their registration numbers and programmes.

Name	Type	Length	Decimals	Null
REGISTRATION_NUMBER	bigint	22	0	
PERSON_CODE	bigint	22	0	✓
PROGRAMME	varchar	30	0	✓

Figure 6. Table "student"

This table includes all information regarding the projects. The attribute "REGISTRATION_NUMBER" links the particular project to the particular student. The attribute "PERSON_CODE" links student projects and their supervisors. The other attributes do not apply to the project.

Name	Type	Length	Decimals	Null
PROJ_ID	bigint	22	0	
REGISTRATION_NUMBER	bigint	22	0	✓
COURSE_CODE	varchar	12	0	
PERSON_CODE	bigint	22	0	✓
START_DATE	varchar	7	0	✓
TOPIC1	varchar	50	0	
TOPIC2	varchar	50	0	✓
TITLE	varchar	60	0	
STATUS	varchar	8	0	
DESCRIPTION	blob	0	0	
REQ_SKILLS	varchar	100	0	✓
AUTHOR_PROGRAMME	varchar	22	0	✓
NOTES	blob	0	0	✓

Figure 7. Table "susx_course_projects"

This table includes information about loads of faculty members required to assign second examiners correctly to student projects based on these loads.

Name	Type	Length	Decimals	Null
COURSE_CODE	varchar	12	0	
CONTACT_TYPE	varchar	30	0	
PERSON_CODE	bigint	22	0	
PROJ_LOAD	int	2	0	✓
TAKE_STUDENTS	char	3	0	

Figure 8. Table "projs_sv_list"

4.2.2 Description of database tables of this system

This section includes the same type of information as previous, apart from the fact that the tables of the database designed particularly for this project are described here.

This table includes information about the availability of students, faculty members and rooms. The attribute "TYPE" determines that either it is related to a student, faculty member or room. These particular types are stored in the lookup table "type_lookup" (see table "type_lookup" below). The attribute "TARGET_ID" is an ID/primary key one of "hes_people", "student" or "room" table. The attributes "START" and "END" determines a time when one of the previous mentioned objects are **NOT** available.

Name	Type	Length	Decimals	Null
CONSTRAINT_ID	int	11	0	
TYPE	int	11	0	
TARGET_ID	int	11	0	
START	datetime	0	0	
END	datetime	0	0	

Figure 9. Table "time_constraints"

This table includes information regarding rooms where the student project will be presented.

Name	Type	Length	Decimals	Null
ROOM_ID	int	11	0	
ROOM_CODE	varchar	20	0	✓
ROOM_DESCRIPTION	varchar	100	0	✓

Figure 10. Table "rooms"

This table includes all information that is used for setting a timetable scheduler in order to produce a valid timetable. Column "DATE_START" and "DATE_END" specify when is the first and last day of student presentations, respectively.

Name	Type	Length	Decimals	Null
DATE_START	date	0	0	✓
DATE_END	date	0	0	✓
TIME_START	time	0	0	✓
TIME_END	time	0	0	✓
LUNCH_START	time	0	0	✓
LUNCH_END	time	0	0	✓
SLOT_LENGTH_MINS	int	11	0	✓
LIMIT_SAME_PAIRS	int	11	0	✓

Figure 11. Table "system_config"

This table includes information about time slots. The timetable consists of time slots, which are used to build the whole timetable. The time slot includes information regarding a student project (attribute "PROJ_ID"), faculty members who mark it (attribute "2ND_MARKER"), room (attribute "ROOM_CODE") where the project is presented and time when the time slot begin and end (attribute "BEGIN", "END" respectively).

Name	Type	Length	Decimals	Null
SLOT_ID	int	11	0	
2ND_MARKER	int	11	0	✓
PROJ_ID	int	11	0	✓
ROOM_CODE	varchar	10	0	✓
START	datetime	0	0	
END	datetime	0	0	

Figure 12. Table "time_slots"

This table represents all time slots available to be scheduled. They are generated on the basis of information held in "system_config" table.

Name	Type	Length	Decimals	Null
ID	bigint	20	0	
START	datetime	0	0	✓
END	datetime	0	0	✓

Figure 13. Table "all_time_slots"

This table includes pairs of project ids and person code of second examiners. The second examiners/markers must be assigned to the projects as specified in this table.

Name	Type	Length	Decimals	Null
PROJ_ID	int	11	0	
SUPERVISOR_CODE	bigint	22	0	✓
SECOND_MARKER_CODE	bigint	22	0	✓

Figure 14. Table "pair_project_marker"

This table includes pairs of project ids and person code of second examiners. The second examiners/markers must be assigned to the projects as specified in this table.

Name	Type	Length	Decimals	Null
SUPERVISOR_CODE	bigint	22	0	
SECOND_MARKER_CODE	bigint	22	0	

Figure 15. Table "pair_supervisor_marker"

This table includes pairs of supervisor person codes and person codes of second examiner. The second examiners/markers must be assigned to all the projects supervised by the supervisor as specified in the table.

Name	Type	Length	Decimals	Null
SUPERVISOR_CODE	bigint	22	0	
SECOND_MARKER_CODE	bigint	22	0	

Figure 16. Table "pair_not_supervisor_marker"

This table includes pairs of project programmes and room ids. Available rooms are assigned to projects belonging to the particular programme. These projects can be presented in the room assigned to them.

Name	Type	Length	Decimals	Null
AUTHOR_PROGRAMME	varchar	22	0	
ROOM_ID	int	11	0	

Figure 17. Table "pair_programme_room"

4.3 Backend Engine

All tasks related to producing a timetable and assigning second examiners to student projects are done by the backend engine. It consists of two packages ("dal" and "solver" package) described in detail in the following sections. It includes also "servlets" (see definition in section 4.1) to enable a user to communicate with the backend engine by using an Internet browser.

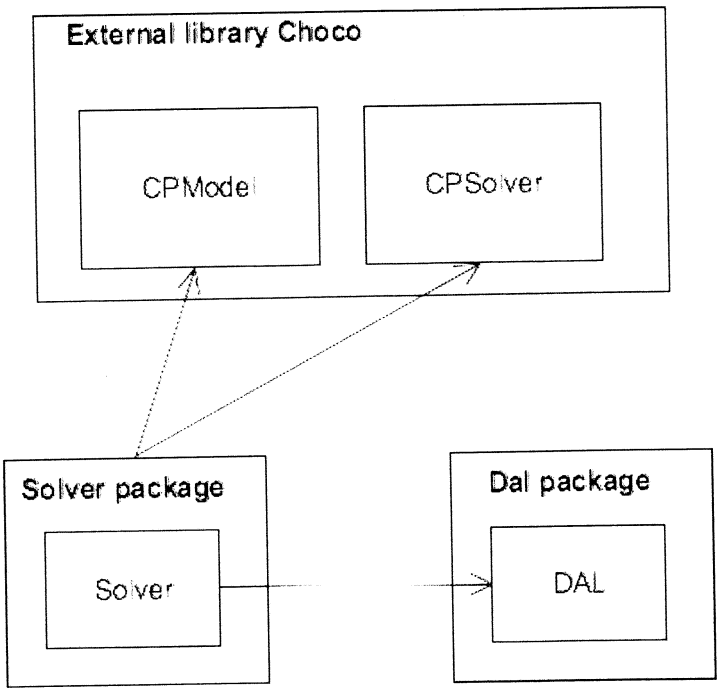

Figure 18. High-level backend engine package diagram

4.3.1 External Library "choco"

This section describes the use of the external library named "Choco" used to solve the Constraint Satisfaction Problem (CSP), particularly in this project a timetabling problem. The "Solver" package relies on this library.

"choco is a java library for constraint satisfaction problems (CSP) and constraint programming (CP). It is built on an event-based propagation mechanism with back-trackable structures. choco is open- source software, distributed under a BSD licence and hosted by source- forge.net. choco is mainly developed by people at École des Mines de Nantes (France) and is financially supported by Bouygues SA and Amadeus SA." [choco white paper, p.1]

This library consists of two main parts: a problem modeller and problem constraint solver. The more detailed descriptions of both parts are stated below.

A problem modeller is a library that models CSP. In order to solve CSP, all variables, their constraints and domains must be specified by using the API of this library. "Choco" provides various types of variables such as integer, set, real and expression variable. In this project, integer and expression variables are used. Also a wide range of constraints are available to model a problem. It provides over 70 types of constraints including logic operators, which are sufficient to model this particular problem.

This part of the library is a constraint solver. "choco can either be used in satisfaction mode (computing one solution, all solutions or iterating them) or in optimization mode (maximisation and minimisation). Search can be parameterized using a set of pre-defined variable and value selection heuristics." [choco white paper, p.3]

"choco is a Java library that chose to provide a clear separation between modelling and solving. Figure 3 represents the overall architecture of the choco library. There are two separate parts:

- The first part (from the user's point of view) is devoted to expressing the problem. The idea is to manipulate variables and relations to be verified for these variables (constraints) disregarding their potential implementation (either from the variable point of view or the constraint point of view). A complete API is provided to be able to state a problem in a way as user- friendly as possible.
- The second part is devoted to actually solve the problem. In Figure 1, only CP related information is provided. Solving includes specific memory management for tree-based search (as in CP)." [choco white paper, p.4]

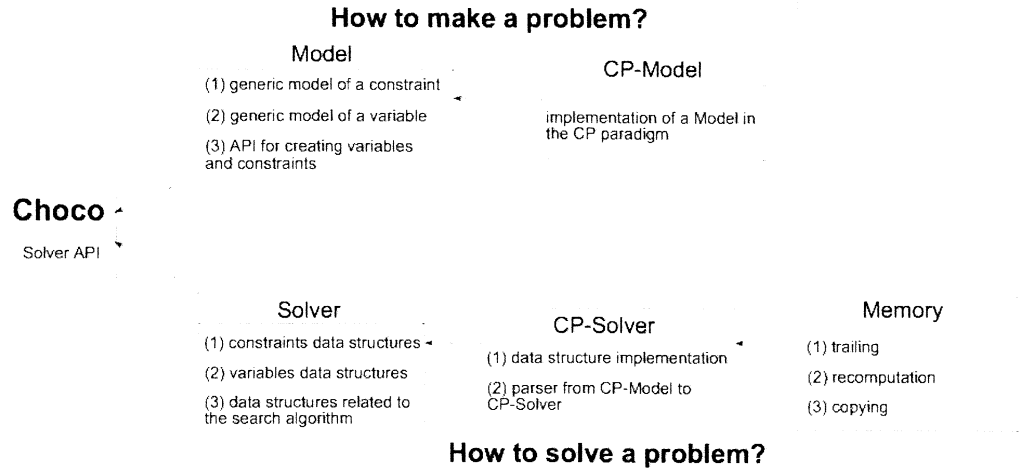

Figure 19. *"choco 's general architecture. The separate parts are clearly identified: a modeling part for stating the problem and a solving part (here only the CP related information is described) for actually solving the modeled problem."[choco white paper]*

4.3.2 Constraint model of problem

In this section the logic of the constraint model is discussed in more detail.

Two sets of variables are used to model the CSP.

List of variables of the model:

- Variable **"project slot"** represents a time slot assigned to a student project. Project ids are values of domains of these variables
- Variable **"room"** represents a room where a student project is presented. Room codes are values of domains of these variables.

Domains of variables are sets of values, which fulfil all constraints.

List of domains of the particular variable:

- Domains of variables "**project slot**" are sets of all time slots determining time frames when students, their supervisors and second markers are available.
- Domains of variables "**room**" are sets of all rooms that are available for projects presentations.

List of all constraints of the model:

- Informal:
  The same time slot cannot be assigned to the different project where their supervisor is an identical faculty member.

  Formal:
  project slot [i] $\neq$ project slot [j] AND
  supervisor (project slot[i]) = supervisor(project slot[j])
  where $0 < i < n$, $0 < j < n$, $i \neq j$, n is equal to the number of projects
  (see descriptions of indexes j and j at the end of this section)

- Informal:
  The same time slot cannot be assigned to the different project where their second marker is an identical faculty member.

  Formal:
  project slot[i] $\neq$ project slot[j] AND

second examiner[project slot(i)] = second examiner [project slot(j)]
where $0 < i < n$, $0 < j < n$, $i \neq j$, n is equal to the number of projects

- Informal:
  The same time slot cannot be assigned to the different project where supervisor of one project is a second marker of another project or vice versa.

  Formal:
  project slot [i] ≠ project slot[j] AND
  supervisor (project slot[i]) = second examiner [project slot (j)]
  where $0 < i < n$, $0 < j < n$, $i \neq j$, n is equal to the number of projects

- Informal:
  The same time slot can be assigned to the different projects if different rooms are assigned to them, otherwise different time slots are assigned to the projects.

  Formal:
  project slot [i] = project slot [j] AND room [i] ≠ room [j] OR
  project slot [i] ≠ project slot [j]
  where $0 < i < n$, $0 < j < n$, $i \neq j$, n is equal to the number of projects

- Informal:
  Another extended requirement is to provide a sufficient time for relocation of faculty members involved in marking of music projects. As time slots are represented as integer numbers in ascending order (earlier time, lower number), it is easy to recognise which time slot is before and after a particular time slot.

  Formal (pseudo code):
  For each music project
  Get its supervisor and second examiner
  Set the constraints that slots with id less or more by one, these cannot be assigned to the project which the supervisor and second marker are involved in.

List of constraints required during assigning second examiners to student projects:

- Informal:
  Faculty members can mark projects as second markers/examiners. The limited times will be derived from their loads and number of projects they supervise. The loads for second examiners are computed before they are assigned to the student projects.

Formal:

maxOccurence(second examiner [project slot (j)) = computed value based on loads
where $0 < i < n$, n is equal to the number of projects

- Informal:

  One of the extended requirements is to limit the same pair of faculty members involved in marking as a supervisor and second examiner or vice versa (see section extended requirements)

  Formal:

  maxOccurence(pair [i]) = value specified by administrator
  where $0 < i < n$, n is equal to the number of pairs containing of one supervisor and second examiner

The indexes i and j of project slots are indexes of an array of size n (number of projects). The mapping table is used to assign one index to one particular project id.
For example:

Index	Project id
0	512
1	564
2	600
3	614
4	654
5	678
6	723

4.3.3 Solver Package

The main objective of this package is to model the CSP problem by using the external "choco" library. Another important function is to assign second examiners to student projects.

Solver package

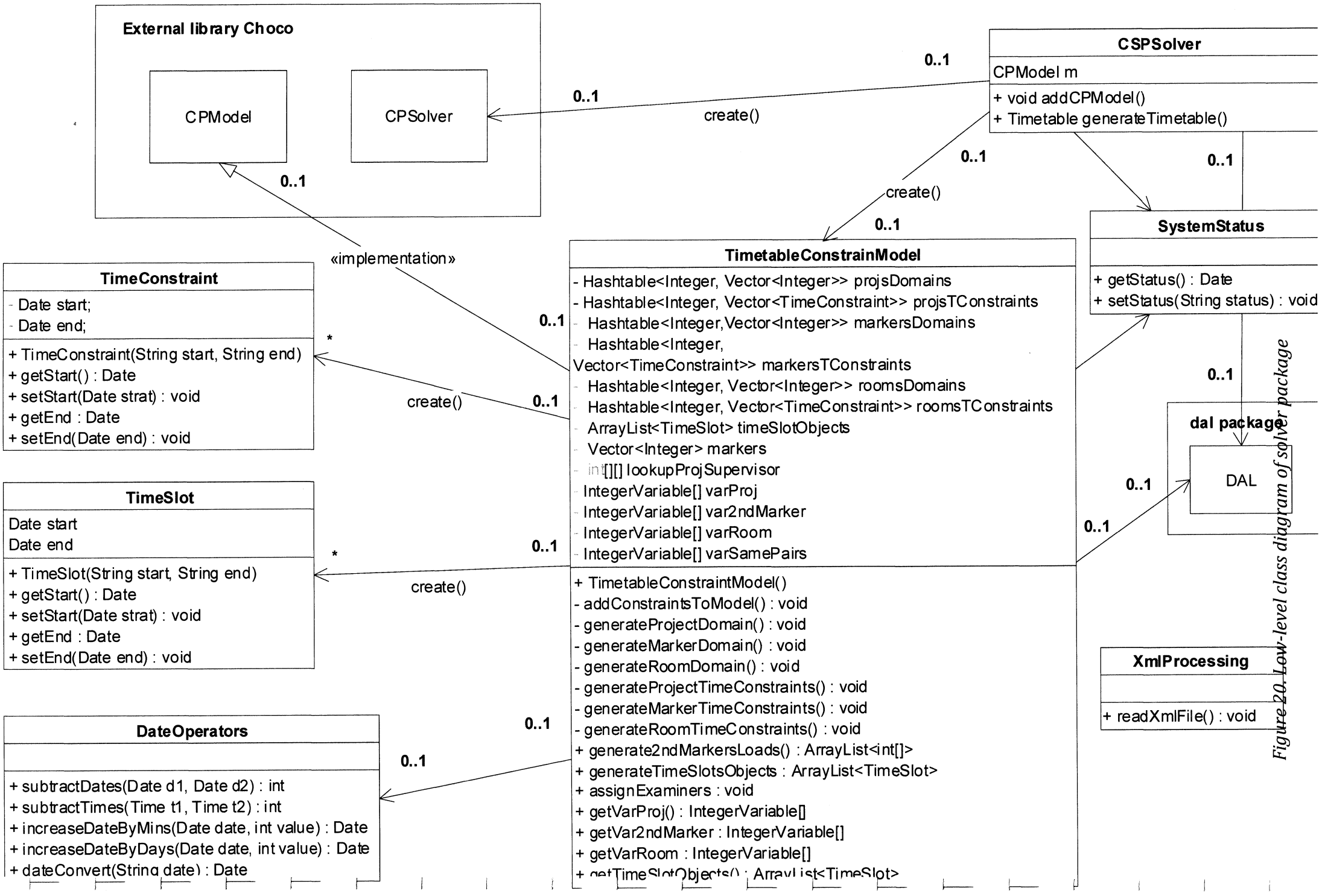

Figure 20. Low-level class diagram of solver package

This diagram shows how the objects within solver package interact from a time prospective. "CSPSolver" object, created by "servlet" method, creates the "TimetableCostraintModel" object. This object invokes its methods to generate time slots, time constraints, domains of variables, second examiners and other constraints (see section 4.3.2.3). During this process "TimeConstraint" and "TimeSlot" objects are created and stored in the "TimetableConstraintModel" object. A result of every action is set as a status of the system by the "SystemStatus" class.

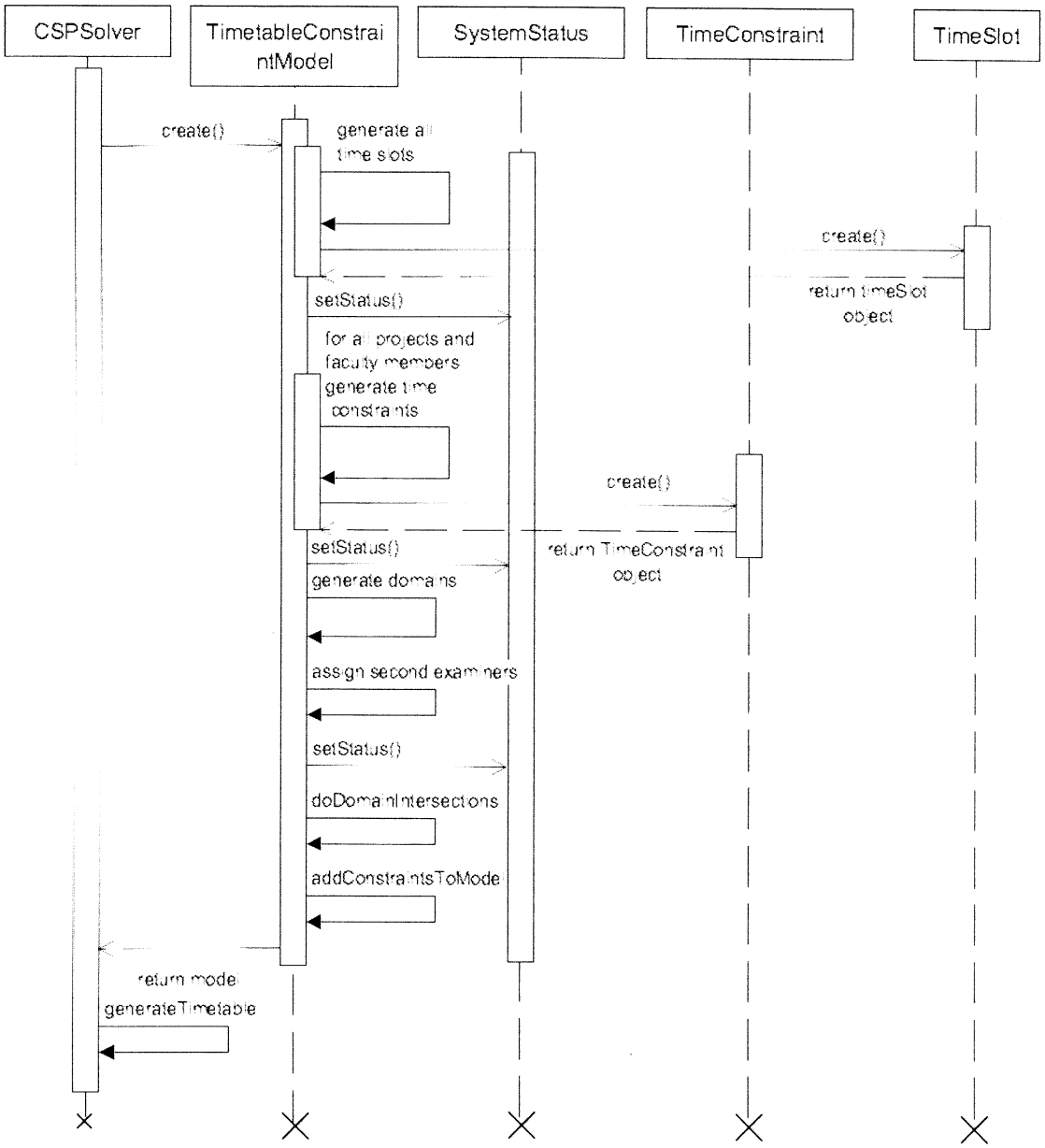

Figure 21. Low-level sequence diagram (look at classes interactions from time perspective)

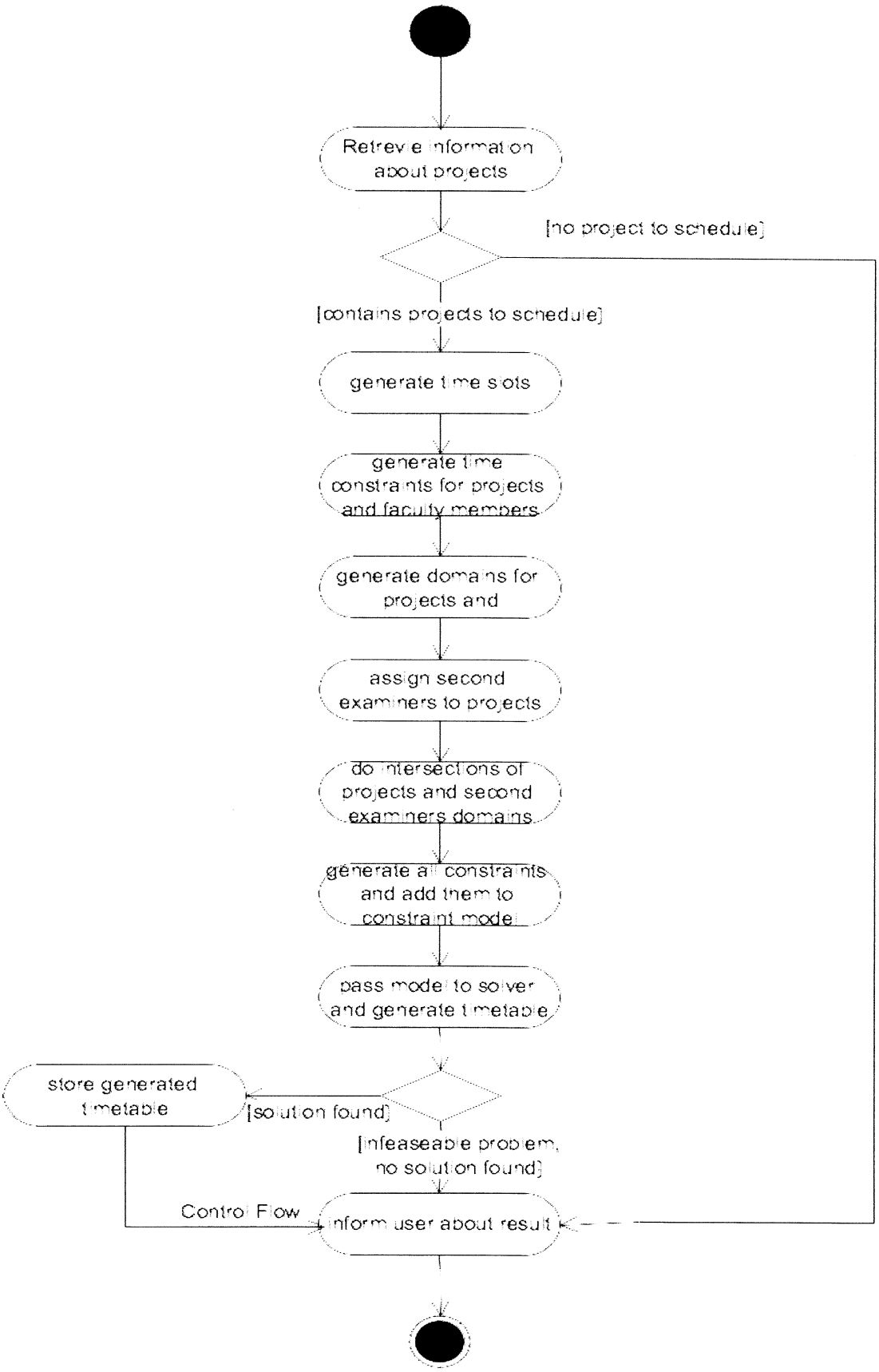

Figure 22. Activity diagram describing the activity during generating a timetable

In this section, descriptions of classes of solver package are provided along with explanations of their most important methods.

4.3.3.3.3 Class "TimetableConstraintModel"

This class extends "CPModel" class of "choco" library and represents the constraint model. Within this class, all variables, constraints and domains are generated to create a constraint model of CSP. The model is used by "CSPSolver" object to find an optimal timetable.

Key methods:

addConstraintsToModel

> Implements all constraints (section 4.3.2.3) of the CSP. It is executed after all variables, time constraints and domains are generated by other methods.

generateProjectDomain

> Generates projects domains based on projects time constraints. It compares projects' time constraints (including supervisors and students' time constraints) against time slots available for student presentations and only ids of time slots fulfilling the constraints are assigned to the domain.

generateRoomDomain

> The same approach is taken as in the method described above. It compares rooms' constraints against available time slots.

generateProjectTimeConstraints

> Retrieves all the information regarding time constraints from the database by using DAL and creates "timeConstraint" objects stored in a vector. These object are used in generating domains.

generateRoomTimeConstraints

> The same approach is used as the method described above.

generate2ndMarkersLoads

> Retrieves information about loads of faculty members by using DAL and computes loads of second examiners.

generateTimeSlotsObjects

Generates all time slot objects based on settings made by the administrator.

assignSecondExaminers

Assigns second examiners to student projects. All second examiners selected for particular projects by administrator are assigned first. Afterwards, all second markers selected to be assigned to particular supervisors are processed before second markers are assigned automatically. Within this class the method "generate2ndMarkersLoads" is invoked to get loads of second examiners in order to assign appropriate amount of projects to them.

doDomainIntersections

Does intersections of domains related to project variables. It is invoked after the second examiners are assigned to the projects. It compares all values of a supervisor and second examiner domain and does an intersection of these two sets. The intersection is a new domain for the project variable.

4.3.3.3.2 Class "CSPSolver"

This class implements "CPSolver" class of "choco" library. It uses the constraint model to find a solution. Within this class, a searching strategy can be set up. To speed up the process of searching the solution, values of project domains are selected on a random basis. A variable with domain including the lowest number of values is assigned the value first, and then other variables in an ascending order.

4.3.3.3.3 Class "TimeConstraint"

This class represents a time constraint. It includes information when student, faculty member or room is not available.

4.3.3.3.4 Class "TimeSlot"

This class represents a time slot. It includes information such as the start and end of a time slot.

4.3.3.3.5 Class "SystemStatus"

This class stores a system status as a string to a file. Therefore, during the generating of a timetable the status can be displayed at user interface by AJAX requests to "servlet" (see definitions in section 4.1), which reads the file and returns the status as string.

4.3.4 Database access layer (DAL) Package

The functionality of this package is to query a database and return desired data back to objects, which invoke the method. As stated earlier, the system relies on the existing database and a new database designed particularly for this project. Therefore, this package is separated in two parts independent of each other, dealing with different databases. This approach provides an opportunity to have each database located on different database servers.

dal package

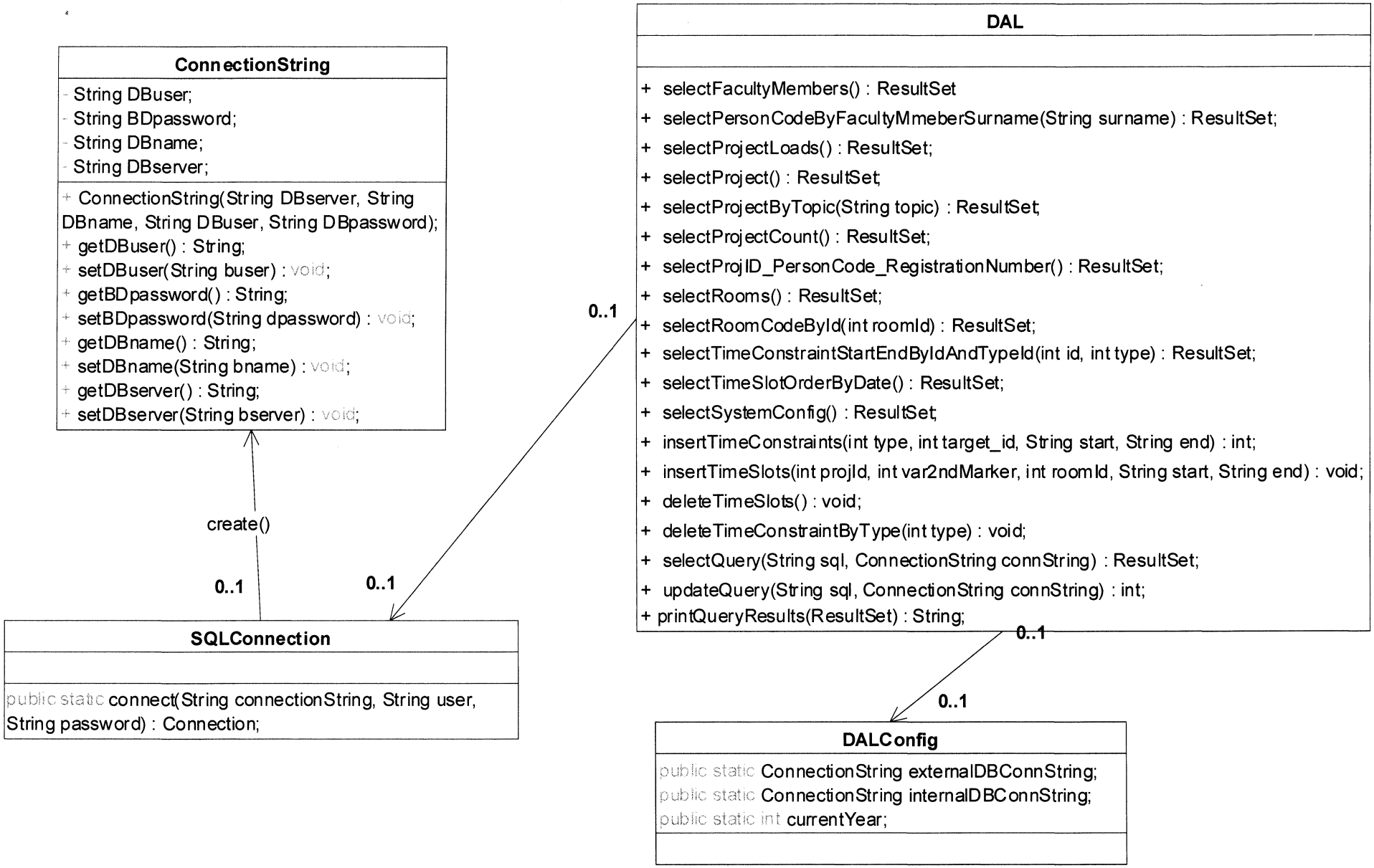

Figure 23. Class diagram of DAL package

This section explains objectives of the class located in DAL package.

4.3.4.2.1 ConnectionString

This class represents a connection string, a string containing all information in order to connect to database successfully. The string includes information such as host name, name of database, username and password.

4.3.4.2.2 SQLConnection

This class makes a SQL connection to database server, and handles errors occurred during the connection.

4.3.4.2.3 DAL

This class has methods containing all SQL queries. The names of methods reveal their functionalities.

4.3.4.2.4 DALConfig

This class contains actual data used to make a connection to the database.

4.4 User interface

This section describes the user interface. There are three main groups of users: students, faculty members and administrators. After successful authentication, the appropriate user interface is displayed. Students can only display timetables, either personal or one containing all slots. AJAX is used to make the interface more interactive. The whole interface is based on the "Symfony" PHP framework (see section below).

4.4.1 Symfony PHP framework

"A framework streamlines application development by automating many of the patterns employed for a given purpose. A framework also adds structure to the code, prompting the developer to write better, more readable, and more maintainable code. Ultimately, a framework makes programming easier, since it packages complex operations into simple statements. Symfony is a complete framework designed to optimize the development of web applications by way of several key features." [Potencier, p.10]

"Symfony is based on the classic web design pattern known as the MVC architecture, which consists of three levels:

- The Model represents the information on which the application operates - its business logic.
- The View renders the model into a web page suitable for interaction with the user.
- The Controller responds to user actions and invokes changes on the model or view as appropriate.

The MVC architecture separates the business logic (model) and the presentation (view), resulting in greater maintainability. For instance, if your application should run on both standard web browsers and handheld devices, you just need a new view; you can keep the original controller and model. The controller helps to hide the detail of the protocol used for the request (HTTP, console mode, mail, and so on) from the model and the view. And the model abstracts the logic of the data, which makes the view and the action in dependent of, for instance, the type of database used by the application." [Potencier, p.19]

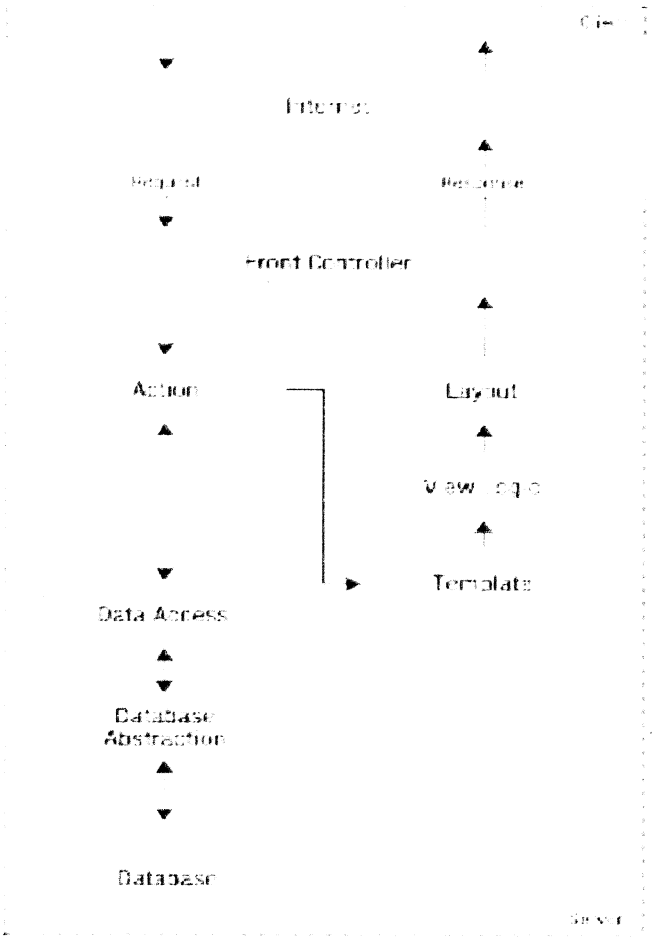

Figure 24. Diagram showing the architecture of MVC pattern in Symfony [Potencier, p.27]

These are the directories found at the root of a symfony project:

```
apps/
   my application/
cache/
config/
data/
   sql/
doc/
lib/
   model/
log/
plugins/
test/
web/
   css/
   images/
   js/
   uploads/
```

Directory	Description
`apps/`	Contains one directory for each application of the project (typically, `frontend` and `backend` for the front and back office).
`cache/`	Contains the cached version of the configuration, and (if you activate it) the cache version of the actions and templates of the project. The cache mechanism (detailed in Chapter 12) uses these files to speed up the answer to web requests. Each application will have a subdirectory here, containing preprocessed PHP and HTML files.
`config/`	Holds the general configuration of the project.
`data/`	Here, you can store the data files of the project, like a database scheme, a SQL file that creates tables, or even a SQLite database file.
`doc/`	Stores the project documentation, including your own documents and the documentation generated by PHPdoc.
`lib/`	Dedicated to foreign classes or libraries. Here, you can add the code that needs to be shared among your applications. The `model/` subdirectory stores the object model of the project (described in Chapter 8).
`log/`	Stores the applicable log files generated directly by symfony. It can also contain web server log files, database log files, or log files from any part of the project. Symfony creates one log file per application and per environment (log files are discussed in Chapter 16).
`plugins/`	Stores the plug-ins installed in the application (plug-ins are discussed in Chapter 17).
`test/`	Contains unit and functional tests written in PHP and compatible with the symfony testing framework (discussed in Chapter 15). During the project setup, symfony automatically adds some stubs with a few basic tests.
`web/`	The root for the web server. The only files accessible from the Internet are the ones located in this directory.

Figure 25. Table describes the content of the root directories [Potencier, p.31]

The tree structure of all application directories is the same:

```
apps/
  [application name]/
    config/
    i18n/
    lib/
    modules/
    templates/
      layout.php
```

Directory	Description
`config/`	Holds a hefty set of YAML configuration files. This is where most of the application configuration is, apart from the default parameters that can be found in the framework itself. Note that the default parameters can still be overridden here if needed. You'll learn more about application configuration in the Chapter 5.
`i18n/`	Contains files used for the internationalization of the application—mostly interface translation files (Chapter 13 deals with internationalization). You can bypass this directory if you choose to use a database for internationalization.
`lib/`	Contains classes and libraries that are specific to the application.
`modules/`	Stores all the modules that contain the features of the application.
`templates/`	Lists the global templates of the application—the ones that are shared by all modules. By default, it contains a `layout.php` file, which is the main layout in which the module templates are inserted.

Figure 26. Table describes the content of application directories [Potencier, p.31]

4.4.2 Modules and actions

This module contains actions and templates that can be invoked only by the administrator. It includes most administrator functionalities such as assigning second examiners to supervisors or projects, rooms to programmes and adding or deleting rooms available for scheduling.

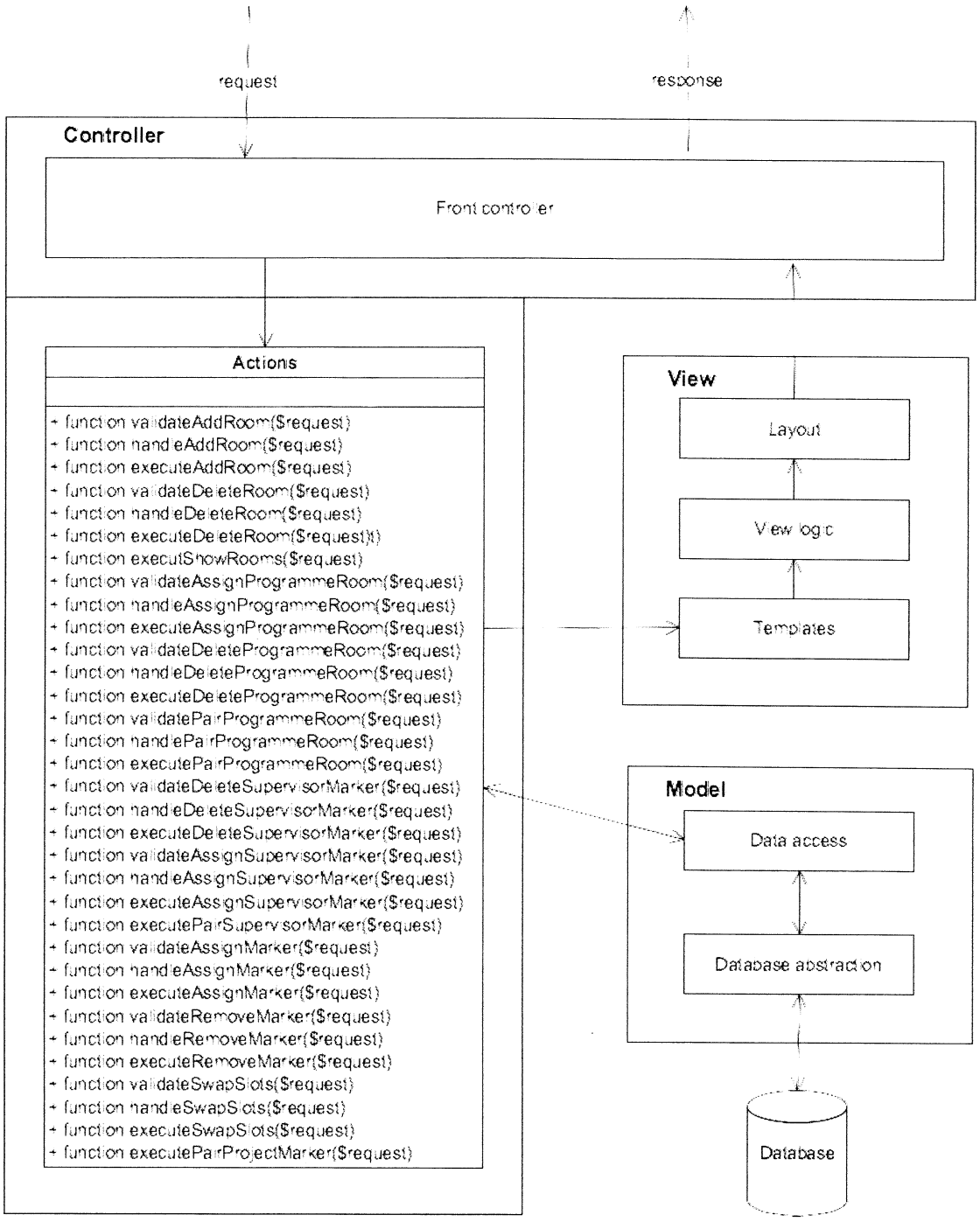

Figure 27. Class diagram of admin module

Login module contains just one template and three actions "executeLogin", "executeLoginSubmit" and "executeLogout". The main functionality is the authentication of users.

This module contains all templates related to displaying and editing a timetable.

4.4.3 High-level prototypes

A navigation menu is based on drop down boxes. It separates clearly the content of interface in two main options, timetable and settings. All web pages relating to the timetable are under the category "Timetable". Pages related to setting up the scheduler system are under the "Setting" category. The student navigation menu contains just options for displaying the timetable, either personal or one containing all student presentations. Faculty members have an additional feature to add their time constraints. Administrator have all these features and in addition can navigate to web pages such as edit timetable, assign second examiners to projects, assign second examiners to supervisor and vice versa, and assign rooms to programmes. Under the main navigation is located another menu, but this is related to the particular page.

Displaying timetable is in the calendar-based form. Each column represents one time slot containing information such as name of student, supervisor and second examiner, the time and place where the student's project is presented.

<<|Navigate to different dates|>>

<table>
<tr><th>Room 1</th><th>Room 2</th><th>Room 3</th></tr>
<tr><td>| Date & time</td><td>| Date & time</td><td>| Date & time</td></tr>
<tr><td>Student
Supervisor
Second examiner</td><td>Student
Supervisor
Second examiner</td><td>Student
Supervisor
Second examiner</td></tr>
<tr><td>| Date & time</td><td>| Date & time</td><td>| Date & time</td></tr>
<tr><td>Student
Supervisor
Second examiner</td><td>Student
Supervisor
Second examiner</td><td>Student
Supervisor
Second examiner</td></tr>
<tr><td>| Date & time</td><td>| Date & time</td><td>| Date & time</td></tr>
<tr><td>Student
Supervisor
Second examiner</td><td>Student
Supervisor
Second examiner</td><td>Student
Supervisor
Second examiner</td></tr>
<tr><td>| Date & time</td><td>| Date & time</td><td>| Date & time</td></tr>
<tr><td>Student
Supervisor
Second examiner</td><td>Student
Supervisor
Second examiner</td><td>Student
Supervisor
Second examiner</td></tr>
</table>

Figure 28. Format of timetable

The feature "edit timetable" enables an administrator to edit a generated timetable based on the format described above. All time slots will be draggable objects able to be moved after clicking and holding the mouse cursor on it. After dropping the time slot the time slots are swapped and changes are saved using Javascript and AJAX request.

To assign a second examiner to a project, the object representing the project is dragged and dropped within the area representing a list of projects to be marked by the second examiner.

Figure 29. High-level prototype of interface used to assign second examiners to projects

To assign a second examiner to supervisor, select faculty members from drop down boxes as supervisor and second examiner and click on the single yellow arrow. The double yellow arrow represents assigning in both ways for instance faculty member B as second examiner is assigned to supervisor A, and faculty member A as second marker is assigned to all project, which supervises B.

The same rules apply to red arrows apart from the fact that they represent pairs of faculty members who cannot be assigned together. Below these components, lists of all of the already assigned pairs will be provided.

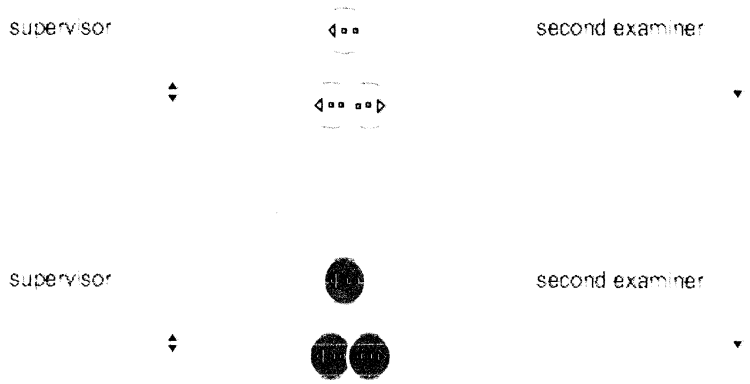

Figure 30. High-level prototype of interface used to assign second examiners to supervisors and vice versa, or create pairs of faculty members who can be assigned to each other.

This feature will be implemented as a simple dropdown box containing all programmes and another dropdown box containing all the available rooms. After selecting appropriate options, a button to submit the data will be provided. All rooms already assigned will be displayed below the components.

5 Implementation

This section describes the implementation phase of this project. It covers explanation of technologies and the approach used, as well as a detailed description of the problems arisen during the coding. The whole implementation process is divided in 3 deliverables. The first deliverable is a scheduler system without any extended features. The second deliverable is the scheduler system including extended features explained in section 3.3. The last deliverable is a graphical user interface (GUI) provided for all groups of users.

Section 5.1 contains a description about DAL package (backend engine) implementation.

Section 5.2 describes the scheduler system (mainly solver package) without extended functionalities (first deliverable), problems arisen after implementation of the first deliverable and how the problems were solved.

Section 5.3 describes an implementation of extended functionalities of the scheduler system.

Section 5.4 is about the GUI development process.

5.1 Database access layer of backend engine

The database access layer (DAL) package is coded in Java language version 1.6 and designed to be as maintainable as possible. To switch to a different Database Management System, software managing databases (DBMS) compatible with SQL queries, just one method "connect()" in the class "SQLConnection", needs to be changed depending on what DBMS will be used. All settings information related to a connection to DBMS is kept in one class "DALConfig" to make it maintainable for future changes. The class containing all SQL queries "DAL" is separated into two parts. One part contains the queries retrieving information from the old system database (external). The latter one is related to the database design specifically for this system (internal). Therefore, the database can be located on different database servers if needed.

5.2 Scheduler with core functionalities (1st deliverable)

All classes of backend engine were coded in Java language version 1.6 by using programming tool "Eclipse". The version of Java on the department informatics server where the system will be deployed will be checked before the decision about a constraint solver is made, because of the fact that the most recent version of Java was required by "Choco" library.

5.2.1 Application Programming Interface (API) of "Choco"

To implement the constraints described in section 4.3.2.3 the API of "choco" library was required to be explored in detail. Small blocks of Java code are written by exploring the API to be sure of the use of appropriate functions to create a constraint model.

List of constraint functions of "Choco" library used in the constraint model:

- allDifferent
  "States that all pairs of variables have distinct values (which is useful for some matching problems) (v_1 != v_2 != v_3 != ... !=v_n)."

- and
  "States that every constraints in arguments have to be satisfied (c_1 ∧ c_2 ∧...∧ c_n)."

- atMostNValue
  "States that the number of different values occurring in the array of variables to be at most nvalue."

- eq
  "States that the two arguments are equals (x = y)."

- gt

  "States that the first argument is greater than the second one (x > y)."

- ifThenElse

  "States that if the first constraint is satisfied, the second one should be also verified, otherwise the third one should be verified ((c_1 ∧ c_2) ||(!c_1 ∧ c_3)).

  Can also state that if the first constraint is satisfied, it returns the second parameter, otherwise it returns the third one."

- lt

  "States that the first parameter is less than the second one (x < y)."

- max

  "States that the last argument is equal to the greater value of the other arguments (z = max(x_1, x_2, ..., x_n))."

- min

  "States that the last argument is equal to the smaller value of the other arguments (z = min(x_1, x_2, ..., x_n))."

- neq

  "States that the two arguments are not equals (x != y)."

- occurenceMax

  "States that the occurrence variable is at most equal to the number of occurrences of the given value in the list of variables."

- or

  "States that one or more of the constraints in arguments have to be satisfied (c_1 || c_2 ||...|| c_n)." [choco constraints]

5.2.2 Generating time slots

Before any process of the scheduler starts, time slots need to be generated. For this purpose the function "generateTimeSlotObjects" is implemented. It retrieves all required information such as a start and end date of a schedule, start and end time of a day in the schedule and a length (number of minutes) of one time slot. After all these data are populated from database, the method iterates in two nested loops. The outer one has a stopping condition when no more days are available for scheduling. The inner one creates time slot objects and iterates from the start time by adding the length of one slot to it until it reaches the end time. The method returns an arraylist containing the generated time slots.

5.2.3 Generating time constraints

Before time constraint objects can be generated, the actual time constraints must be inserted into the database. An XML file structure will be designed and sent to the department informatics office in order to obtain all faculty members' time constraints. This approach is used because there is restricted access to the database containing the timetables of all faculty members. When the file was provided the method "readXmlFile" of class "XmlProcessing" reads it and stores the constraints for further processing.

Afterwards, the methods "generateProjectTimeConstraints" and "generateRoom-TimeConstraints" are invoked. The basic principle of these two classes is identical. They iterate through the time constraints retrieved from database and create time constraint objects. The objects are stored in a hash table where a key is a project id or room id and the value is a vector of time constraints objects.

XML Structure of the file:

```
<?xml [illegible] ="1.0"?>
<constraints>
    <faculty>
        <member>
            <person_code> sampson</person_code>
            <constraint>
                <start>2009-05-05 14:00:00</start>
                <end>2009-05-05 15:00:00</end>
            </constraint>
        </member>
    </faculty>
<constraints>
```

5.2.4 Generating variable domains

Each variable of the "choco" constraint model must have a domain containing all valid values. The idea of generating a project domain is that the method "generateProjectDomain" iterates through a collection of time slots and time constraints of supervisor and students involved in the particular project and add to the domain just ids of slots, which are valid for both of them. To find out which slots are valid the method "isSlotInDomain" is used to compare a time slot and constraint, passed as parameters.

The rule is: start time of slot >= end time of constraint AND
end time of slot >= start time of constraint

5.2.5 Generating second examiner load

One of the requirements is to generate second examiners' loads (maximum amount of projects they are allowed to mark).

The basic rule is: L2 = 2*L – NP

where L2 stands for load of faculty member as second examiner
L stands for load of the faculty member as supervisor
NP stands for a number of projects the faculty member supervises

Because the sum of loads generated by this rule is very rarely equal to the number of projects to be marked, the loads must be amended. The idea of how the loads are changed, is to decrease or increase each load by one until the sum of loads is equal to the number of projects.

5.2.6 Generating timetable

To generate a timetable, the object of "choco" class "CPSolver" must be created and the constraint model must be added to it. Before the method "solve", that searches for a solution can be invoked, the optimization of solver should not be omitted. Two key points are important to make searching more efficient: how to prioritise selecting variables and how to select values from variables' domains.

The initial search strategy is:

Selector for variable: "DomOverDeg"

> "A heuristic selecting the variable with smallest ration (domainSize / degree), the degree of a variable is the number of constraints linked to it." [choco search strategy]

Selector for value: "RandomIntValSelector"

> "Selecting randomly the value in the domain"
> [choco search strategy]

5.2.7 Key problems and their solutions

Testing the first deliverable revealed that java heap space had to be extended. After setting enough amount of memory, the problem was that the time needed to generate a solution was substantially long. The process ran for 48 hours and no solution was found. After consideration of possible alternatives, it is decided to divide the CSP into two separate problems; assigning second examiners to projects and assigning time slots and rooms to projects. This decreases the level of constraints and scope of search space. It also enables higher control over the process of assigning second markers (see next section).

5.3 Extended features of scheduler (2nd deliverable)

5.3.1 Assigning second examiners to projects

The pseudo code of assigning second examiners to projects (method "compareDomains") is as follows:

1. Generate loads of second examiners by invoking method "generate2ndMarkersLoads"

2. Assign second examiners to particular projects as set up by administrator

3. Assign second markers to projects of particular supervisors as set up by administrator

4. While there are projects without second examiners do
   Find project with smallest domain
   Iterate through collection of second examiners and do intersections of project (including supervisor and student) and second examiner domains, and select those who have the intersection containing the highest number of values. Exclude second examiners who were already assigned to projects and fully used their loads.

   Check whether the pair of supervisor and second examiner is not assigned to more than the limit set up by administrator. If so, select a different examiner and check again.

   Assign the second examiner to the project and decrease the load of the examiner by one.

   If the load is less or equal to zero, mark the examiner as unavailable to be assigned.

   The extra features limiting same faculty members to be assigned to the same projects, assigning second examiners to projects or supervisors are implemented within this method.

Second examiners with the smallest domains have higher priority to be selected first and are assigned to projects where intersections of project and second examiner domains are largest sets. The reason is that the bigger domains are able to obtain the higher probability that can be achieved to generate a valid timetable. Also it supports the feature, which helps to find the maximum number of slots to be possibly swapped.

5.3.2 Relocation time constraint

To implement extra feature that gives each faculty member a time for relocation, the following pseudo code was used:

1. Select all music projects

2. Iterate through music project collection

    For each supervisor of a music project, set the constraint that they cannot be assigned to a non music project one time slot before and after

5.3.3 Possible swaps feature

This feature is implemented as "servlet" method "doGet" of class "Possible-Swaps". When the user sends a request to this "servlet" with the parameter containing a project id, the method retrieves the time slot id of the project from the database. Afterwards it iterates through all of the project domains (intersections of student, supervisor and second examiner domains) and returns the ids of the projects whose domains contain the slot id of the project passed as parameter. The projects are marked with a different colour and are valid to be swapped.

5.4 Graphical user interface (GUI)

All parts of the GUI are implemented in PHP language by using PHP framework "Symfony". Three modules are created within the GUI application. Module "Login" deals with authentication containing functions to login and logout. Module "Timetable" deals with showing a timetable either personal or containing all presentations. Only administrators can amend the timetable. Module "Admin" contains all administration functions.

As one of the requirements is to make the application user-friendly and easy to use, the GUI usability is critical.

5.4.1 Drag & drop technique

To make the interface interactive and easy to use, the drag & drop technique was used. One of the biggest advantages is that a user is not temped to type anything in and it prevents the user from easily making an error. The second significant advantage is that manipulation with the timeslots is quick and clear to users, which is important when managing large-scale timetables.

This technique will be coded in JavaScript using the external "Prototype" library. When a user clicks on a "draggable" object (project time slots) and holds down the left mouse button, the object can be dropped by releasing the button on droppable objects (containers for project tome slots).

When drop event occurs, an AJAX request to invoke an appropriate function is made to the web server. The function executes and sends back a response to the client. The response is processed on the client side (JavaScript) and if required the document (web page) is amended by Document Object Model and results of the request are displayed.

During the process the user is always informed of the current status of application, either if it is waiting for a response (loading bar), exception occurred, the task was accomplished, or failed.

5.4.2 Login

To login to the system, a simple login form is provided to fill in with required information such as username and password. When the form is submitted, the "submitLogin" function is invoked to authenticate the user. After successful authentication, the timetable is displayed.

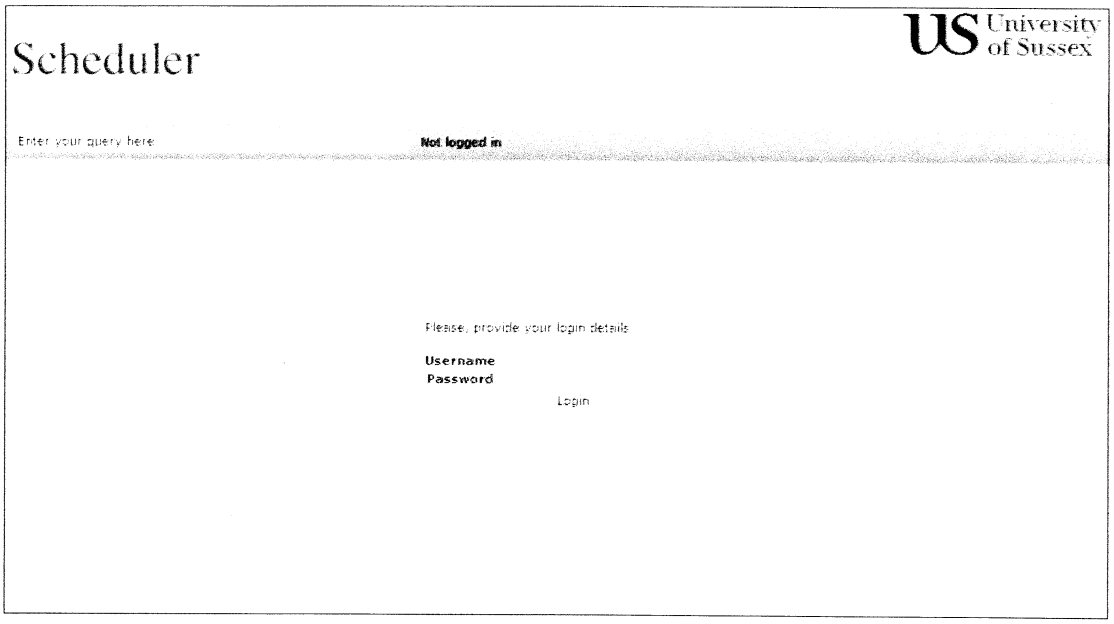

Figure 31. Login page

5.4.3 Personal timetable

When a user is authenticated and clicks on the option "Personal timetable", their login details are read from a session and an appropriate timetable is displayed to the user.

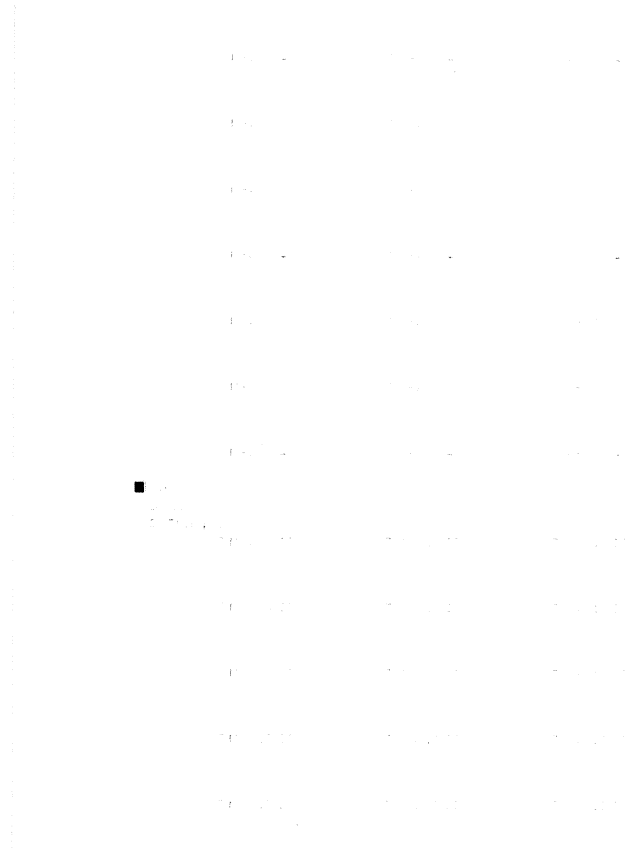

Figure 32. Personal timetable page

5.4.4 Timetable containing all presentations

This option shows a timetable containing all student presentations. The timetable is implemented as a component in the timetable module (path "actions/components.class.php"). The method "executeTimetable-Component" iterates through all time slots comparing them with assigned time slots to presentations. If an empty time slot is hit, an empty column html code is generated; otherwise code for the time slot representing the presentation, including information about student, supervisor and second examiner is generated.

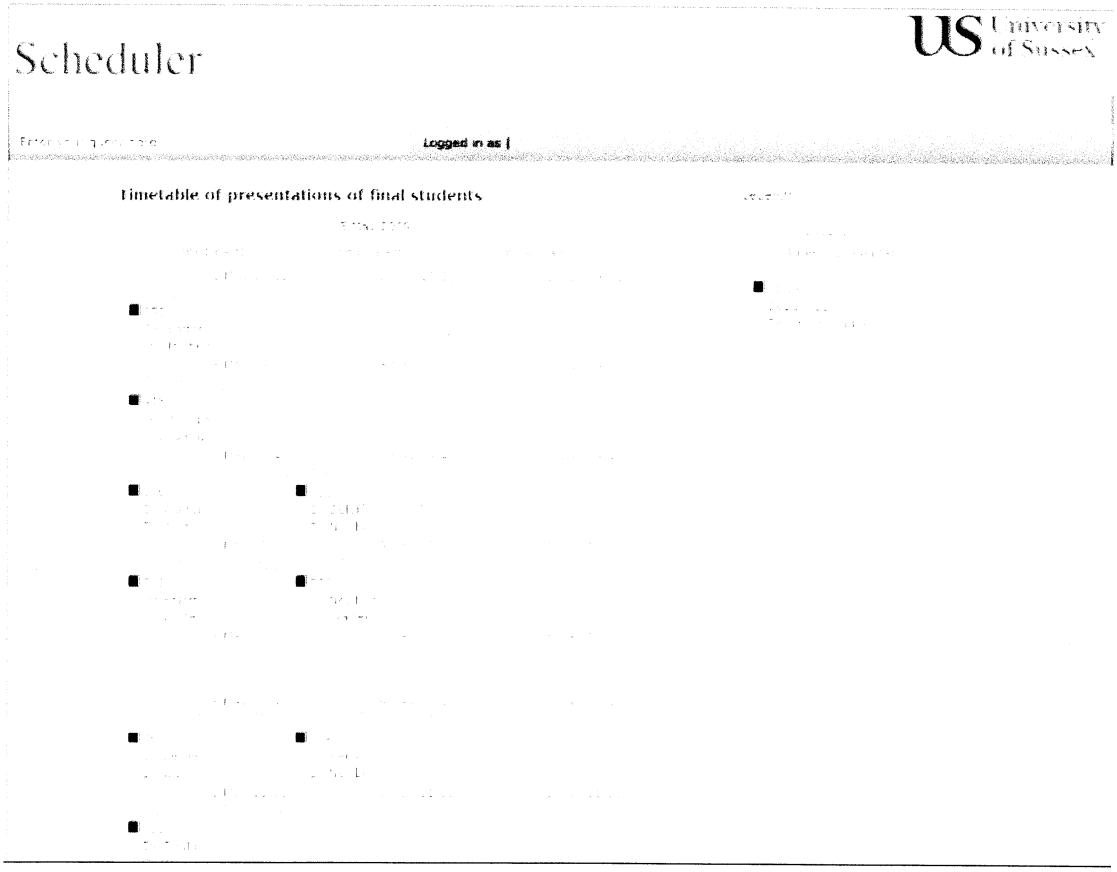

Figure 33. Page shows a timetable containing all student presentation.

5.4.5 Edit timetable

This feature is accessible only by the administrator. The timetable can be amended just by dragging and dropping timeslots into desired positions. This "drag&drop" technique is implemented using JavaScript (external library "Prototype" used), AJAX and PHP. All timeslots are draggable and can be dropped to empty containers with a fixed date and time. During the generation of these containers the slot ids are mapped to the ids of the containers. The same rules apply to project time slot ids and therefore they become mapped to the project ids. When the user drags the timeslot and drops on some container, an AJAX request with parameters project id and timeslot id is made to the web server. A PHP function updates the data and returns a response containing information to swap the project. It is recognized whether the project is swapped either with another project or an empty time slot. The swap is stored and a "step back" functionality is provided in case that the swap was made by mistake. The processes are saved as a stack, thus there is no restriction how many steps back can be processed. Note that the number of possible steps back is shown in the brackets in the button. After clicking on the button "Show All" a timetable containing all presentations is displayed.

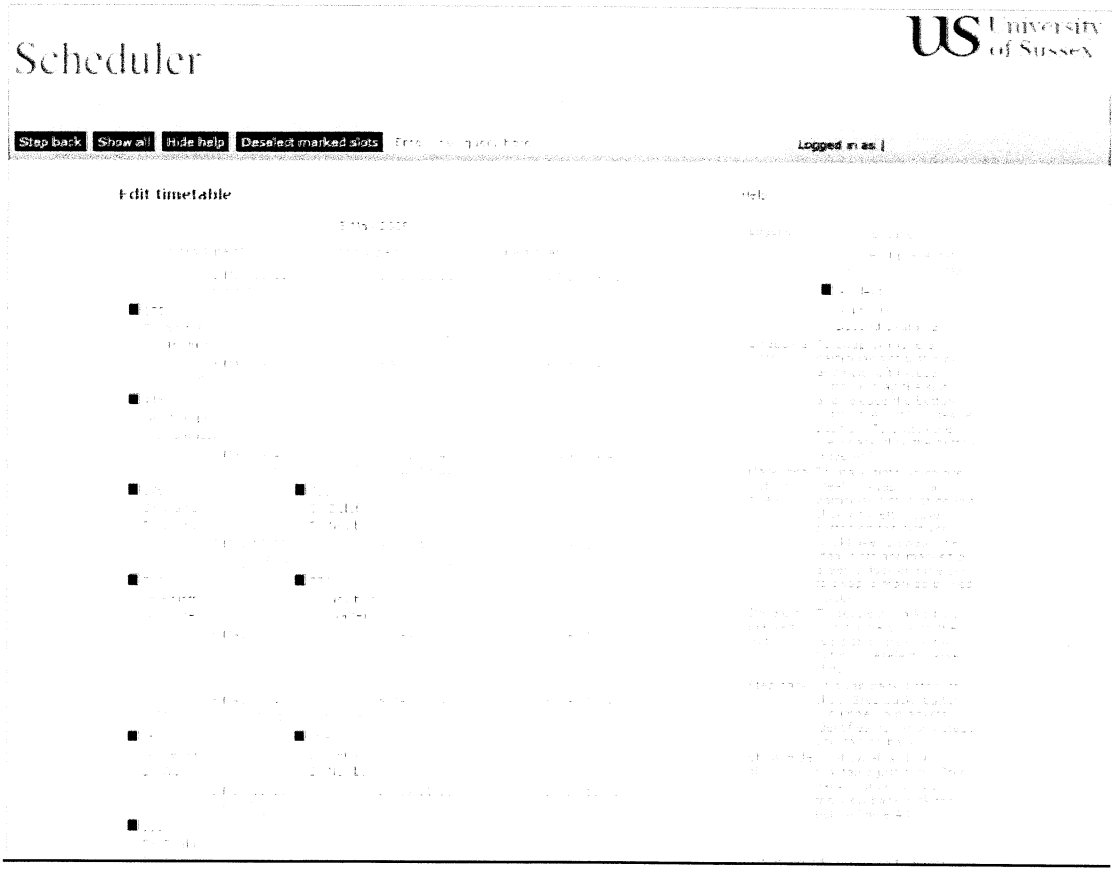

Figure 34. Edit a timetable

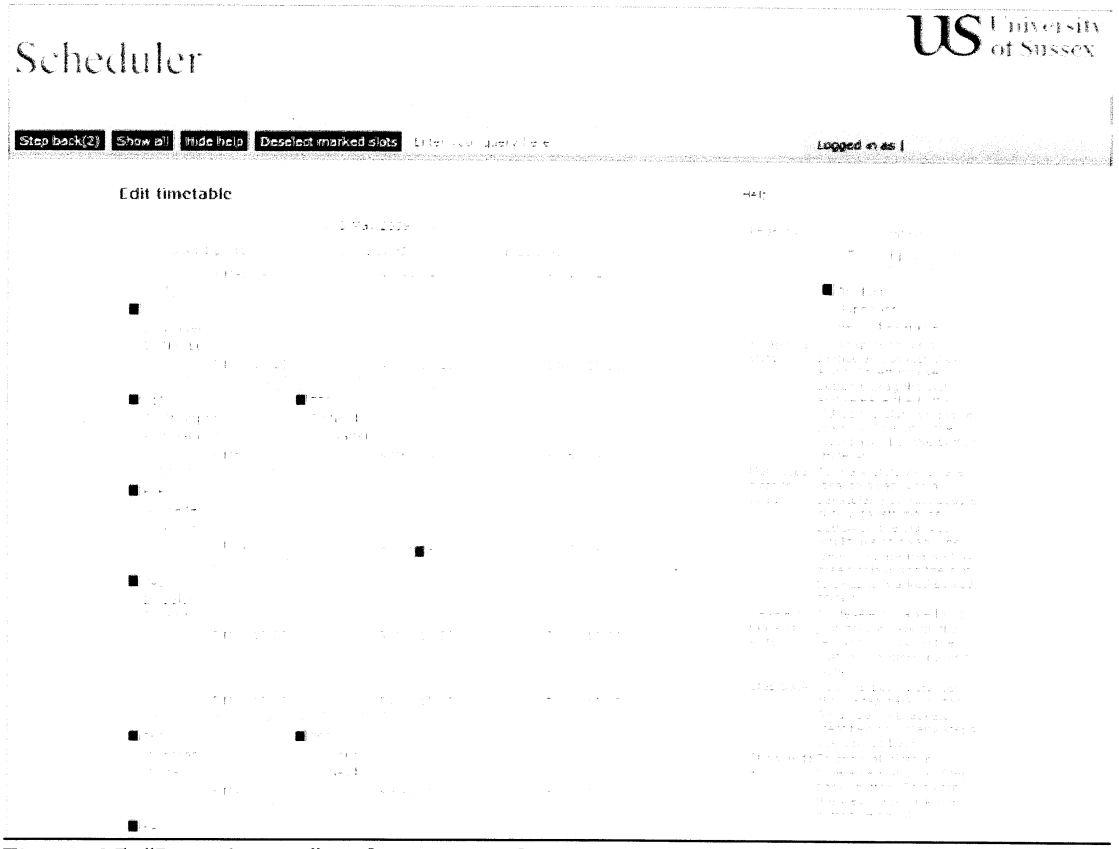

Figure 35. "Drag&Drop" technique used to swap a time slot

The extended feature described in the section 3.3 (showing timeslots valid to swaps) can be invoked just by double clicking on a project slot. All projects valid to be swapped by the project clicked on, are marked with a different colour to distinguish from the timeslots not valid to be swapped.

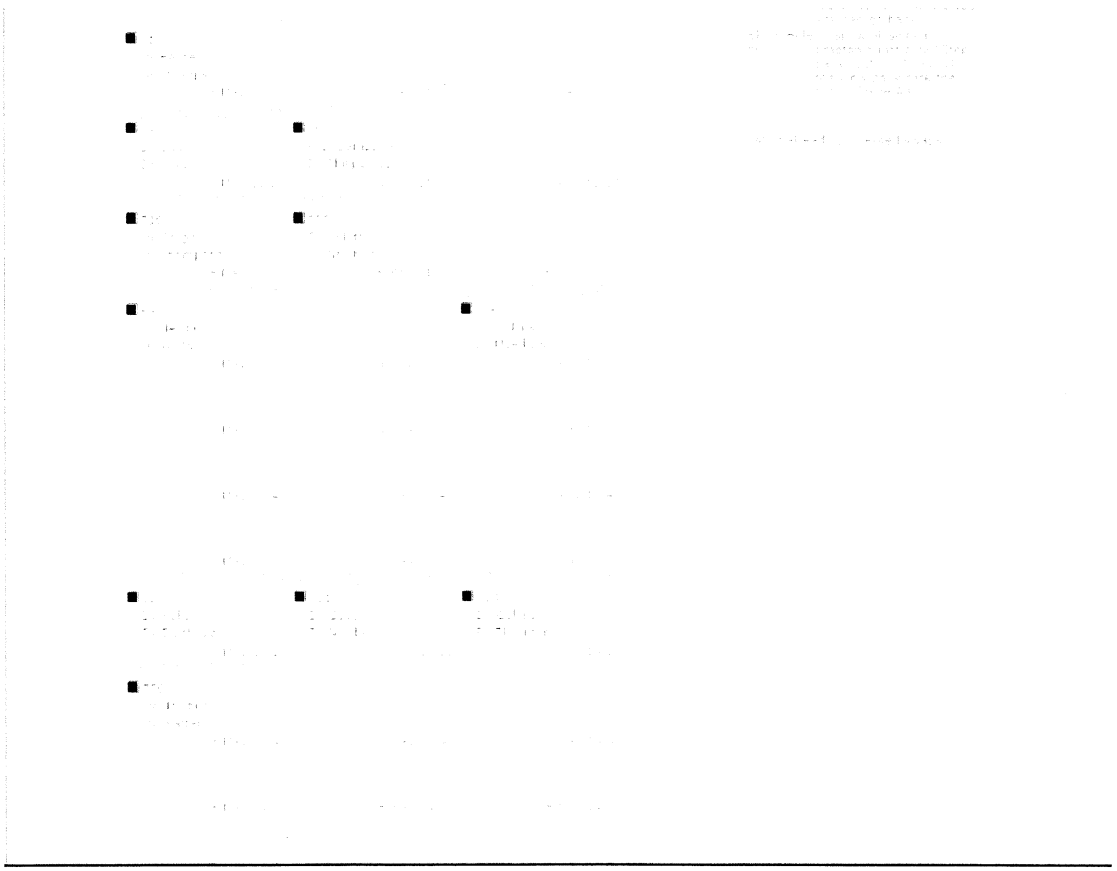

Figure 36. Valid project to be swapped are marked with different colour.

5.4.6 Informing user about current status

Whatever function is performed on the system by the user will have to have a comment or notification informing the user that it has either been successful or now, along with the progress that an action is actually taking place. The following screenshots show how this information is displayed to the user.

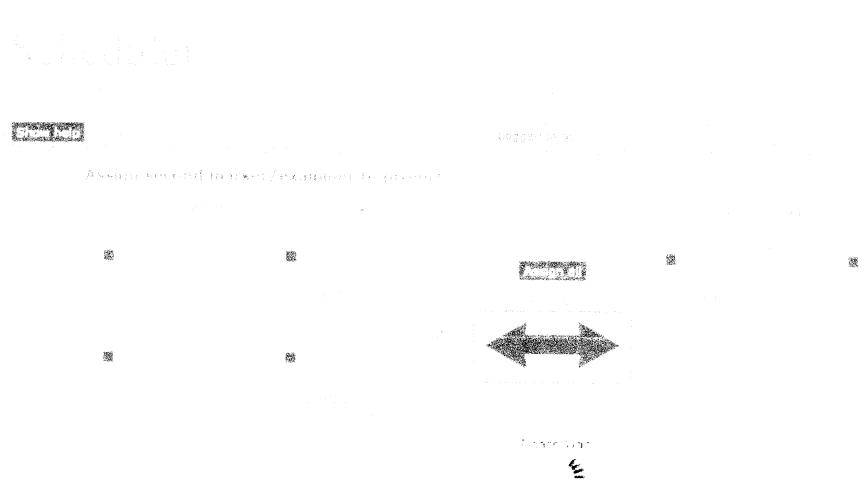

Figure. 37 System is waiting for a response from server

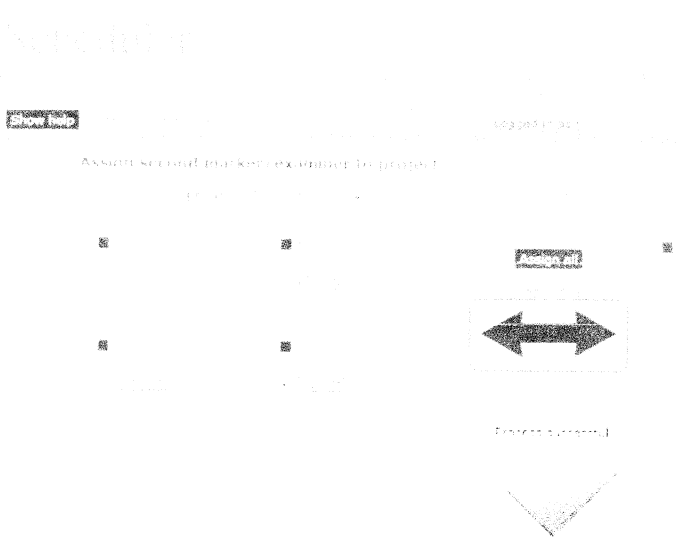

Figure 38. A request or process was successful

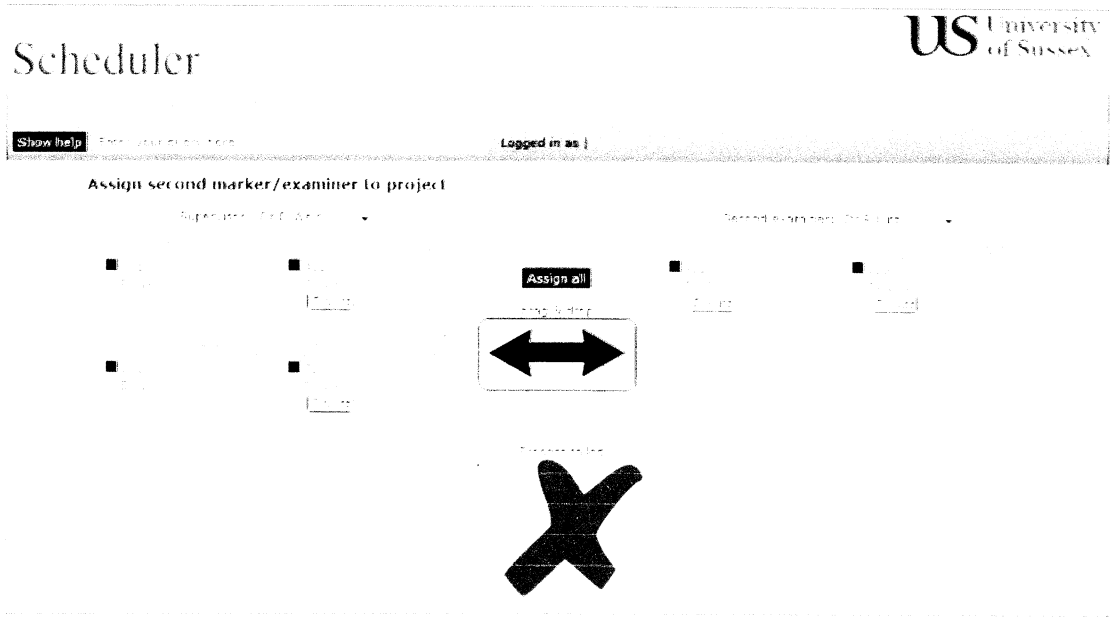

Figure 39. A request or process failed

5.4.7 Adding time constraints of faculty members

The interface for faculty members is provided to add their time constraints. To add an entry, a drop down boxes to select the dates and time need to be used. Time constraints can be deleted just by clicking on the buttons next to them.

Figure 40. Adding faculty member's time constraints

5.4.8 Assigning second examiner to supervisor and vice versa

This feature provides the administrator to assign particular second examiners to projects supervised by a particular faculty member. It can also be done the other way, so that both of them are involved in projects either as supervisors or second examiners. The assignment can be deleted by clicking on the button just next to it.

The administrator can also specify which members of faculty cannot be involved in the same project as the supervisor and second marker.

In order to comply with usability, the feature of the drop down box is implemented in order to minimize user error and increase efficiency.

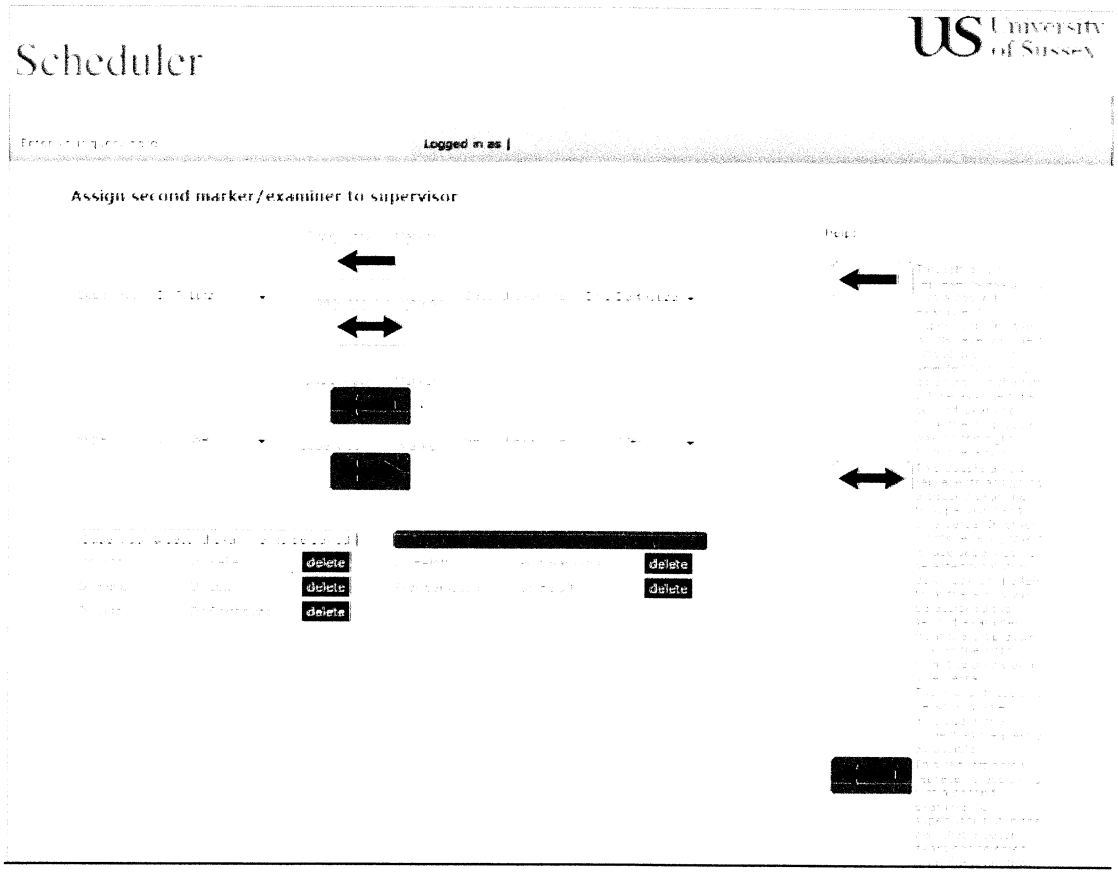

Figure 41. Page shows a feature of assigning faculty members to each other

5.4.9 Assigning room to programme

The feature "assigning a room to particular programme" is implemented by the interface displayed in the screen shot below. The same idea is applied as in the previous section.

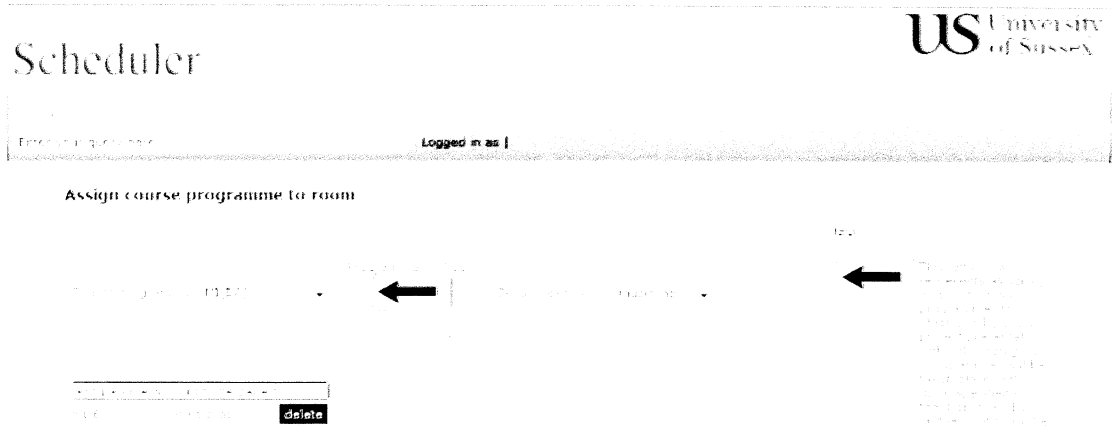

Figure 42. Assigning rooms to particular programmes

5.4.10 Add room for scheduling

The administrator is allowed to add or delete rooms where projects are presented. The simple interface is provided for this task.

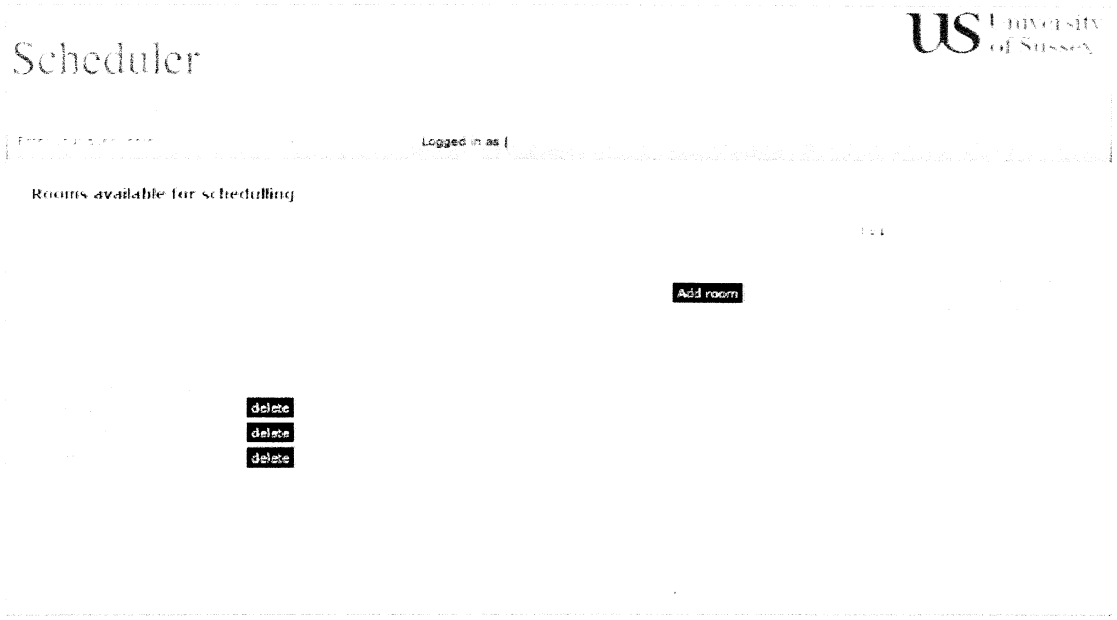

Figure 43. Adding or deleting rooms available for scheduling

6 Testing

6.1 Correctness of timetable

A black-box testing approach was used to test the correctness of the results produced by the scheduler. A unit test method was written specifically to test every constraint applied during the generating a timetable. It reveals which particular constraints are not satisfied. The test automatically runs after each generation of timetable to minimise the fact that an invalid timetable is provided to the informatics department office.

6.2 Optimization and efficiency of solver

Several settings of the search strategy were tested to improve the efficiency of the scheduler. The search strategy has the impact on a selection priority of variables and values in domains.

The list of strategies tested:

Test strategy #1

Variable selector:

StaticVarOrder (A heuristic selecting the first non instantiated variable in the given static order)

Value selector:

MinVal (Selecting the lowest value in the domain)

Results: after 48 hours no solution

Test strategy #2

Variable selector:

StaticVarOrder (A heuristic selecting the first non instantiated variable in the given static order)

Value selector:

RandomIntValSelector (Selecting randomly the value in the domain)

Results: after 48 hours no solution

Test strategy #3

Variable selector:

MinDomain (A heuristic selecting the variable with smallest domain)

Value selector:

RandomIntValSelector (Selecting randomly the value in the domain)

Results: The time was changing because of randomness, a solution founded within approximately 3 minutes.

Test strategy #4

Variable selector:

DomOverDeg (A heuristic selecting the variable with smallest ration (domainSize / degree), the degree of a variable is the number of constraints linked to it.)

Value selector:

RandomIntValSelector (Selecting randomly the value in the domain)

Results: The time was changing because of randomness, a solution founded within approximately 5 minutes.

6.3 Use case testing

All functionalities were tested by following the scenarios and instructions of the extended functionalities section.

7 Deployment

The system was deployed and tested on a local machine. "WAMPP" software was installed to make the machine behave like a web server. The source code must be placed in "www" directory to be able to execute.

A life-version of the system will be deployed on the informatics web server. After discussion with the helpdesk, the server's version of PHP will be updated to the most recent one. This is required to deploy the system based on PHP "Symfony" version 1.2. If any unexpected problems arise related to the compatibility with the PHP version, the version 1.1 of the framework will be used otherwise. The compatibility with the "Symfony" version 1.1 has already been tested by running the compatibility test script on the server.

8 Conclusion

In general, no serious problems have arisen during the whole project. It was found that during the analysis stage of the project, only the core functionalities of the system were discovered. However during the design phase of the project and through further discussions with the client, there were other extra functionalities took into consideration that would be very beneficial to be implemented from the administrator point of view.

This project has taught the valuable importance of the analysis stage and how this is one of the most difficult parts of the project, due to the fact that truly being able to find out what the user of the systems actually wants is not easy. This has proved a valuable learning curve because unlike previous assignments where the task and the problem had been given, within the project a more real-world experience has been taken into consideration, having to find out the needs of the user and then develop the system accordingly.

The use of open sourced software "Symfony PHP framework" and "choco" made the project more challenging, because it relied on the qualities of these systems. Consequently, if there was a bug in either of these programs this would have a direct effect on the stability of the system created.

8.1 Assessment of success

All of the functionalities even the extended one have been implemented and the success of the project has been proven by the fact that it has been used to generate this year's final years presentation schedule.

One limitation discovered during the real life testing was that the time slots or chunks could be dispersed. This is a feature that could be implemented into the system with more time given.

8.2 Suggestions for extensions

Currently importing of faculty members' time constraints is done by reading them from the xml file provided by the informatics department office. As an extension of the system this could be replaced by importing the data straight from the current University timetabling system.

The other possible extension to the system is to use a thread controller, which can be implemented in order to have greater control of other threads generating the timetable. This means that more timetables can be generated at the same time whilst being parallel. If a function is implemented that can measure the quality of the timetable, then one of them can be chosen as the best solution. The controller can pause or suspend the thread if required.

The timetable could be exported as an "ical" file, so users can use external software supporting this type of file and add the timetable to the other applications such as google calendar, ical, outlook express, etc. Also an ical file

parser can be written that would read the time constraints from the ical file provided by user.

BIBLIOGRAPHY:

- [Pinedo, 1995] Pinedo, M. (1995). *Scheduling - Theory, Algorithms, and Systems.* Prentice Hall, Englewood Cliffs.

- [T'kindt, Billaut, 2006] T'kindt, V. and Billaut, J. (2006). *Multicriteria Scheduling - Theory,Models and Algorithms.* Springer-VerlagBerlinHeidelberg, Berlin ,Heidelberg.

- [np-complete] Wikipedia, The free encyclopedia, 2008. *NP-complete.* URL: http://en.wikipedia.org/wiki/NP-complete Accessed: 15th Nov 2008

- [Corne, Fang, Mellish] Dave Corne, D., Fang, H. and Mellish CH. *Solving the Modular Exam - Scheduling Problem with Genetic Algorithms.* DAI Research Paper No. 622

- [code&conduct] BCS, September 2004. *Code of good practice.* URL: http://www.bcs.org/server.php?show=conWebDoc.1589 Accessed: 11 Nov 2008

- [choco white paper] Choco Library, 2008. *General description of the library* URL: http://www.emn.fr/x-info/chocosolver/lib/exe/fetch.php ?id=presentation&cache=cache&media=pdf:choco-presentation.pdf Accessed: 27 Feb 2009

- [ajax def] Wikipedia, The free encyclopedia, 2008. *Ajax.* URL: http://en.wikipedia.org/wiki/Ajax_(programming) Accessed: 26th Apr 2009

- [servlet def] Sun Microsystems, 2009. *Java Servlet Technology.* URL: http://java.sun.com/products/servlet/ Accessed: 26th Apr 2009

- [php def] Wikipedia, The free encyclopedia, 2008. *PHP.* URL: http://en.wikipedia.org/wiki/PHP Accessed: 26th Apr 2009

- [Potencier] Potencier, F. (2009). *Symfony, the definitive guide to symfony.* Sensiolabs.

- [javascript def] PCMag, 2009. *Javascript.* URL: http://www.pcmag.com/encyclopedia_term/0,2542,t=JavaScript&i=45585,00.asp Accessed: 27th Apr 2009

- [choco constraints] Choco manual, 2009. *Cocho constraints.* URL: http://www.emn.fr/x-info/choco-solver/doku.php?id=constraints Accessed: 27th Apr 2009

- [choco search strategy] Choco manual, 2009. *Search strategy.* URL: http://www.emn.fr/x-info/choco-solver/doku.php?id=solver #search_strategy Accessed: 27th Apr 2009

Appendix

Log

Analysis phase of a development process

13/10/ 2008 | **Initial meeting with supervisor**

- General discussion of issues related to a final project
- General discussion about intern report
- General discussion regarding a project prospect

20/10/ 2008 | **Interview customer to gather requirements**

- Discussion related to system requirements
- General discussion about user of the system
- General discussion about user scenarios

26/10/ 2008 | **Analysing requirements**

- Analyse requirements from a description of the problem

04/11/ 2008 | **Interview customer to gather non-functional requirements**

- Discussion about the requirements related to system deployment
- Discussion about user interface
- Discussion related to efficiency of producing a timetable

14/11/ 2008 | **Generating Use Case Diagrams**

- Producing a use case diagrams by using the analysis of the system
- Producing a use case diagram for an administrator
- Producing a use case diagram for a faculty member
- Producing a use case diagram for a student

18/11/ 2008 | **Writing scenarios**

- Writing scenarios for an administrator
- Writing scenarios for a faculty member
- Writing scenarios for a student

25/11/ 2008 | **Producing project plan**

- Produce project plan

26/11/ 2008 | **Background research**

- Research related to scheduling/timetabling problem

01/12/ 2008 | **Study Constraint Satisfaction Problem (CSP)**

- Research on CSP
- Research on how to model CSP
- Research on techniques using to solve CSP

11/12/ 2008 | **Research regarding constraint solver**

- Research on constraints solver available as open source software
- Research on optimization techniques used to optimize the problem in order to speed up the solving process

Design phase of a development process

15/01/ 2009 | **Design high level class diagram**

- Identify main classes
- Create basic class diagram representing general structure of the system
- Identify main relationships between classes

18/01/ 2009 | **Design database model**

- Design Entity-Relationship Model (ERM) of database

21/01/ 2009 | **Design high level sequence diagram**

- Identify the sequence of creating objects and function calls
- Design high level sequence diagram

/01/ 2009 | **Meeting with supervisor**

- Discussion regarding intern report
- Discussion about overall structure of object oriented model of the system

28/01/ 2009 | **Design low level class diagrams**

- Design low level diagram for every part of the system

Coding and testing phase of a development process

06/02/ 2009 | **Implement designed ERM**

- Implement previously design ERM

21/02/ 2009 | **Meeting with supervisor**

- Discussion related to produced ERM

24/02/ 2009 | **Implement Database Access Layer (DAL)**

- Implement DAL of solver package (see description of package)

30/02/ 2009 | **Design overall/ formal model of CSP**

- Formal model of constraints of the CSP
- Model formally every constraint of the CSP

08/03/ 2009 | **Implement constraint model for constraint solver**

- Implement the solver package of the system (see description of the package) by using Java version 1.6

27/03/ 2009 | **Testing database access layer (DAL)**

- Test all functions of DAL

28/03/ 2009 | **Testing solver package**

- Write a code for unit tests to validate produced timetables

30/03/ 2009 | **Study documentation of "Symfony" PHP framework**

- Study all parts of the PHP framework used to implement the GUI of the system
- Write simple code to practise the syntax of the framework

07/04/ 2009 | **Implement a graphical user interface (GUI)**

- Implement GUI for all users of the system
- Advanced techniques (AJAX) used to provide an interactive interface for users of the system

14/04/ 2009 | **Testing of GUI**

- Follow user scenarios to test the implemented functionalities

07/04/ 2009 | **Meeting with supervisor and department officer**

- Discussion last details related to producing a final timetable for the year 2009
- Discussion regarding deployment and problems related to it

19/04/ 2009 | **Fixing bugs of solver package**

- Implement GUI for all users of the system
- Advanced techniques (AJAX) used to provide an interactive interface for users of the system

20/04/ 2009 | **Producing a draft of final timetable/schedule 2009**

- Set up all setting of the system and run the scheduler to produce a draft of final timetable/schedule

21/04/ 2009 | **Write final report**

- Put all diagrams and notes taken during the whole process to produce a final report