OPUS-College Timetable Module
Design Document

A. Cornelissen, M.J. Sprengers, B. Mader[1]
Radboud University Nijmegen

Nijmegen, July 6, 2010

[1]Contact Information: oum@giphouse.nl

Abstract

This document will provide an easy to implement design for a timetabling module for the OPUS-College (further referred to as OPUS) university administration software system, specifically tailored to the needs and constraints of the Copperbelt University (CBU) in Kitwe, Zambia. An overview of the system module is given, including descriptions, use-cases, class models, database models, activity diagrams, and in more detail an elaboration on the inner workings of the algorithm behind timetabling. It is our intention that this will lead to a kick-start for a development team responsible for the actual programming of this module.

Keywords: timetable, memetic algorithm, scheduling, natural evolution, CBU, OPUS.

1 Introduction

The goal of this document is to act as a blueprint for the implementation of a timetabling module for the OPUS university administration software system, specifically tailored to the needs and constraints of the Copperbelt University (CBU) in Kitwe/Zambia.

This document will provide an easy to implement design for a timetabling module, together with an in-depth view at the current situation at the Copperbelt University (CBU) and their requirements, so that the implementation of this module can commence as quickly and as error free as possible.

We start by giving an overview of what the OPUS Timetabling module should do (basic functionality) in Section 2. To design this module it is important to consider what the current situation is, what assumptions we can make, what dependencies are in place, and what the constraints for the timetable module are in general and for the specific situation at CBU. This is described in Section 3. In this document we also describe a blueprint for the implementation. The implementation includes a choice for an algorithm (Memetic) and the rationale for this choice is described in Section 4. General design information in the form of use-cases, class diagrams, activity diagrams and database models is given in Section 5 which describes the architecture of the module. It is explained in more detailed in Section 6.

2 System Overview

This document contains the design of a module for the OPUS university administration system. This module offers the automatic generation of a timetable for educational institutions. The design is based on requirements gathered at the Copperbelt University Kitwe (Zambia) in April/May 2010.

2.1 Basic functionalities

The module for the OPUS system that this document specifies has one main purpose; the generation of a complete timetable for a whole academic term/year for a university/school. This module (especially the timetable generation algorithm) is tailored specifically to the requirements of the Copperbelt University but it should be easy to customize it to fit the needs of other higher learning institutions.

The way it is supposed to work is as follows:
Every school/department selects the subjects they want to give this year/term and the start and end time of their work day. Additional information about the subjects is retrieved from the database, such as study load, assigned teacher, number of students and so on. Only after every school/department has specified their desired subjects, the module retrieves the information about all the rooms of the university, such as capacity, facilities (PCs, Blackboard, ...), location and so on. After all necessary information has been retrieved, the module will generate a timetable

that takes into account all the predetermined constraints and preferences. The algorithm used to generate the timetable tries to generate the best possible timetable which satisfies all constraints and as many preferences as possible.

The generated timetable should fit the requirements of the university but if it doesn not, there should be the possibility to repeat the generation process until a satisfactory timetable has been generated.

3 Design Considerations

When designing and implementing the timetable module for OPUS there are some things that need to be known beforehand. As this module is meant to be tailored to the specific requirements of CBU, information about the current timetabling situation and organizational structures may be required. These points are discussed in the following subsections.

3.1 Current Situation

Currently all timetabling at CBU is done manually with the help of programs like Microsoft Excel © . The following subsections will give you an overview of how the timetables are currently generated at CBU and how the academic year is organized, as well as additional information that might be useful during the development of the timetabling module.

3.1.1 Timetabling Procedure

As this module aims to add an automated timetabling solution to the OPUS system, and is specifically tailored to the requirements of the Copperbelt University Zambia, the first step is to take a look at the current situation at CBU.

The Copperbelt University consists of seven schools (similar to faculties at Radboud University), all of which have their own facilities and each of which is responsible for generating the timetable for their study programs themselves. The person responsible for the timetabling is the assistant dean of each school. At the beginning of the academic year/term, each assistant dean compiles a timetable for his school with Microsoft Excel or an equivalent program. Some schools are in the situation that they do not have enough rooms for all their students, so they have to "borrow" rooms from other schools.

After the timetables have been generated, the assistant deans meet and compare their timetables. Most times there will be clashes concerning location and time. This is due to the lack of communication and lack of rooms in some schools. These clashes are then resolved by manually changing the timetables.

Most schools have the preference to use their own rooms, although that might not always be

possible. There are enough rooms available on campus, but due to the manual rostering procedure and the lack of a centralized place to look for free rooms, the available facilities are not used as efficiently as they could be.

There is frequent input from students during and after the rostering process. The people responsible for the timetables try to incorporate the wishes of the students into the timetable, but this is a tedious process, mainly because of the lack of a centralized room management system.

One thing that has to be taken into account during the timetabling procedure is that the campus of CBU is big, therefore the time it takes the students to get from one building to another has to be considered when generating the timetable.

The current timetabling process very time consuming. The main reason for that seems to be that each school wants to retain complete control over their resources. Even without a dedicated IT supported timetabling system, the current situation could be greatly improved through more cooperation of the schools during the timetabling procedure. Should the schools want to share the control over their resources, there might be less problems during the adoption of the timetabling module.

3.1.2 Organizational Information

The academic year at CBU is split up in 3 terms, each of which is 10 weeks long. The last week of each term is a dedicated test period. Additionally, after the last term there is an exam preparation period where no lectures take place. After the preparation period is over the exam period starts. In this period a student can have up to 2 exams a day, one at 9:00 and one at 14:00. Every exam is in written form, and has to be supervised by a staff member. Please take note that a teacher may not supervise the exam of his own subject, but he has to be present for the first 30 minutes to answer potential questions. In order to participate in the examination, a student has to attend 80% of the subjects lectures.

During the exam period it is very difficult to find enough rooms for all the students/exam groups. This is because during exams the students have to be spatially separated, and therefore the capacity of the rooms decreases. Because all exams are in written form, different exams can be held in the same room, which helps to remedy the problem of decreasing room capacity.

The normal working hours at CBU are from 8:00 to 17:00. All lecturers have to be available for lectures during this time frame but are not required to be on campus the whole time.

A subject is often part of more than one educational program and attended by students from different schools. Although the subject is the same for all students, its code is different in each school. This is very unusual, and should be changed before the roll-out of the OPUS system starts.

Students who fail a subject are allowed to repeat it the next year if they fulfill certain requirements. But this is currently not factored into the timetable, therefore students often have to choose between taking last years or this years subject if their lecture time overlap. The OPUS timetabling module has to be able to avoid these collisions and give every student the chance to take part in

all required subjects.

3.1.3 Network Infrastructure

The local network at CBU consists of state of the art server hardware. The different buildings are connected by fibre-optical cable while normal network cables connect the computers in the building to the backbone. The CBU is also connected to the network of the University of Zambia (UNZA) in Lusaka via a fibre-optical connection.

The problematic thing at CBU is that the internet access is very limited. For instance: when we were in Zambia, the internet connection from CBU was inferior to the one we had in the hotel. The problem is partially caused by 2 long range wireless connections. The first one connects the CBU to their Internet Service Provider (ISP), and the second one connects the ISP to its internet access point. Due to these 2 long range wireless connections, a high percentage of IP packets are lost on their way to the destination. Furthermore, the fact that the packets have to pass through a lot of routers, even before they leave Africa, their Time To Live (TTL) is often exceeded before they reach their destination. This leads to these packets being discarded, making it even harder to access sites that are not located near to Zambia.

3.2 Assumptions and Dependencies

A timetabling module for OPUS has to satisfy certain requirements, provide certain functionalities and conform to certain restrictions. What these requirements, functionalities and restrictions are will be explained in detail in the following subsections.

3.2.1 Generic Timetabling Requirements

Of course there are some basic constraints that are valid for every timetabling system. The following list should include all of the basic constraints that have to be considered when implementing the timetabling module.

Room Constraints: A room can only be occupied by a limited number of people, the number of people is defined by the capacity of the room. A room can only be occupied by one subject at a time.

Time Constraints: Lectures can only be scheduled during normal working hours.

Staff Constraints: A staff member can only be in one place at a given time.

3.2.2 Individual School Requirements

During the gathering of the requirements for the module it became clear that individual schools want to retain control over all of their lectures and administration. Although all schools had many requirements in common, almost every school had individual wishes. Often these wishes were even said to be strict constraints rather than optional preferences.

In the design of the module, especially the one of the memetic algorithm we use, we decided not to take into account all of the individual requirements of the schools. We decided that for a first version it would be sufficient to satisfy the global constraints. This keeps the design simple enough for version 1.0. Implementation of the individual constraints would lead to greatly increased complexity while at the same time failing to deliver substantial functional improvements for a large part of the target group.

All of the gathered requirements, global and individual, are contained in this document to make them available for later versions of the module. Global requirements can be found in Section 3.2.3 while the individual requirements are located in the Appendix 7.1. Please remember that only the global requirements are currently represented in the module's design while the individual ones are not.

3.2.3 Global Requirements

These are the constraints and preferences that are common to all schools of CBU. The design of the module currently only takes these requirements into account. At first lets take a look at the constraints. These have to be satisfied by the final timetable, they are *not* optional!

Constraints:

Prerequisite subjects have to be considered: Some subjects have requirements regarding prerequisite subjects. That means that a student can not register for a subject if he has not passed the required subjects. In that case, the module may not schedule this subject into the timetable of this individual student even if they are in the curriculum for his/her current term.

No subjects on Friday for final year students: Students who are in the final year of their program have to complete a mandatory project in order to receive their diploma. In order to give them enough time to work on their project, they are not supposed to have lectures on Fridays.

Only one 'difficult' exam a day per student: Some exams are known to have a high failure rate. In order to avoid putting students under to much stress, it is required that the generated timetable only contains one hard exam a day per student.

Maximum of 2 exams a day per student: It is only allowed to schedule a maximum of 2 exams a day per student.

Lecturer may not supervise the exam: A teacher/lecturer may not be the supervisor of the exam of one of his own subjects. The reason for this is to prevent cheating, as lecturers might be interested in having their students score high test results. Nevertheless, the lecturer is required to be present for the first 30 minutes of the exam so he can answer any questions that may arise.

Listed below are requirements by the CBU that are not mandatory to be reflected in the timetable. However, the generation algorithm will try to satisfy as many of these preferences as possible in the final timetable.

Preferences:

Only 1 hard subject a day: Some subjects are known to be harder than others, so only 1 hard subject should be scheduled per student per day.

Exams supervised by staff of school: Exam supervisors should be staff members of the school that organizes the subject.

Room exclusion: It should be possible to exclude rooms from the timetabling process, so they are not considered by the generation algorithm.

Time preferences for lecturers: Lecturers should be able to specify preferred day or time of teaching (morning/afternoon, evening)

Repeating of subjects should not result in overlaps: As mentioned in Section 3.1.2, students are allowed to repeat subjects that they did not pass the first time. This often results in overlaps where students have to choose to attend one subject or the other. This overlaps should be avoided if at all possible.

3.3 General Constraints

Apart from the functional requirements that have to be satisfied by the timetabling module, there are also other things to consider before and during the implementation process. Certain properties of the local (African) infrastructure can severely impact and restrict the desired functionality of the module, therefore the most important aspects that have to be considered will be illustrated in the following Subsections.

3.3.1 Computing Resources

Timetable generation using Memetic/Genetic algorithms is no trivial task. Designing a good, suitable algorithm is a time consuming and complex process. The same is true for the calculation of the optimal timetable by the algorithm. The calculation requires a lot of resources, in regard to the memory as well as the CPU.

Dictated by the architecture of the OPUS system, the calculation will take place on the server and not on a client machine. Likely there will not be a dedicated server for the timetable module, therefore a system serving many different services will calculate the timetable. Therefore it would be wise to calculate the timetable at a time where there is no high load on the server from other services; at night or in the weekends should be a feasible time to do the calculation.

3.3.2 Internet Access

Internet access in Zambia is very limited in terms of speed and bandwidth, so depending on where the system will be hosted this might cause problems. Therefore we strongly advise to host the whole OPUS system on a server within the local network, this way you can ensure a high speed connection.

4 Architectural Strategies

The Timetabling module will use a 'Memetic Algorithm'. A Memetic Algorithm is a search technique used to solve problems by mimicking molecular processes of evolution including selection, recombination, mutation and inheritance. The rationale behind using this technique is that constructing a timetable consists of selecting, recombining, and mutating the dates, times, and locations of subjects and exams. Furthermore, timetable construction is a 'NP-complete' problem[1], meaning that there is no known efficient way to find an optimal solution to the problem of timetabling. The time required to solve the problem to find the optimal solution using any currently known algorithm increases very quickly as the size of the problem grows. As a result, the time required to solve even moderately large versions of many NP-complete problem easily reaches into the billions or trillions of years, using any amount of computing power available today. As a consequence, determining whether or not it is possible to solve these problems quickly is one of the principal unsolved problems in computer science today.

4.1 Conclusion

Because it is impractical to consider enumerating all possible combinations we need to choose an approach which visits a subset of the solution space. In practice, it is often sufficient to find a good solution instead of an optimum. The challenge is to produce in a minimum time a solution as close as possible to optimal ones. We use a bio-inspired algorithm known as a Memetic Algorithm for this (For a more detailed description see Section 6.1).

[1] `http://en.wikipedia.org/wiki/List_of_NP-complete_problems`

5 System Architecture

This section describes the system architecture in terms of models and representations. First we will show how the timetabling problem can be represented. Then we will describe the database and class models needed for the implementation. Finally, we will describe the use cases for the system.

5.1 Problem representation

One of the main problems of representation is how one wants to combine dates (or periods), subjects/exams and available rooms. We have chosen to keep the periods dynamical and the subjects and rooms static. We made this choice because in a real system (implemented in a University), the total number of rules and subjects is statically determined. For example a University could have 150 rooms and 200 subjects, while the periods can range from one day, one week to one semester. We define the problem encoding as follows:

$$\begin{bmatrix} C_1 & \cdots & C_n \\ \vdots & \ddots & \vdots \\ C_m & \cdots & C_{nm} \end{bmatrix}. \tag{1}$$

Here, C_i represents a subject i, n is the total number of periods and m is the total number of subjects. If we keep the number of periods dynamical, a user can specify what a period is. You can have for example 40 periods in a week, 400 in a term, etcetera. At the start of the timetabling algorithm, the number of rooms and number of periods should be known. The algorithm then knows the exact size of the representation (for example, when you have 40 periods and 30 rooms, you have matrix with 1200 instances). Consider the following example:

- Physics with subject code C2 is given on Monday the first hour (from 8:00 till 9:00, period 1) in room 3 and Tuesday the second hour (from 9:00 till 10:00, period 10) in room 2.
- Mathematics with subject code M3 is given on Tuesday the second hour (from 9:00 till 10:00, period 10) in room 1 and Friday the last hour (from 16:00 till 17:00, period 40) in room 3.
- Business administration with subject code A1 is given on Monday the first hour (from 8:00 till 9:00, period 1) in room 2 and Friday the last hour (from 16:00 till 17:00, period 40) in room 4.

The representation would then be as shown in Figure 5.1.

Another thing that we have to model are the constraints and the preferences of specific teachers and schools. It is needed for our algorithm that we can calculate the fitness (or 'goodness') of a generated timetable, based on the constraints and preferences. As we have to deduce rules out of these constraints, we have decided that it is best to hard-code this in a so-called 'fitness evaluation function'.

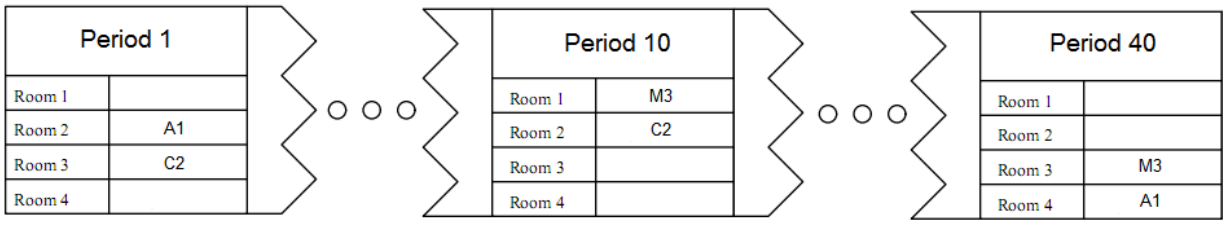

Figure 1: Example of a timetable in our representation

5.2 Database model

The data that was originally available in the OPUS system was not sufficient for our our Timetabling module. For example, there was no table that contained the available rooms on the campus. To keep this module 'plug-and-playable' we designed an extension of the original database model, in which we put the data needed for the timetable module. Our version is shown in the appendix in Figure 7.2. The model itself is pretty straightforward, but we will explain some specific things in it:

- It is a standard database UML model designed with Microsoft Visio 2007.
- Data needed for the input of our algorithm coloured blue. So this data need be available before our algorithm starts.
- Data that our algorithm will generate is stored in tables that are coloured green.
- The table 'subjectprerequisite' contains the subjects that must be preceded by another subject.
- The table 'room' is completely new. The field 'responsible' contains the person that is responsible for the room, in terms of key-holder, beamer-installation etcetera. The field 'type' contains the type of room. This can be 'presentation', 'standard' or 'special'. Where special means that it can be used for physics or chemistry. The field 'location' can contain the geographic data that is needed to represent the physical place of the room. This is an option for the future: this can be used for a routing algorithm that calculates the travel time between two different rooms.

The database tables need only be created once, when you 'insert' the timetable module. If the module is removed, the data and tables will still be there but not accessible from within the opus system.

5.3 Class model

The class model we designed needs some more explanation, it is shown in the appendix in Figure 7.3. It is a UML class model also made with Microsoft Visio 2007. Every class contains attributes

and operations on those attributes. To keep the design clear and simple we left out all trivial functions like getters, setters and database connectors. We will now give an overview of the classes, attributes and functions:

- `InputInitialData` This is the class where the algorithm should start. The `staffMembers[]` and `finished[]` arrays contain the references (ID's) to the assistant deans of every school, who are responsible for making the timetables. When all assistant deans have filled the database (coloured blue in the database model) with the appropriate data, the `runTimetabling()` function can be called, which initialize an instance of `MemeticAlgorithm`.

- `MemeticAlgorithm` This is the main function of the algorithm. It first starts by initializing the data needed for the algorithm, so we are working on a local copy of the data from the database. It then generates an initial population population and calls the `GeneticAlgorithm`. This algorithm takes the initial population of timetables and starts reproducing new ones (operations `selectParents()` and `reproduction()`). To keep the timetables evolving towards better ones, the algorithm modifies some elements in the population with a small chance. (This is done by the operation `mutate()`). After one run of the `GeneticAlgorithm`, the `LocalSearch` algorithm is called, which performs individual learning on every element in the population. This individual learning should be implemented by searching for a local optimum. When the `LocalSearch` algorithm is finished, the `MemeticAlgorithm` starts all over again. Finally, when enough iterations are performed, the `MemeticAlgorithm` returns the `bestElement` and launches the `Output` class.

- `Data` The data is represented in the same way as in the database. We have coloured the datatypes yellow that are already in the system. That is also why we did not show all attributes. Our new datatypes add new attributes and inherit the others from the existing datamodel, for instance `TimetableStaffMember` inherits from `staffMember`.

- `Population` The population is represented as a collection of elements. The `sort()` operation must override the normal sort() function in Java, because it must sort the elements based on their fitness.

- `Element` One single element represents a whole timetable for a specific period, modeled by a matrix (also see Equation 1). Depending on what kind of timetable the user wants, the subclasses `timetableElement` or `examTimetableElement` are used. The `NotScheduled` array contains the Exams or Subject that are not yet in the timetable matrix.

- `Output` Depending on what kind of timetable the user wants, the Output class fills the database (the tables coloured green in the database design) with the generated `bestElement`.

How to implement the specific classes and operations, like 'evaluateFitness()' and the memetic algorithm, is described in Section 6

5.4 Use cases and activity diagrams

We made several use cases and activity diagrams. They are shown in Section 7.4 of the appendix. Because the diagrams are pretty straightforward, we will only discuss the first one (see Figure 7.4). In this activity diagram, the whole system is shown. To make the timetabling process a success it is necessary that all Assistant Deans have added courses from their own 'organizationalunit' and that all teachers have edited the work times they prefer. When all this information is available for the system, the generating could start.

6 Detailed System Design

In this section there will be a more in-depth look into an essential part of the timetabling module; the timetabling algorithm. We explain the algorithm in text and in pseudo-code. We will also give an indication on how to create the fitness evaluation function.

6.1 Memetic Algorithm

Widely known by the general public for his popularization of science and his outspoken atheïsm, Richard Dawkins is also a renowned biologist, and the one who coined the concept 'meme'; the cultural equivalent of the gene. It is this concept, which stands at the basis of *memetic algorithms*, an active topic of research in Computer Science.

In Computer Science terminology, what is meant by the term 'Memetic Algorithm' is the combination of a generic Genetic Algorithm (See Section 6.2) with a Local Search algorithm (See Section 6.3) representing the 'cultural refinement' of memetics. Each memetic algorithmic step is in fact composed out of two different steps. First, the 'individuals' in a 'population' are evaluated according to a specific genetic implementation and its parameters, selected, and allowed to reproduce 'offspring'. This is followed by the feeding of (a part of) this offspring into the local search part of the algorithm, thereby further optimizing it. This cycle continues until a stopping condition is reached.

An overview of the Memetic Algorithm in pseudo-code is given below, together with some comments on how this can be reflected on generating timetables for the OPUS project.

```
Procedure Memetic Algorithm
   Generate an initial population; //Create a set of (valid) timetables
   while (Stopping conditions are not satisfied) do
       //Start genetic algorithm
       Evaluate all individuals in the population. //Compute fitness for all
       Select a set of individuals, sub;
       Create offspring from sub; //Create new timetables
       Offspring replaces the worst individuals in population; //Order on fitness,
```

```
                                                    //remove worst X in population,
                                                    //insert X offspring in population.
      With a small chance, do an additional mutation; //E.g., swap subjects
      //End genetic algorithm
      //Start local search
      for each individual in population do
          Perform individual learning; //Change all timetables
      end for
     //End local search
   end while
```

The Genetic Algorithm (See section 6.2) and Local Search Algorithm (See section 6.3) are described below.

Generating an initial population (meaning: An initial set of timetables) is the basic set used to generate new, and better, timetables. This can be done in several ways such as randomly picking subjects and putting them in timeslots in certain rooms, possibly with the constraints in mind. The constraints could be checked when inserting each new subject into an initial timetable. The quality of the initial timetables does not matter that much, but the allowing of breaching of hard constraints might make it more difficult for the algorithm to generate new timetables from the old (and wrong) ones.

The stop condition is the point at which the algorithm stops. For this we have 2 choices:

- Stop after a fixed number of runs
- Stop after a goob timetable has been found. Here, 'good' is determined by the fitness function.

6.2 Genetic Algorithm

```
Procedure Genetic Algorithm
       Evaluate all individuals in the population. //Compute fitness for all
       Select a set of individuals, sub;
       Create offspring from sub; //Create new timetables
       Offspring replaces the worst individuals in population; //Order on fitness,
                                                    //remove worst X in population,
                                                    //insert X offspring in population.
       With a small chance, do an additional mutation; //E.g., swap subjects
```

Genetic Algorithms derive their name from the fact that they are inspired by Darwinian natural selection mechanisms. Just like in real life, genetic algorithms work on a 'population' of 'individuals'.

Projecting a Genetic Algorithm onto the problem of timetabling involves some choices that need to be made:

1. How is an individual in a population specified?
2. How is an individual evaluated (more specific, what is the 'fitness function'?)
3. How are individuals selected?
4. How is 'offspring' constructed from a set of individuals?
5. How are individuals mutated?

1. To define an individual it is necessary to first explain what we want to do with the whole population (a set of individuals). The general idea is this: From a population of timetables (thus, an 'individual' is a complete timetable) select (see *3*) a few of the best (see *2*) available timetables (=individuals) and mutate (see *5*) them to create (a few) better timetables (see *4*). This process is repeated a fixed number of times, or more likely until a certain predetermined threshold of quality has been reached.

2. To be able to say anything about the quality of a generated timetable there needs to be a way to evaluate it. This is called the 'Fitness function', the digital version of the biological 'Survival of the Fittest'. The correct construction of the fitness function (see 6.4) is crucial to the proper working of the whole timetabling system! In regard to the timetabling system one can include positive weight in the case a timetable does not have much 'gaps' between subjects, has all lectures between 8:00 and 16:00, and can include negative weight to unwanted things such as breach of (soft)constraints, many 'gaps' between subject, very early or very late lecture times, etcetera.

3. From the population of timetables, a few of the best are selected. The selection process can be done in a few different ways. A 'selection method' is used to select which individuals from a population are allowed to combine their 'genetic code' (their timetable structure) to produce new timetables ('offspring', a technique called 'crossover'), thereby hopefully producing better individuals. The obvious selection method is to sort the timetables in the population depending on their fitness and use the first few with the highest fitness. This, however, can cause the algorithm to get stuck in a certain type of timetable; If the selected timetables all have, for example, 'Physics' on Monday the first hour, then it is very likely that all offspring created by combining two of these timetables also has 'Physics' on Monday the first hour. Thus, a certain degree of randomness in the selection process is needed.

In general there are three selection processes:

Random selection Simply select a set of couples of individuals randomly and have them produce offspring.

Roulette wheel selection Assign individuals a portion of an imaginary roulette wheel based on their fitness function value. The higher a individual's fitness, the bigger a part of the roulette wheel gets assigned to it, and thus the higher the probability this individual has to be selected for propagation.

Tournament selection This approach works by dividing the population into several, usually of user defined size, subsets $P' \subseteq P$. Out of each subset, only the best individual $p' \in P'$ is selected for producing a new generation. Each subset can thus be seen as little tournaments where the winner takes all, hence 'tournament selection'.

In our experience, the selection does not have a major impact on the results. The different strategies however are very easy to implement so creating all three should be no problem.

4. Generating offspring (=new timetables) from parents (=current timetables in the population) needs a strategy on 'how' to create these. Some possible strategies are described below:

Single Switching Take one timetable and switch a few subjects around.

Double Switching Select a few subjects from timetable1 and a few (other) subjects from timetable2 and combine these into a new timetable (and vice-versa; select all other non-selected subjects from timetable1 and the previously not selected subjects from timetable2 to create another new timetable).

Reschedule Remove a few subjects from a timetable and reschedule (put them somewhere else) in the same timetable.

Switch Periods Take a timetable and switch around periods (e.g., switch Monday with Tuesday).

For each of these strategies you must keep the constraints in mind!

5. Mutation is a process which only occurs some times in a run of the algorithm on one (or a few) timetable(s). The chance that it occurs can be specified and can be any value between 10% to 50%. A mutation is just that: A change in a timetable. Any change possible can be used; switch a subject, remove a subject, change rooms, switch days, and so on. Mutation is used to keep 'fresh' timetables within the population.

6.3 Local Search

The local search family algorithms concerns a large set of optimization strategies in Computer Science. It is popular, because it has the potential to approximate optimal values in a large search spaces in a reasonable amount of time, by comparing several objects in the search space with each other.

```
for each individual in population do
    Perform individual learning; //Change all timetables
end for
```

When applying local search you want to keep in mind the exploration/exploitation ratio: Already constructed timetables will get better because of the genetic part of the algorithm and you

want to take advantage of this (exploitation). On the other side you also want to create other, better timetables, which forces you to change parts of the timetable which might (or might not) be a good choice (exploration).

- **2-Opt**: In 2-Opt, an individual gets 'sliced' in the middle and recombined some other way. In the light of timetables this could mean splitting a timetable in the middle (monday becomes thursday, tuesday becomes, friday, wednesday stays wednesday), or splitting it at some other (possibly random point) to allow all days to change.
- **Hillclimbing**: This strategy is based on a per subject basis: Take each subject as they appear in the timetable and reschedule it (without breaking hard constraints) in the time slot which gives the highest fitness value. The idea behind this strategy is that by increasing the fitness on a per subject basis the end result (the fitness of the new timetable) will be better too.

Which strategy is chosen is left up to the developers, because it is impossible to tell which strategy will work best without trying.

6.4 Fitness Function

The quality of any timetable is determined by a 'fitness function'. This function has to keep in mind the hard constraints, soft constraints, and possibly the preferences of individual schools (See Appendix 7.1). It is important to create a good fitness function because the results of the algorithm depend highly on this function. The function should be able to give an indication of the quality of a timetable, in a short period of time. Speed of execution is very important, because this function will be called many many times. A slow function will make the whole module extremely sluggish.

- **Hard Constraints**: These are the constraints that should not be broken. Two tactics: Forbid timetables which breach these constraints, or give a high penalty when these constraints are broken, but still allow them. The decision which of these two tactics should be used has to be made during implementation, after an extensive time of testing.
- **Soft Constraints**: These are the constraints that should rather not be broken. Timetables breaching soft constraints should receive a penalty.
- **Preferences**: These are the preferences of the individual schools. Timetables which fulfill these preferences should get rewarded.

The height of penalties and rewards should be determined through experimentation. There is a choice between choosing either a 'fixed' value for all penalties/rewards, or a 'dynamic' value for each individual constraint or preference. Two *examples* of using this in a fitness function are given below. Note again that finding a proper fitness function is for a large part based on experimentation. Use these two examples as the name says; Examples.

H: Number of hard constraints broken.
S: Number of soft constraints broken.
P: Number of preferences fulfilled.
#h: Total number of hard constraints.
#s: Total number of soft constraints.

$$fitness(x) = \frac{H + S - P}{\#h + \#s} \tag{2}$$

where x is a timetable.

$H[i]$: Hard constraint i broken.
$S[i]$: Soft constraint i broken.
$P[i]$: Preference i fulfilled.
$hp[i]$: Penalty for hard constraint i.
$sp[i]$: Penalty for soft constraint i.
$r[i]$: Reward for preference i.

$$fitness(x) = \frac{(H[i] * hp[i]) + (S[i] * sp[i]) - (P[i] * r[i])}{\#h + \#s} \tag{3}$$

where x is a timetable.

7 Appendix

7.1 Individual School Requirements

7.1.1 School of Mathematics

General Information:

Timetable generation with Excel takes about 3 weeks at the beginning of the term. This school has no own rooms, so they use the facilities of the school of technology. If there is need for additional rooms during the term, finding an available one takes very long. Lecture blocks of 2 hours, each subject 4-6 hours a week. 1 term = 40 hours per subject. They don't make the TT for the first years (as curriculum is the same as other schools)

Constraints:

- No constraints specified

Preferences:

- Exams of difficult subjects should be in the morning
- No lectures on Friday
- Not 2 lectures of difficult subjects a day, preferably even a day between them
- Exams should be supervised by lecturers from own school

7.1.2 School of Business

General Information:

They have enough room for all their lectures/students. Currently their timetable runs from 7:00 to 19:30 (3 breaks: 10:00-10:30, 12:30-14:00, 17:00-17:30). They have an evening program (2 semesters) which runs from 17:30 to 20:40. Students register for each semester, first semester starts in January. They use half subjects (only 2 hours a week), while normal subjects have 4. Subject code identifies half subjects by ending with an odd number. They have difficult subjects identified by a high rate of failure of students in the exam. There is no problem with students repeating subjects, as most of the time the same teacher hold the follow up subject, therefore there are no overlaps.

Constraints:

- Lecturers are required to be on campus from 8 - 16

- Students should be in rooms of their own school. Who has keys for school rooms? One person per school.
- No lectures on Friday (last year students need time for project)
- Only one exam a day
- Prerequisite subjects have to be considered

Preferences:

- Lecturers should be able to specify preferred day / time of day (morning/afternoon, evening)
- Possibility to exclude a room from the timetabling process
- Exam supervisor should be provided by the school
- Not 2 difficult subjects a day
- The more students, the earlier the examination (time of day)

7.1.3 School of Technology

General Information:

This school does not have enough rooms for all its students/lectures, so it borrows rooms from the school of natural resources if the need arises. *Constraints:*

- Prerequisite subjects must be considered and may not overlap with their follow up subjects
- Additional attribute for the type of teacher. Professor supervises 2 exams, Lecturer 6.
- There can be 2 exams a day, as long as they are not both difficult subjects
- Final year students have no lectures on Friday (Time for project)

Preferences:

- Possibility to exclude a room from the timetabling process
- There should not be 2 difficult subjects on one day
- Lecturers should be able to specify preferred day / time of day (morning/afternoon, evening)

7.1.4 School of Built Environment

General Information:

This school has enough rooms for all their students, but they see increase in necessary space over time. Lecturers need to be available from 8-17. There are no especially difficult subjects to consider. They let the school of technology use their spare rooms. Lectures are in blocks of 2 hours. Full subjects: 3-4 hours a week. Half subjects ¡ 3 hours a week. No hard constraints on free days of common students, Friday afternoon free is preferred.

Constraints:

- Lecturer may not supervise the exam of his own subject, but he has to be present for the first 30 minutes to answer questions.
- Last year students have no lectures on Friday
- Not 2 difficult exams on one day

Preferences:

- Exams should be supervised by lecturers from own school
- Incidental exclusion of rooms
- Repeating of subjects should not result in overlaps
- Lecturers should be able to specify preferred day / time of day (morning/afternoon, evening)
- There should only be 1 exam a day

7.1.5 School of Natural resources

General Information:

This school needs to borrow rooms from School of Business and school of technology. Therefore they currently have to wait for the other schools to finish their timetables and then they can start their own. Half subjects: 2 hours a week. Full subjects: 4 hours a week. Lectures are held in blocks of 2 hours.

Constraints:

- There may only be one exam a day
- Lecturer may not supervise the exam of his own subject, but he has to be present for the first 30 minutes to answer questions.

- There may not be 2 lectures of the same subject a day per student

Preferences:

- Lectures of difficult subjects should be in the morning
- Repeating of subjects should not result in overlaps
- Lecturers should be able to specify preferred day / time of day (morning/afternoon, evening)

7.1.6 School of Graduates

General Information:

They use a semester system. 48 hours a subject (full time). Start may 10th, 3 months full time students, 4 months evening evening students. They only have 1 classroom. They seem to not really need an automated system, but would welcome it if it becomes available.

7.1.7 School of Distance Learning

General Information:

Their students only come to CBU when the other students are not there, only evening lessons. For a period of 6 weeks there are about 700 students. 3 bachelor programs and 1 diploma program (teacher education). Every school has its own evening program, students need to attend 85% of a subjects lectures. Lecture hours: 17:30 - 19:30, there are some clashes with School of Business. School of Distance Learning has its own rooms and a privilege on School of Business rooms in the evening. Their evening programs have the same exam timeframe as the rest of the schools.

7.2 Database Model

7.3 Classmodel

7.4 Use cases and activity diagrams

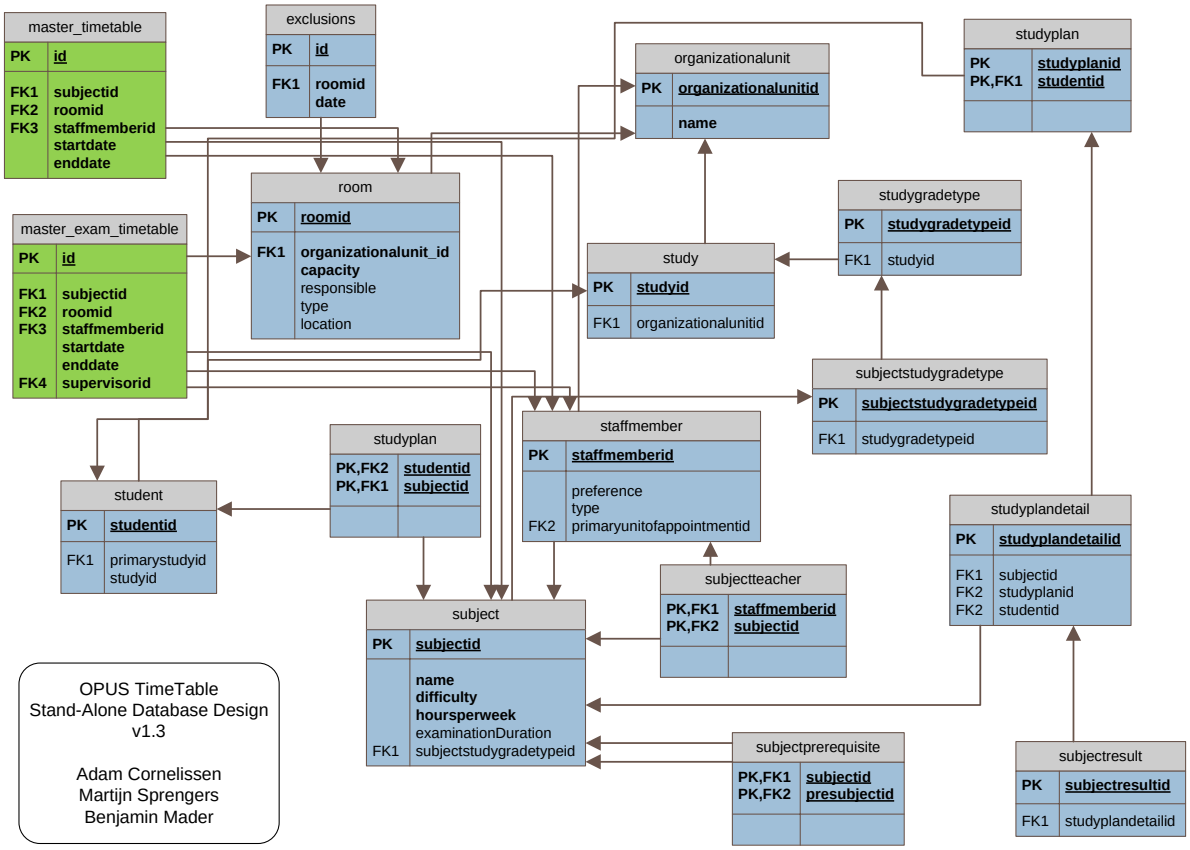

Figure 2: Databasemodel

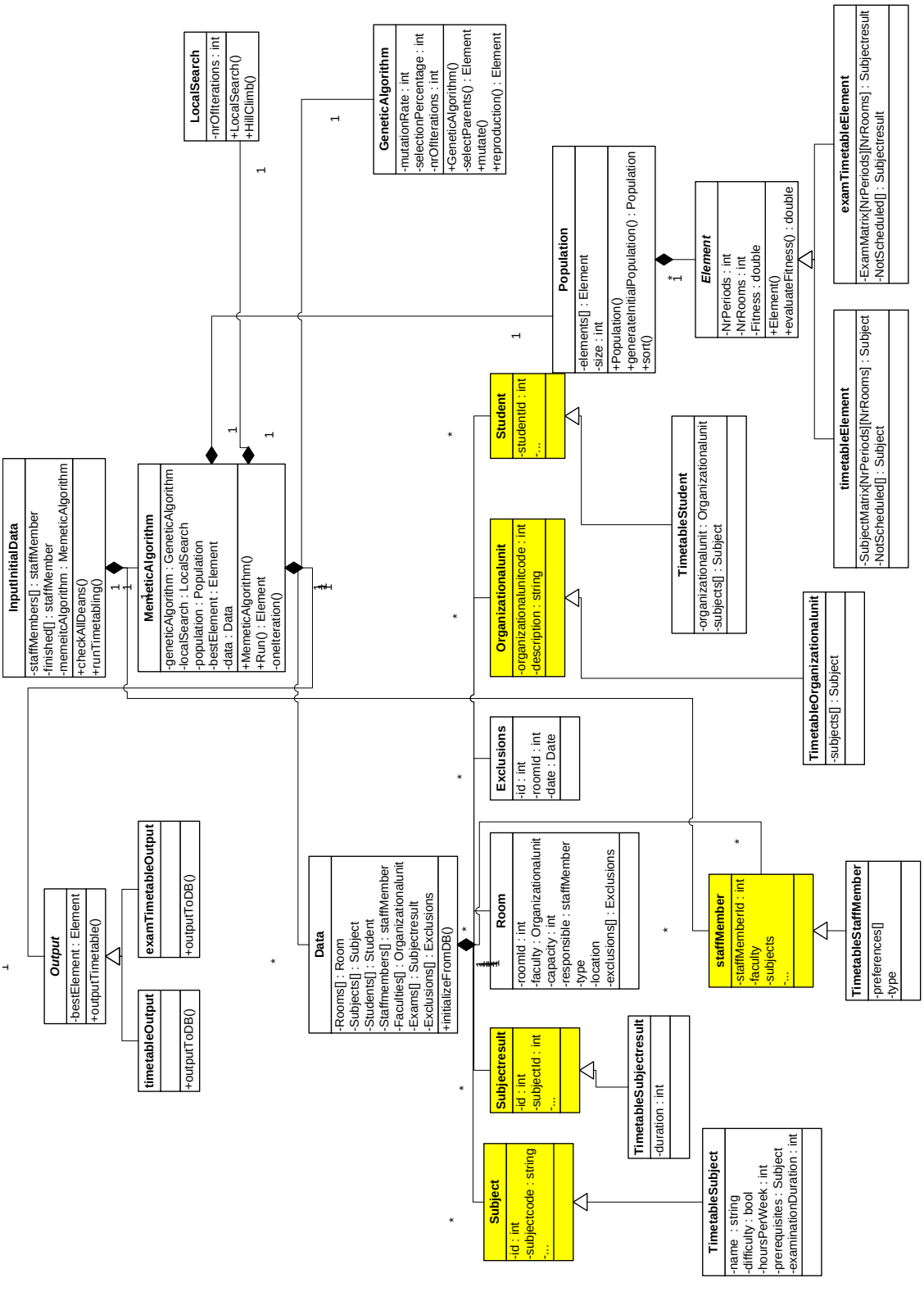

Figure 3: Classmodel

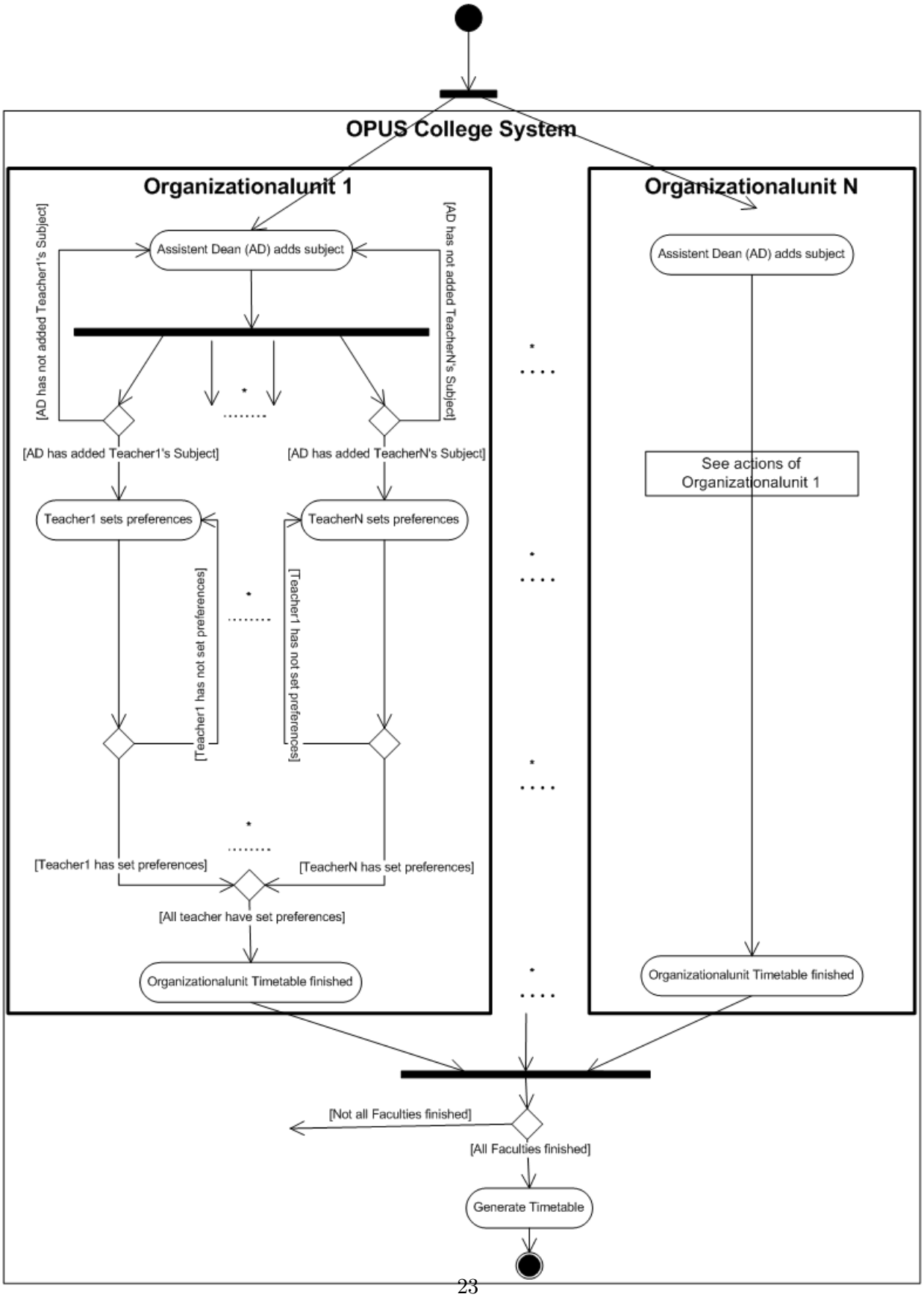

Figure 4: Activity diagram for the whole timetable module

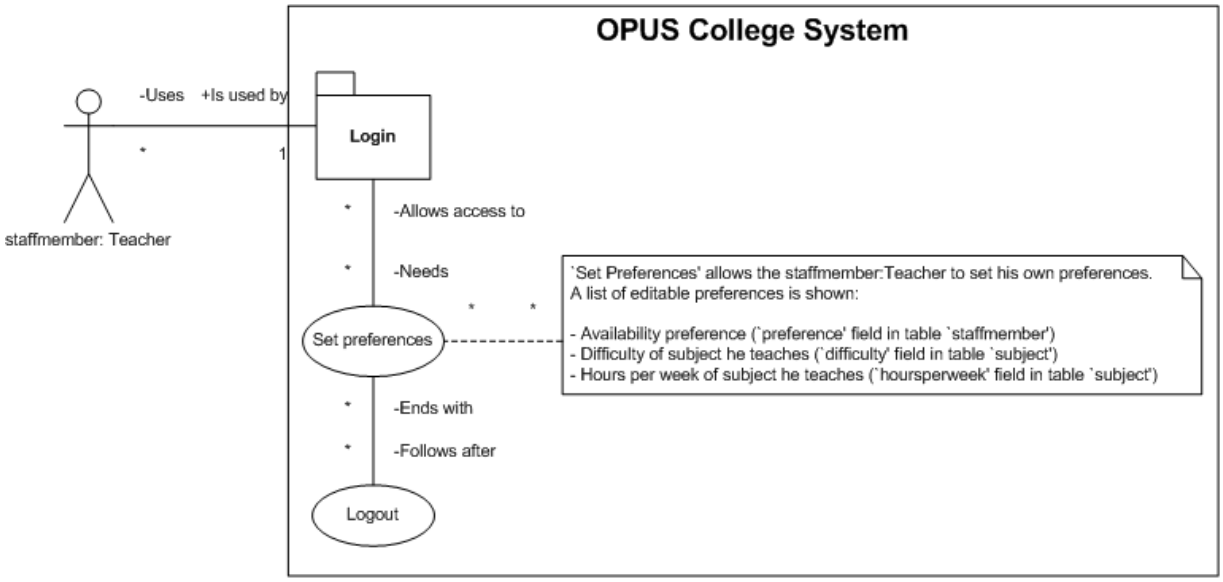

Figure 5: Use case for the teacher to edit preferences

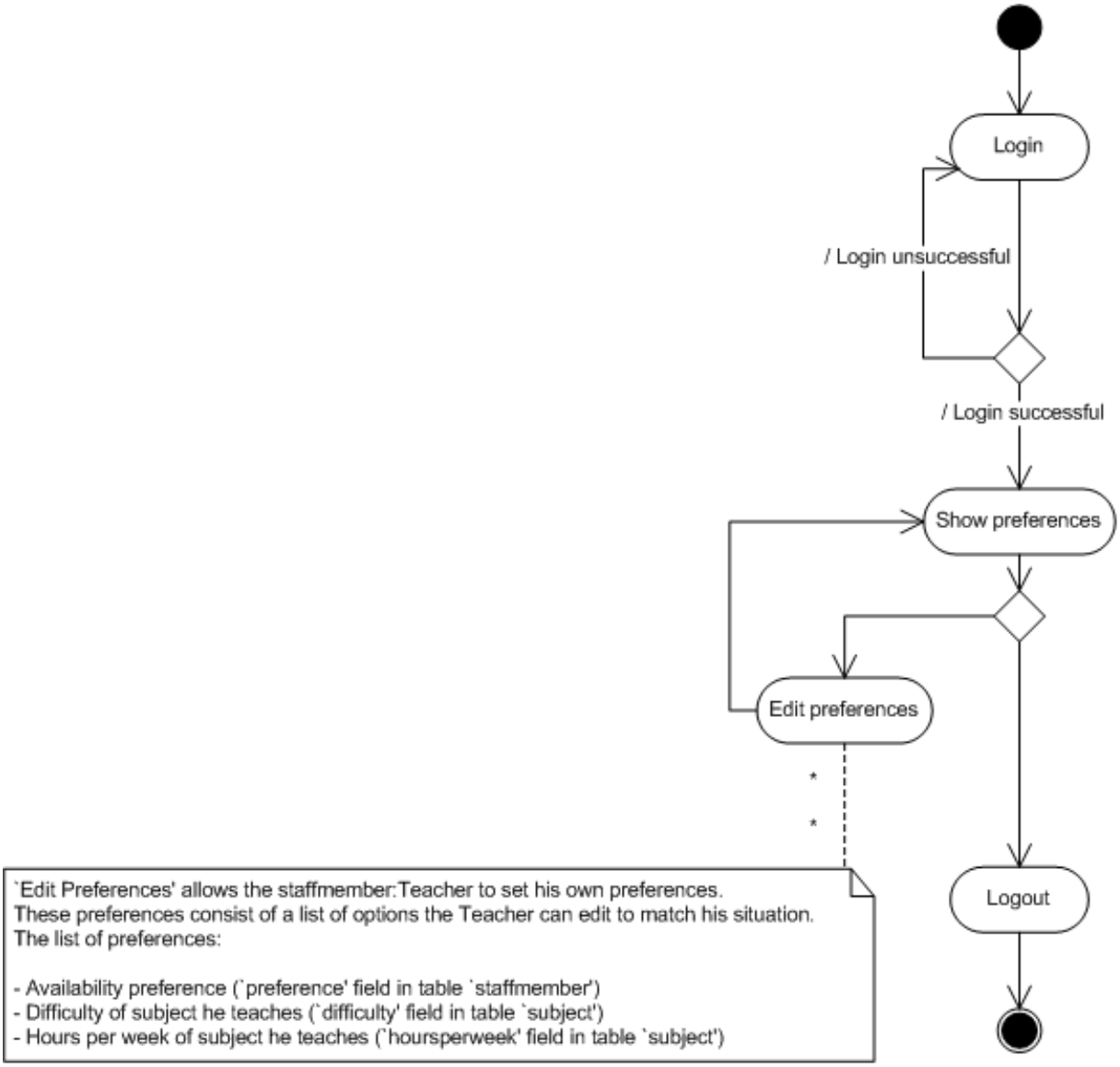

Figure 6: Activity diagram for the teacher to edit preferences

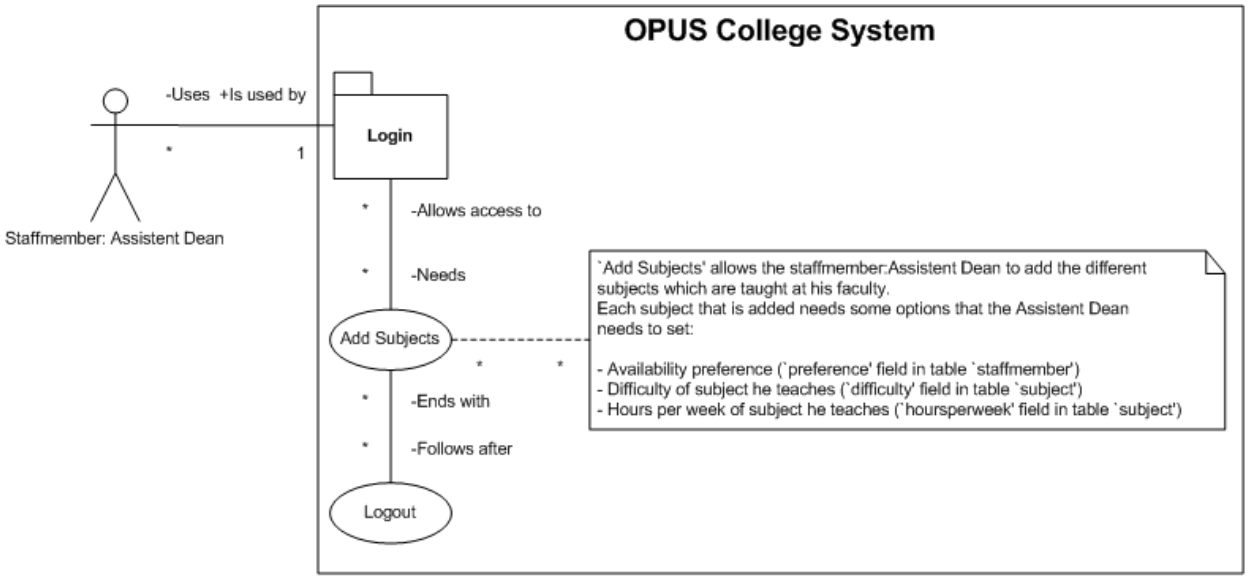

Figure 7: Use case for the assistant dean to add and edit subjects

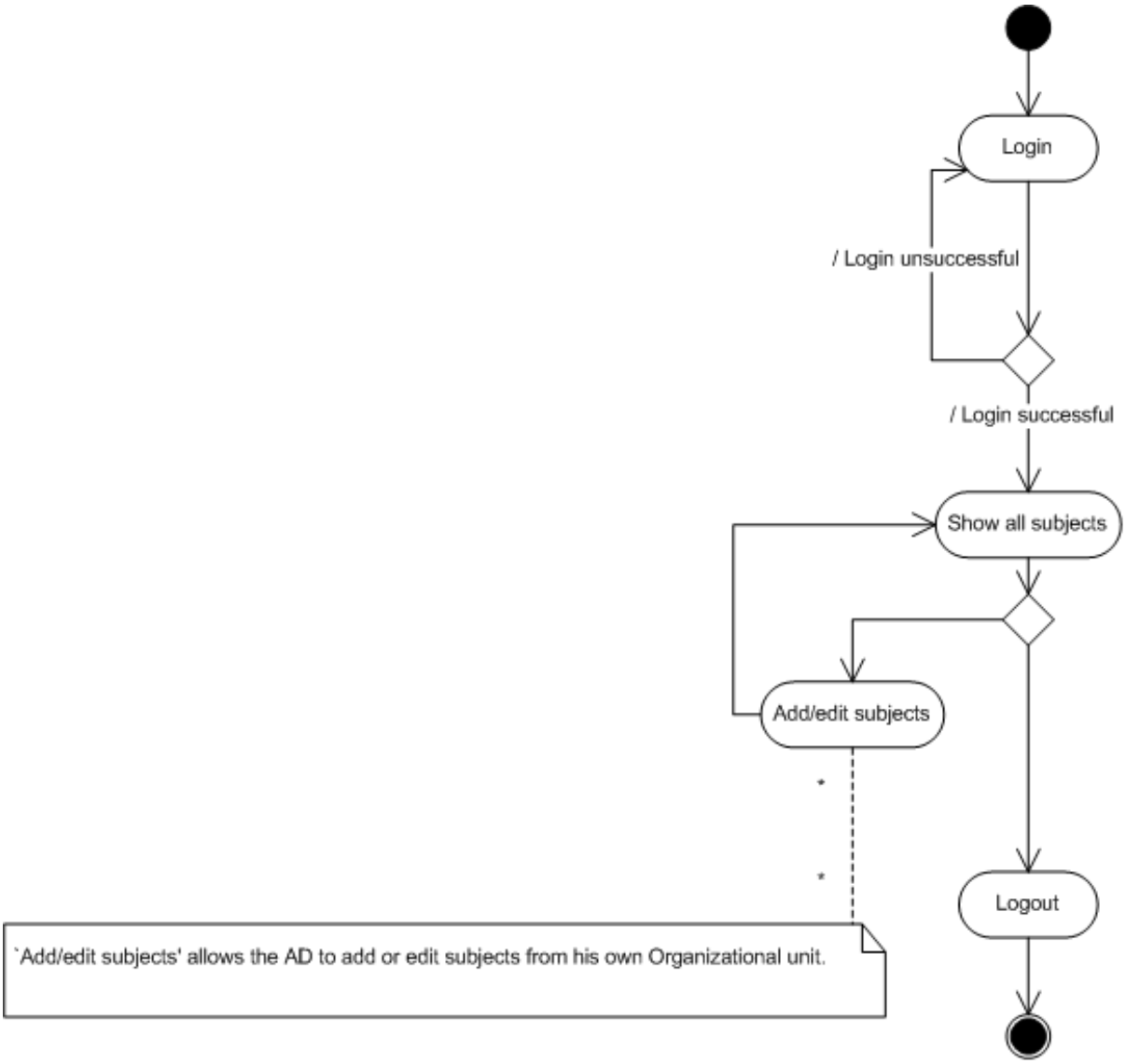

Figure 8: Activity Diagram for the assistant dean to add and edit subjects