S.C.O.R.E MyCourses

Summary report

David Bialik, Julian Fleischer, Hagen Mahnke,
Konrad Reiche, André Zoufahl

January 13, 2011

Free University of Berlin

Department of Mathematics and Computer Science
Institute of Computer Science

Contents

Abstract

This document summarizes the activities of a team of undergraduate students in the context of the student contest on software engineering [SCORE] 2011. The work was done on the project myCourses. It provides a description of the problem, the software development process and the product.

1 Team

We are a team of five undergraduate students of the Free University of Berlin. Even though we are all in the same year our knowledge regarding the creation of software and the used technologies varied widely. We were however all inexperienced in project management. Specifically none of us had been part of a software development process before. The members of the team, their age and their most prominent task are:

David Bialik (23) is responsible for the maintenance of the technical aspects of our developing process. He created our ant-based build system and an automated installer that sets up the database and a webserver. He also kept an eye on documentation and unit testing.

Julian Fleischer (23) is an enthusiast regarding everything that has to do with programming languages and the semantics of data. As advanced studies he attended a course on XML technologies. He specified and implemented an XML-based format for the creation of object-relation models, from which custom code can automatically be generated. The data access layer, as part of the backend of our application, has emerged that way, as have large parts of the web forms.

Hagen Mahnke (24), ~~being~~ interested in database technologies, and

Konrad Reiche (22), whos primary interest lies in algorithms and their efficient implementation, designed and implemented the scheduling algorithm. It works both as a stand alone application as well as an integrated component.

André Zoufahl (22) is working part time in a company doing web development, thus he is experienced in the usage of HTML, CSS and accompanying web frameworks. One of his major interests is the visualization of data, thus he was primarily responsible for the development of our user interface.

Figure 1: From left to right: David Bialik, Konrad Reiche, Hagen Mahnke, Julian Fleischer, Andre Zoufahl, Mister X

2 Development process

An important subfield of software engineering is the use of known methods and principles. The development process implies the used methods and principles. Thus this section will cover the chosen development process, its ideal procedure and how the development process actual took place including a timeline of events.

2.1 Intended development process

As mentioned before *MyCourses* is the first software project of every member in the team. The selection of a development process was heavily based on the software engineering course we heard in the summer 2010. A major part of the project was realized in the period of lectures. On this account we decided against an agile development process, as we did not have the opportunity to communicate and meet on a daily basis.

For the intended development process we chose the spiral model [1]. The combined approach of using an incremental and iterative model seemed to be promising. Dividing our project into iterations would give us a frame of orientation for the project management. The incremental working method was fitting as we were sure there will be no functioning version of the program after the first iteration. In general the spiral model focuses on solving those problems which reduce the projects risks at best. This fact reflected our expected work habits to adressing those problems which are blocking the project accomplishment the most.

We rather chose the spiral model as the basis for our own process and assumed that adaption would become necessary. We did not, however, have a clear concept of what might require adaption and how said adaption might be realized. Further we discussed essentials which should support and promote our work-flow. Key components of our process were therefore:

- **regular meetings** for coordination
- communication via **e-mail**
- intense evaluation of every iteration
- **ticket system** for definition and monitoring of tasks
- **wiki** for collaborative document creation

2.2 Actual development process

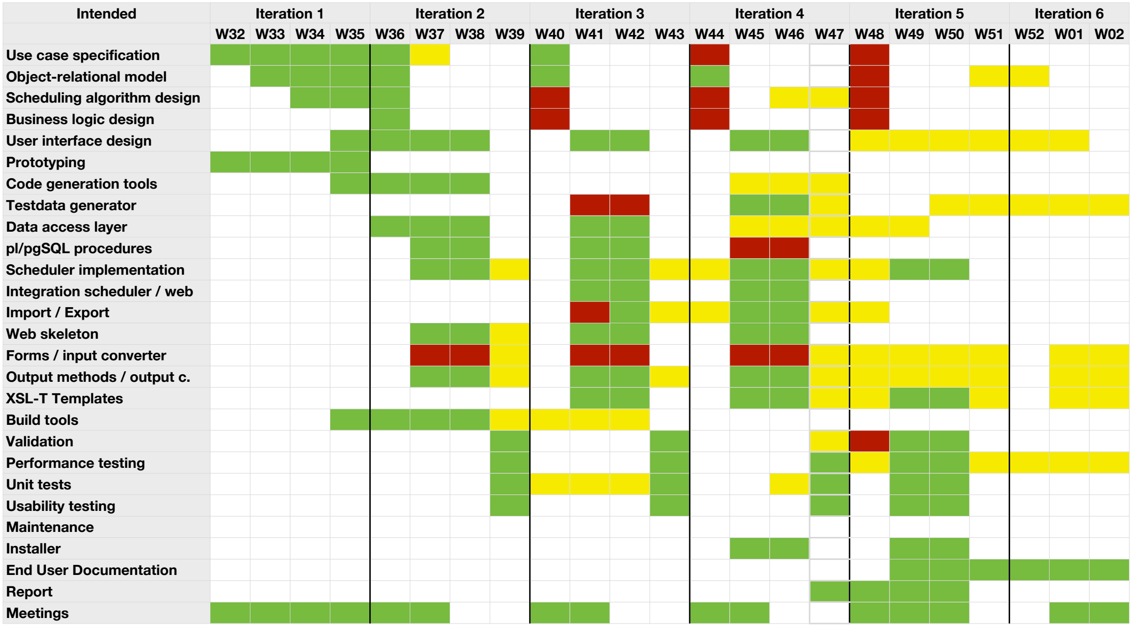
How the actual development process took place can be best understand when reading the timeline in section 2.3. In retrospect to the spiral model certain variations emerged.

The beginning of every iteration~~s~~ is characterized by detailed requirement definitions. An intense requirement elicitation based on the *MyCourses* specification was done in the first iteration with the result of a set of written use cases. In subsequent iterations, however, the requirement refinement was not done in the beginning of each iteration, but distributed over the iterations whenever problems with the current design emerged.

Even though we implemented a minimalistic prototype of *MyCourses* in iteration 1, there was no continuous prototype development for comparation and evaluation as proposed by the spiral model. We focused on incremental development of the final product in every iteration.

In contrast other procedures in the development process went according to the spiral model. We put alot of efford into the evaluation of different~~s~~ architectures and their resulting design. Further we tried to reduce the possible project risks by heavily discussing the next main goals for the iteration. These main goals were chosen in account to realize use cases and integrate different components as fast as possible.

In matters of organizing ourselfs and use of methods we had a steep learning curve. In August and September we selected superordinated goals, but in the middle of each iteration we often lost



green cells: components which we workend on as planned
yellow cells: components we worked on off our plan
red cells: components we did not work on although planned

Figure 2: Our project plan, originally conception and actual process

focus of important tasks. The biggest improvement of the development process was therefore the introduction of weekly task assignments for every team member since October. Based on that the tasks and their results were evaluated ~~in~~ every week, which lead to a purposeful work and fast discussion of emerged problems.

2.3 Timeline

A short overview of the course our project took will be given in the next paragraphs. The goals for a phase were defined after evaluation of the previous phase was completed. Before our project really started we got to know each other and decided which professor should be our contact person at the university.

First iteration started on June 20 and comprised of an initial requirement elicitation, design of our software, prototyping of the scheduler and choosing technologies. During most of this time the academic term was still running and only a fraction of our time was devoted to the project. This iteration ended on September 9 and was evaluated the day before. Much of the time spent in this iteration was still focused on gaining orientation. Still it was necessary for us to take this time and find a suitable approach to the work ahead.

Second iteration started on September 10 and was devoted to producing the scheduling-algorithm, the web-interface and our ORM as well as the servlet code. Work cycles were short and meetings were held every few days, including a weekend devoted to coding. From this point on our iterations were reduced to about one month each and the focus was on implementing and testing. Requirements and design were refactored whenever we felt the need to due to new insights or problems. This iteration ended on October 4 and was evaluated the same day.

Third iteration accordingly started on October 5 but most work was halted until October 16 due to exams and all-day courses at university. Beginning with October 26 weekly two-hour meetings were held at our university in which everyone reported achievements and problems of the last week, as well as set goals for the next week. These weekly meetings have proven to be a good way of coordinating work as well as motivate each other, so they were conducted for the remainder of the project. The main goal and achievement of this iteration was the integration of the scheduler with the web-interface and the connection to the database. Due to this employment of parts of our software many issues were found and fixed. Accordingly we shifted our focus further towards unit-testing and started using a tool to measure code coverage, that is how much of our code is actually being executed as part of a unit-test. At the same time more functionality was added to the web-interface. This iteration ended on November 1 and was evaluated the day before.

Fourth iteration started on November 2 and was primarily focused on the improvement of our web-interface, adding functionality and a unified design. Some improvements to the scheduler-algorithm were made, most notably the introduction of a greedy-algorithm for the set up of the initial population. Work on the summary report ~~was~~ also started in this iteration and the first version which was the basis for all further versions was written. The iteration ended on December 5.

Fifth iteration started on December 6 and was mainly devoted to minor improvements to the scheduler and writing the beta-version of our summary report. One improvement to our database-access-layer which made query-caching available drastically improved scheduling performance by a factor of 8.

Sixth iteration originially was not part of our project plan.

3 Requirements

The next two sections are a overview of the problem and the resulting requirements. The first section will state the problem, while the second describes the specification we deduced from it.

3.1 Problem Statement

1. Organizations like universities and schools are required to allocate courses to the given resources in a sensible way. As the amount of resources and courses grows large, it gets increasingly costly to do the allocation manually. Furthermore the task is repeated quite often and thus a lot of work is spent for it. Often the courses and resources remain similar each term and much of the previous solution can be reused. Each solution to such a problem must satisfy a number of constraints.

2. Usability: **TODO**

3.2 Requirement elicitation

Most of the requirements are already present in the project description of myCourses. While the need to find the requirements was mostly eliminated it is still helpful to be a more specific about the meaning of these requirements. To make the requirements more specific and better understood we created use-cases. These use-cases helped during implementation as the question of their realization was possible.

As we also desired more insight in how a scheduling process may look like, we asked an employee of our institute to give us a demonstration. The demonstration was conducted on the system used at our institute. While it did not reveal completely unknown aspects to us it gave us a better estimation of the importance of certain aspects.

3.3 Requirement specification

The ideal scenario is a program that automatically finds an optimal solution. However the problem is NP-hard and thus a computer-aided scheduling process is the focus.

The problem of course scheduling is defined by the given constraints which have to be satisfied. In order to meet the *MyCourses* specification requirements of high configurability we distinguish constraints by *hard* and *soft constraints*.

Hard constraints are constraints which have to be satisfied. If these constraints are not satisfied the lecturing of courses is not possible. This includes the following constraints:

- not more than 1 course in the same room at the same time
- the course lecturer has no other course at the same time
- courses belonging to the same year do not overlap with each other in time in order to guarantee studiability
- every constraint defined by the user with the priority of 100%, for instance preferred time, preferred room, etc.

Soft constraints are constraints which do not have to be ultimately satisfied. This includes only constraints defined by the user with a priority less than 100%.

An *optimal solution* is a scheduling which asserts the satisfaction of all *hard* and *soft* constraints. Respectively the schedules score for *hard* and *soft fitness* is both 1.0.

As the interests of many employees and students at the universities are affected by the result, they should be allowed to participate in the process or at least be taken into consideration. The requirements that struck us as most important are the following:

- automatic course allocation
- satisfaction of hard- and soft-constraints
- chance for manual allocations
- suited for multi-user environments
- scalable scheduling
- scheduling-related information is available
- definition of scheduling-related information is possible
- comfortable presentation to the user
- restricted access to information

These are the requirements that had the highest influence on our initial design and each subsequent change.

4 Architecture

Many areas of our project required decisions and a rationale behind it. These decisions are sorted by area below and each decision is explained shortly. In retrospect some of these decisions were less than optimal while others turned out great and a description of how these influenced our project and what we learned will be given. The first section consists of a short overview of the system and a graphic representation. The second section talks about our database-access-layer. The third section talks about our web-framework. The fourth section is about a collection of tools for automatic code generation. The last section describes the scheduler algorithm.

4.1 Overview

Early on we divided our software into two major components, which – seen individually – should be able to run as stand alone applications as well as as integrated components. This approach should give us the ability to work on different parts of the software more independently. Furthermore, since both components are largely decoupled, it would be easier changing one of them without breaking the other.

These two components are:

A scheduler which provides for an automatic allocation of courses to rooms and times.

A web application which is splitted into three tiers:

1. The front end
2. A controlling unit *(controller)*
3. The business logic

Both components should work on common ground, which emerges naturally from the need for both to work on the same data. Thus they were to access the same database and we decided to create a data access layer which should be shared by both components.

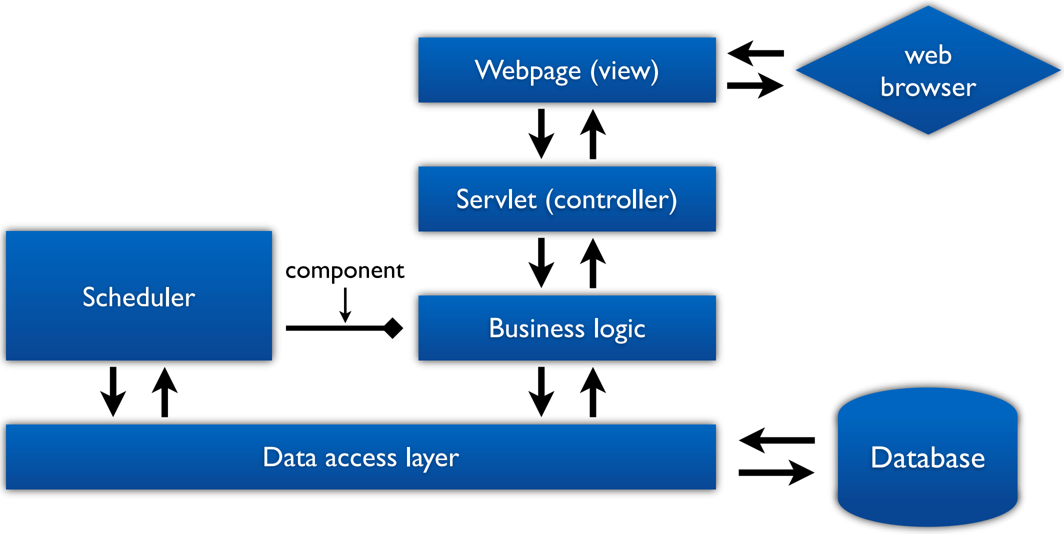


Figure 3: An overview of the components

4.2 Data access layer

In order to create a proper model of the entities we were to work with, we decided to develop an object-relational model, from which the object-oriented classes as well as an entity-relationship-oriented model should be derived.

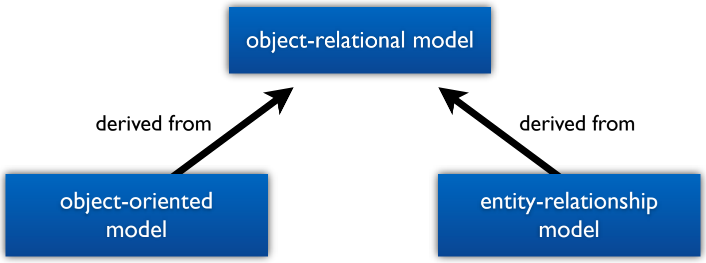


Figure 4: The object-oriented model and the entity-relationship oriented model is derived of the object-relation model.

We did this in order to keep both the OO as well as the ER model consistent, since we needed both of them in order to setup an object-oriented application as well as a relational database.

4.3 Scheduler

Since we split up into working groups focusing on different tasks, the scheduler was designed independent in respect to the rest of the components. The result is a monolithic scheduler which can be used by any software. A monolithic scheduler has the advantage to be decoupled and run on a different machine. As the scheduling process is a computation intense task this seemed to be fitting. In means of architecture an interface is provided in order to use the scheduler.

5 Technologies

Since our design was heavily influenced by the technologies we wanted to use, we will give a brief discussion of the technologies we used and why we chose to use them.

We considered many technologies and looked at some of them intensively. Since we were building a web application we considered PHP, Ruby, Scala, pure Java and AspectJ as our main programming language. As the scheduler is a performance critical part of our application we ~~soonly~~ dropped interpreted languages. As we were all most familiar with Java and development tools for Java are widely available, we favored Java. We finally went with **AspectJ**[1], which is built on top of Java, since it offers great flexibility through so called Aspects. It is tightly integrated in our favored IDE[2], Eclipse, since it is an eclipse project itself. We will discuss the use of Aspects within our application more in-depth later on.

As database backend we chose **PostgreSQL**[3] since it features strong adherence to the SQL standard and the best implementation of *referential integrity* in relational databases we knew of. Another strong argument for PostgreSQL was, that it allows for rich use of stored procedures, i.e. business logic within the database. As a matter of fact our application ~~thus~~ runs with PostgreSQL only, but could be ported to another database-engine easily, since only certain parts would have to be rewritten (PostgreSQL-specific parts within the data access layer and certain Aspects).

There is no choice about the lingua franca in the internet, thus we used web technologies like **HTML**[4] and **CSS**[5] in the front end. To keep things simple we stuck to related technologies which are all part of the **XML family**, like **XSL-T**[6]. We will disucss the use of XML within our web application in the appropriate sections of this report.

[1] `http://www.eclipse.org/aspectj` The eclipse AspectJ project
[2] Integrated Development Environment
[3] `http://www.postgresql.org/`
[4] http://www.w3.org/html/ – W3C HTML Homepage
[5] `http://www.w3.org/Style/CSS/` – W3C Cascading StyleSheets Homepage
[6] `http://www.w3.org/TR/xslt` – XSL Transformations (XSLT)

5.1 Tools

As mentioned in section 2, we made use of a wiki and a ticket system. Both of them are contained in **Trac**[7], a webinterface to **Subversion**[8], which we both used for source and version control.

TODO: Cobertura / JUnit; Ant / GNU Make; IDE Eclipse

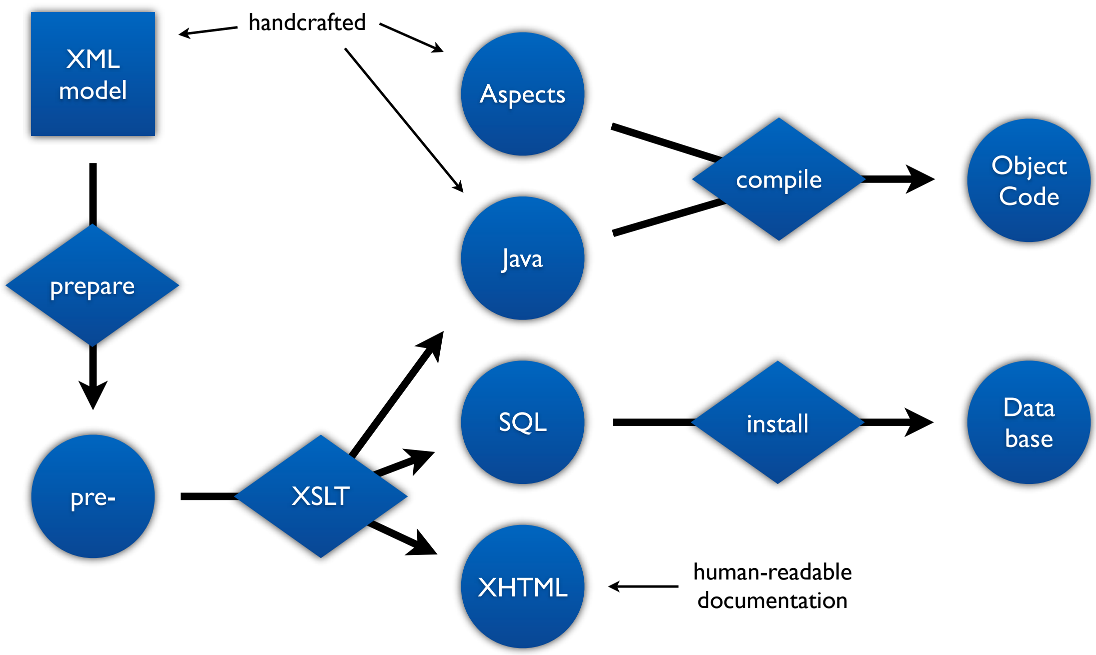


Figure 5: The process of generating code within scetris

6 Design

6.1 Object-relational model

One problem we have to face was that different universities have different syllabus of instruction. Our approach was to offer a system that could be made into anything. For instance the whole courses offered at university was difficult, since we want to reflect the reality as good as possible. Foundation was an abstract *Course*, identified by name, which can have several *CourseElements*. These we're no real courses a student could attend to. But you may create instances of them to get *CourseInstances*, which are connected to a year and the several *CourseElementInstances*, which have a date and time. This way a student can attend to a *CourseInstance*, but can only pass the *Course*, since it is unimportant for other *Courses* when he passed a required *Course*.

[7] `http://trac.edgewall.org` – The Trac project
[8] `http://subversion.tigris.org/` – Subversion homepage

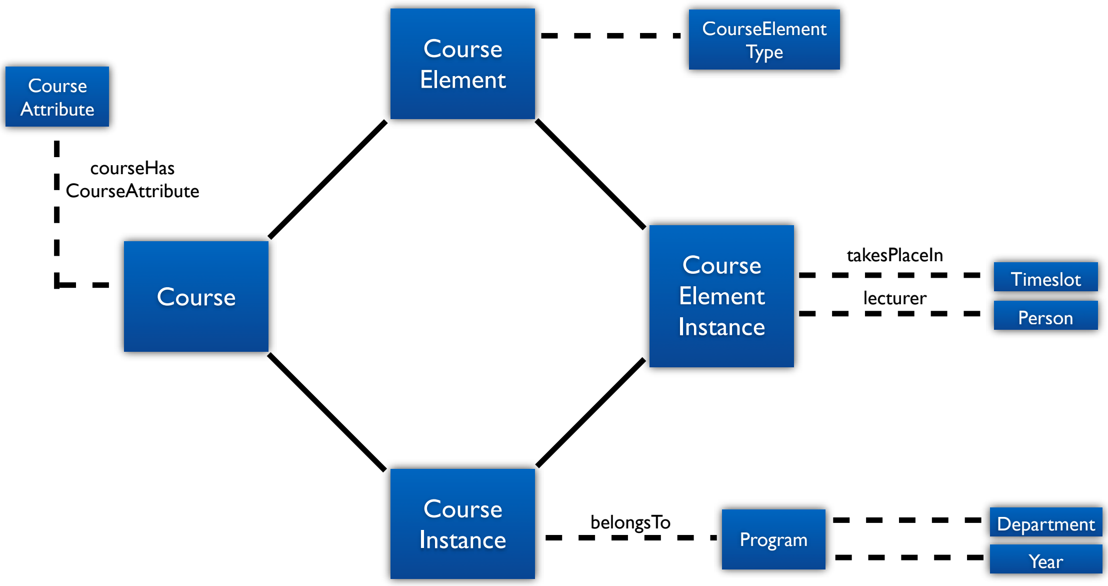


Figure 6: TODO

Another issue were information about a course, person or anything else that wasn't obligatory for the scheduling but for the application. As stated above every institution stores different data, so there is no *right* way to design the entities entirely predefined without making them big. Our idea was, to let the user decide, what a person, a course, etc. is. It's possible to define attributes a person can have, types of courses or various courseattributes. Moreover every courseattribute itself can be text, number or a boolean. This way, by default, we only store scheduling information and the user can steadily add content regarding metainformation.

A third aspect worth pointing out is our usermanagement. Basically it's a connection betweeen the persons who can log in, roles and different privileges. Every role can enable various privileges and every person can have several roles. In addition a privilege can directly be assigned to a person. It should not be possible to create or delete privileges, since they result from the defined actions and are therefore all predefined on startup.

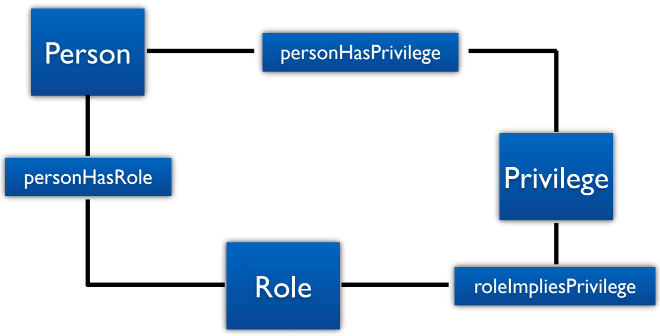


Figure 7: TODO

Note: Absatz zu features hinzufügen (Entity Feature - Constraints)

6.2 Design of the data access layer

Since our object-relational model is built around entities and relationships which somehow are entities themselves (some relationships are attributed) we decided to represent entities as well as relationships as individual objects. These objects are defined by classes which derive the interfaces Entity and Relationship. ~~The~~ both of them are data-base objects, which in turn are relations. Thus the object-oriented representation was named Relation.

The objects should both represent the data as well as control the communication with the database, that said, they should be *DAOs* (*data access objects*) and *DTOs* (*data transfer objects*) at the same time. To accomplish this, we designed them as *Active Records*.

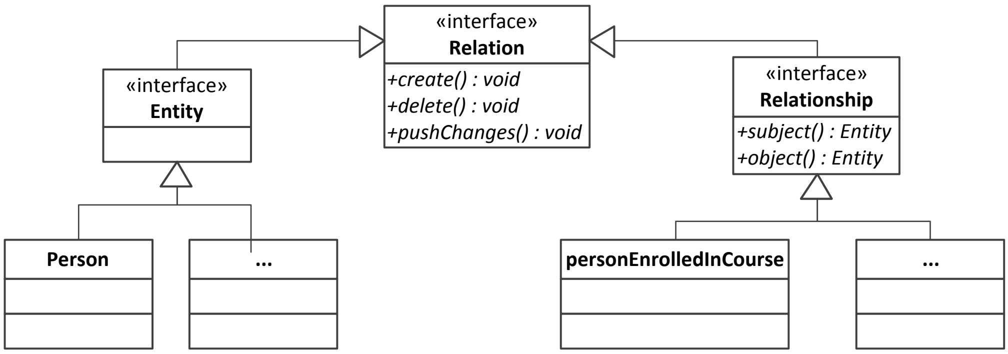


Figure 8: Overall structure of the data access layer

6.3 Design of the web application

Our web application is, as stated in **??**, layered into three tiers.

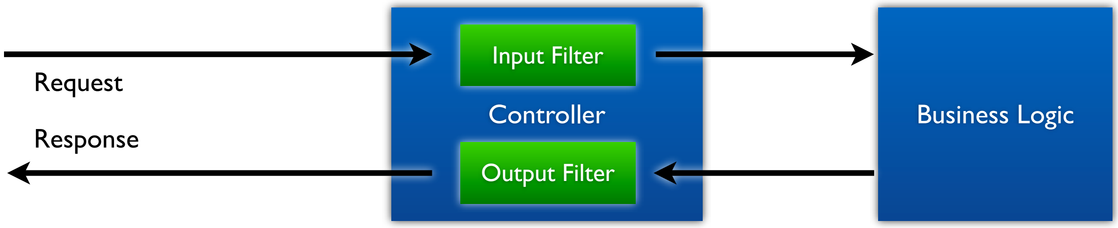


Figure 9: Data flow within our web application

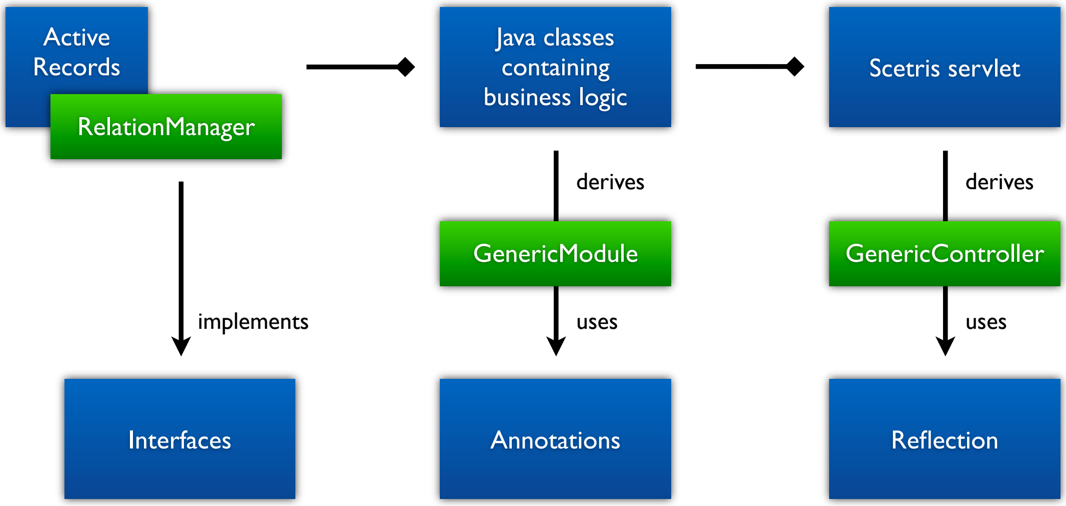


Figure 10: The process of generating code within scetris

6.4 Scheduler Algorithm

The problem of course scheduling is known to be NP-hard. The course scheduling problem can be approached by using search algorithms. As a matter of fact this works only for simple cases. Course scheduling, especially at universites and similiar big facilities, is faced with complex constraints. With the rise of input and more constraints finding an optimal solution cannot be computed ~~in a~~ sufficiently fast ~~time.~~ Based on several papers the approach of using a genetic algorithm is famous, seemed to be promising and was chosen therefore.

Genetic algorithm is a subset of metaheuristic optimization algorithm. Metaheuristics, being part of stochastic optimization, are algorithms using to some degree randomness in order to find optimal or as optimal as possible solutions to hard problems. Metaheuristics are applied when there is little to know about how the optimal solution looks like, there are too little heuristics to search on and brute-force is not questionable as the space of possible solution is too big. However when a candidate solution is given it can be scored in order to evaluate how good it is.

In order to understand the the scheduler algorithm the genetic algorithm operations, *setup*, *fitness function*, *crossover*, *mutate* and *selection* are best understood in means of core components of the scheduler algorithm. These operations are described below.

Setup generates the initial population of candidate solutions in a random or semi-random way. A candidate solution is created when every course is allocated in a room at a given time. The *setup* process is finished when λ candidate solutions were created.

Fitness function evaluates the candidate solution by scoring it. The score is mapped to the number of constraints satisfied. The more constraints are satisfied the higher the candidate solution is scored. The score reaches from 0.0 to 1.0.

Crossover creates a new candidate solution by mixing and matching parts of two given candidate solution. How the mixing and matching is done relies on the representation of a candidate solution. As our representation is a mapping from courses to allocated room and time, these allocations are mixed and matched.

Mutation creates a new candidate solution by taking a given candidate solution and changing a specified amount of course allocations to new, randomly chosen, course allocations.

Selection iterates the given candidate solutions and keeps only the μ best solutions. The solutions are selected, according to the score given by the *fitness function*, through dropping the rest of the candidate solutions.

We decided to model these parts as exchangable components, i.e. individual objects, which could be combined in any way. For this reasons we implemented the Factory method pattern in order to only create the components in a cohorent way. Since we did not have any experience with these kind of algorithms this design should help us to scale the different parts of the algorithm if changes have to be made.

Figure 11 illustrates the components interaction.

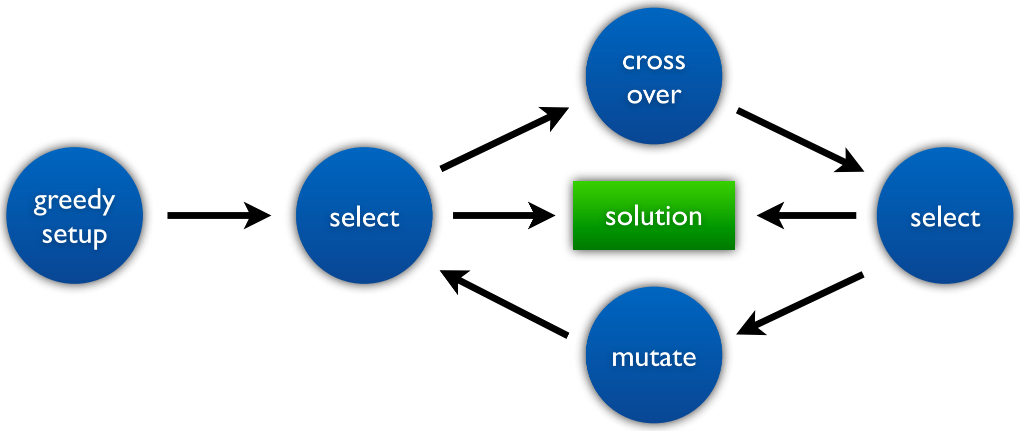


Figure 11: Routine of the scheduler algorithm. An initial population of candidate solutions is generated. Every candidate solution is scored. If non-solvable conflicts appear the scheduling has to be terminated by means of not being scheduleable. Otherwise the Setup phase is followed by optimizing the candidate solutions by applying crossover and mutation operation until an optimum schedule is found.

It is also possible the wished scheduling is not scheduleable at all. For instance when two course have the hard constraint to be placed at the same room with overlapping time. In this case the scheduling is stopped and the user has to resolve the constraint conflict by himself.

7 Implementation

The Implementation section will cover certain topics about how we solved non-trivial implementation tasks. Problems and their solutions will be discussed.

7.1 Data access layer

As we mentioned in "Architecture" (section 4), we did not want to maintain two separate models, but one. To achieve this technically we created an XML-file which represented an abstract description of our object-relational model. The syntax and semantics were defined using XML Schema[9]. Using different XSL-T Stylesheets we automatically generated Java source files, an SQL install script, and documentation in XHTML.

7.2 Web

7.3 Scheduler

The major problem of implementing a genetic algorithm is the question of how to represent the solutions. The presented genetic algorithm operations can only be applied when a fitting data

[9] `http://www.w3.org/XML/Schema` – XML Schema

model was designed. Historically a representation in the form of a fixed-length vector of values is used. The *crossover* operation applied on two candidate solutions would lead to mixing these values with each other. The *mutation* operation on one candidate solution would lead to randomly change values of the vector.

A generic approach to solve this problem is the encoding of the data model to a byte representation. But since we chose AspectJ respectively Java to implement the scheduler we wanted to use the concept of object-orientied programming and all its advantages. Fortunately we had given a reference solution which was implemented in C++ on which we based our implementation.

Instead of using a byte encoding we used a similarity to the vector. The candidate solutions are represented by using a *Map*.

$$Course \mapsto (Room, TimeSlot)$$

Every course is mapped to a tuple defined by a room and a time slot. The time slot is the starting time slot of the course [Figure 12].

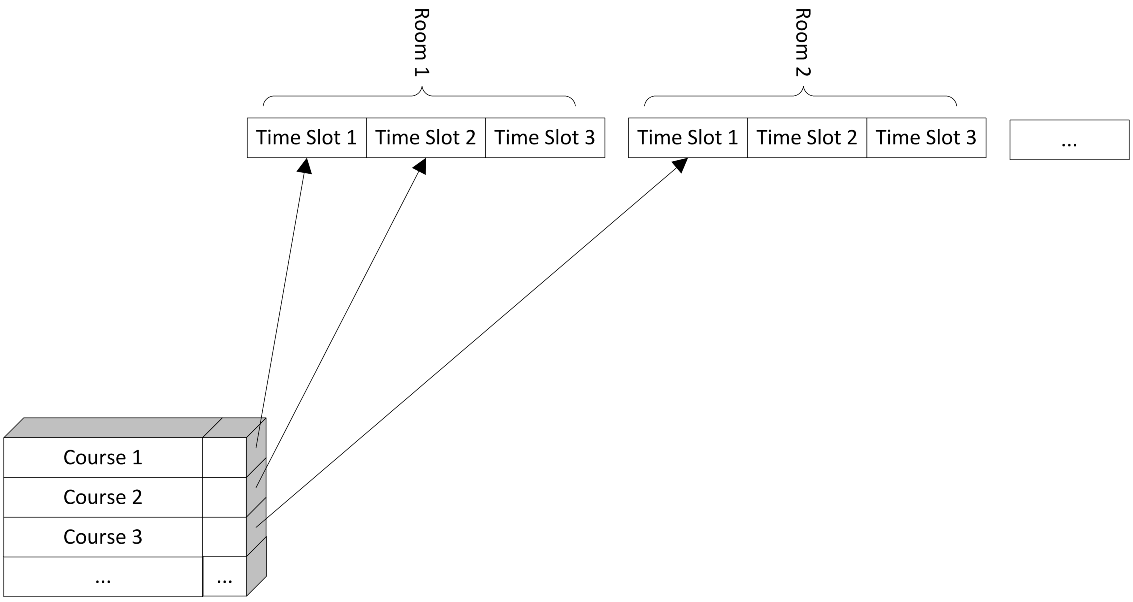


Figure 12: Every Course is mapped to a tuple defined by a room and a time slot.

As a matter of fact this is not sufficient to model the whole candidate solution. Behind this data model lies another data model modeling the whole timetable. The data model is a list of rooms with each having a further list with time slots. On every time slot is another list with courses allocated to this position. A list of courses was chosen because there is the possibility of having one course in the same room at the same time.

$$[(Room, [(TimeSlot, [Course])]$$

However applying *crossover* and *mutation* takes only affect on the *Map*. An example of executing *crossover* on the designed data model is illustrated in Figure 13.

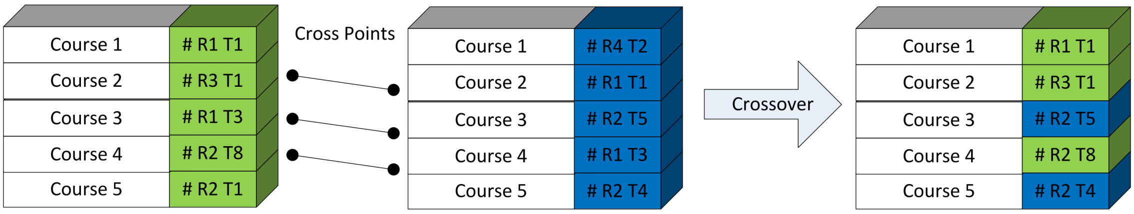


Figure 13: Crossover Operation: A specified amount of cross points is chosen randomly. The courses are iterated and taken over to the offspring candidate solution. When a cross point is met the other candidate solution being crossed over is used to take over its course allocations.

In the implementation phase it turned out using a *setup* allocating the course completely random works for little input very well. With increased input, however, the results by the classical *setup* lead to quiet worse results. These results are supposed to be optimized by the phase of applying *crossover* and *mutate*. As a matter of fact this happens to a certain degree but starts to converging to a certain score.

Therefore an alternative *setup* was implemented. *Greedy setup* combines the approach of a genetic algorithm using metaheuristic with the Greedy algorithm being a subset of the combinatorial algorithm.

Greedy Setup generates the initial population of candidate solutions by using a Greedy algorithm. Every course is allocated by placing it to the room which fits its constraints the best, for instance the requirement for a specified amount of seats. In order to avoid choosing rooms which exceed the required constraints every room is scored and sorted. In order to avoid overlapping in space and time the courses are place one after another in the timetable.

At the start of each scheduling a initial population containing λ possible schedule solutions is generated by *Greedy Setup*. Randomly chosen courses are placed into the room which fits the course constraints the best. In this step the time slots are chosen at the earliest possible time slot and are distributed along all days at the week. If, however, there is a constraint for a preferred room or a preferred time slot entered by the user this spot is chosen directly.

There is the possibility of finding an optimal solution in the phase of *Greedy setup*. Therefore the *fitness function* is applied on e[illegible]generated candidate solution. If a optimal schedule was found the algorithm has terminated.

8 The Application

Description of our product. Features and the concept behind the UI. Possible improvements and add-ons.

8.1 Usability

MyCourses is able to create a semi-automatic scheduling for programs at universities or schools. It can also provide a scheduling that was run fully-automatic but the recommended usage is to have MyCourses create preliminary versions of a scheduling. These will then be improved by the users via the functionality for collaborative scheduling also provided by MyCourses. This process is best used iterative through several cycles of automatic assignment and manual improvement. To get better results from the automated scheduling users can define requirements and assign those to courses or resources. A requirement that is assigned to a course is regarded as a constraint that either must be met in case of a so called hard-constraint or simply improves the rating of a scheduling in case of a soft-constraint.

~~In~~ Furthermore MyCourses provides functionality to

- enroll in courses

- view your own schedule
- view schedule for a room
- import and export data
- create, read, delete and modify data related to schedules

9 Verification and Validation

This section will cover testing and other measurements which were taken to convince ourself of the correctness of our implementations. But also performance measurement is part of Verification and Validation.

9.1 Unit tests using JUnit and automated testing using Cobertura

Since total code or even branch coverage was a too time intense task, we decided to concentrate the unit tests on only the most importang parts of *MyCourses*.

Testing the scheduler for correctness is a non-trivial task. Not only raises the presents of a genetic algorithm the complexity, but also does its random-factor make it hard to test conditions which have to be satisfied all the time.

One approach to improve the testability of the scheduler is the introduction of a seeded random generator. Whenever randomness is applied in the scheduler, it can be reproduced by using the same seed.

In order to achieve a high guarantee of the schedulers correctness JUnit tests were implemented as little as possible. There are unit tests for every component of the scheduler, for instance a unit test testing the *crossover* operation. On the downside unit tests allow only little input testing. Data-driven testing would be a more promising approach for testing the scheduler. On the other side this makes it also harder to check for correctness.

9.2 Performance and Benchmarking

Performance, especially of the scheduler, can be easily tested by test runs with varying size of data input. However this gives no detailed information about the actual performance of separate components. For that reason the profiler **JProfiler** was used to inspect the Java Virtual Machine while executing the program. This lead to a step-by-step process of picking up the slowest component display in the profiler and trying to improve its performance. In some cases this procedure gave us huge performance boosts.

A weak point of the scheduling performance was the scoring of candidate solutions. A lot of constraints and therefore *SQL Joins* have to be done, which results in a lot of blocking I/O which slows the scheduler down. We coped with this task by introducing a query cache.

10 Outcomes and lessons learned

10.1 Outcomes

We managed to accomplish the most use cases and fulfill the most requirements. Our final web application is ready to be installed with an easy to use installer. It is highly configurable and has a functional web interface. We used FLOSS software consequently and produced a lot of project- and other documentation of any kind along the way.

Beside our final application we also produced some side products which are worth to be mentioned. These include but are not limited to:

junction : JavafUNCTIONs you missed.

bakery : auto generated code reduces errors

weave : annotation based web development in Java

lego : xsl building blocks making web design easy

10.2 Lessons learned

For all of us this was the first project of this size and duration. We took the chance to put into practice what we had learned in our software engineering course. Especially things that are not applicable for smaller projects.

We found that project oriented approaches are hard to adhere to in an educational setting. Our team could not meet as often as we wanted and normal university business claimed much more of our project time than we had planned. Nevertheless we managed to establish a systematic meeting schedule. Afterwards it could be noticed that we should have made our iterations shorter in order to better keep track of tasks being postponed.

At the beginning the proposed architecture of our software changed several times. Even with the given set of requirements described. This was due to the hard to understand requirements. There where different interpretations of the described requirements. It took us excessively long to clarify the requirements and gain a shared understanding.

We had to undergo the experience, that a complex development environment has to be easy to set up. Thus the importance of automatic build tools is crucial or developing will be a pain for everyone involved. Another thing to mention is unit testing. It helps to detect errors and avoid them in code, so making your code base clean and correct. But it also takes a huge amount of time which could be used for implementing features. You have to balance the feeling that you do nothing useful against the advantages of unit testing which is not quite easy depending on the person developing.

Our team had not worked together previous to this project. We experienced that we are very different people with different approaches to problems. Near to the end of this project we are good friends. Everyone learned to get along with the originalities of the others. We have to highlight, that respect and honesty among the members of a team is very important and will hopefully lead to a productive development environment. The project gave us valuable experience on teamwork and how underestimated it is for the success of projects.

During the project we learned some new technologies and kept learning new aspects about well-known technologies. List of things we learned stated by the team members:

- XML, XSL
- Apache Ant
- aspect oriented programming and therefore AspectJ
- enhanced knowledge about web technologies
- deepened the understanding of the Java programming language and object oriented programming in general

11 Conclusion

A short summary repeating the process, and the product.

References

[1] Barry W. Boehm. A spiral model of software development and enhancement. *Computer*, 21:61–72, 1988.

[2] Xia Wang, Lin Huang, and Qing Zou. The Research on Multi-Strategy Course Scheduling Algorithm. *Information Science and Engineering, International Conference on*, 0:2419–2422, 2009.