Automated timetable generation for Egyptian schools

Bachelor's Thesis

S.R.P. van Hal
K.N.M.M.H. Osman

Delft University of Technology

TU Delft

Automated timetable generation for Egyptian schools

Bachelor's Thesis

by

S.R.P. van Hal
K.N.M.M.H. Osman

in partial fulfillment of the requirements to obtain the degree of

Bachelor of Science
in Computer Science and Engineering

at the Delft University of Technology.

Project duration:	April 29, 2018 – July 12, 2018	
Supervisor:	F.F.J. Hermans,	TU Delft
Thesis committee:	F.F.J. Hermans,	TU Delft
	O.W. Visser,	TU Delft
	M. ElHefnawy,	Key2Soft

An electronic version of this thesis is available at https://repository.tudelft.nl/.

Preface

This report concludes our work for the Bachelor Project of the bachelor Computer Science and Engineering at the Delft University of Technology. Over the course of two and a half month, we have created a system that automatically generates a timetable from a given set of timetable resources. This was done in collaboration with Key2Soft, an IT company in Cairo, Egypt, where we have worked on premises to complete the project.

Our time in Egypt has been challenging but at the same time educational and a very useful experience. We would like to thank the Key2Soft team for their supervision and for giving us the opportunity to conduct this project at their company. We would also like to extend our thanks especially to Mahmoud ElHefnawy, the CEO of Key2Soft, Elsayed Hamed, the CTO of Key2Soft and Reda Osman, the Key2Soft project manager.

We would also like to thank Felienne Hermans from the Software Engineering Research group at the Delft University of Technology for being our supervisor during this project, as well as her feedback, weekly meetings and guidance.

S.R.P. van Hal
K.N.M.M.H. Osman
Delft, July 2018

Summary

Cairo-based IT company Key2Soft is working on a comprehensive system to automate various systems in Egyptian primary-, middle- and high schools. This software system, named Key2School, includes a timetabling component, with which the company aims to relieve the workload of timetablers by providing them with a system which automatically generates timetables for all teachers, students and subjects. In consultation with the company, both functional requirements and timetable requirements have been composed for the timetabling part.

A literature study has been conducted to find and compare existing timetabling algorithms and libraries in order to select the best match for the company. All existing algorithms in literature were found to be too slow, so a system has been designed around an open source timetabling program. This system contains a part where the program is managed, a part which interfaces with the database of Key2Soft and a part where the timetable resources are constructed in a compatible manner.

The system has been implemented according to and in consultation with programmers at Key2Soft and will be integrated in Key2School in the future. The system is programmed mainly in C# and uses XML files to configure the timetabling library. The system has been thoroughly tested with NUnit, a platform-specific unit testing library, which enabled the developers to verify the code quality. The code has furthermore been evaluated by the independent IT consultant SIG.

Contents

1 Introduction

Each year, schools and universities all over the world create timetables for their students and teachers. To create a good timetable, a lot of factors play a role. Take for example the preferences of the teachers, the availability of classrooms or preferred starting times for certain subjects. The problem of creating an optimal timetable is well researched and known to be NP-Complete [1]. Currently, most schools create timetables by hand, which is a time-consuming process.

We have conducted our Bachelor Project at Key2Soft, an information technologies company located in Cairo, Egypt. Their focus lies on delivering value to clients by providing consultancy that solve their business problems along delivering innovative IT solution based on latest technologies that help them to achieve operational excellence[1]. Key2Soft is developing various software solutions for governmental and educational bodies in Egypt and Saudi Arabia.

One of those solutions is called Key2School. Key2School aims to automate a large number of processes within schools and Key2School therefore consists of a wide range of administration systems, financial systems, academic systems and two mobile apps. The academic systems, for example, connect teachers with students and parents, provide continuous access to the student's grades and supports direct messaging between teachers and parents.

For this bachelor project, an automated timetable generation system has been created as part of the academic systems of Key2School.

This report contains firstly, in chapter 2, the research behind this project. The analysis of the problem, translated into functional and timetable requirements, and a literature study has been conducted to find existing solutions to solve the timetabling problem can be found here. This chapter also dives deeper into the chosen existing timetabling solution.

Next, in chapter 3, the overarching software design is explained and the architectural choices are detailed. This chapter also describes how the data stored in Key2Soft data sources is translated to a compatible data format for the timetabling library.

In chapter 4, the actual implementation of the system components as described in chapter 3 is explained. The project structure, the timetable resource input implementation as well as the timetabling library output retrieval and storage are detailed.

Chapter 5 dives into the testing methodology which has been used to continuously verify the code quality. Creating maintainable and testable code has been a strong point of focus for this project. The test results are then interpreted and explained.

Chapter 6 contains an extensive evaluation of the requirements set forth in chapter 2 and the resulting timetables.

[1]see: `https://www.key2soft.com`

In chapter 7, the development process is explained and reflected upon as well as the interaction with the stakeholders. Also, recommendations are made for further work on the project.

Chapter 8 concludes this project and contains the key points to remember of this report and our time at Key2Soft in Cairo.

2
Research

2.1. Introduction

The first weeks of the project have been devoted to research. This chapter defines the (automated) timetable creation problem and the software- and timetable-specific requirements (or: constraints). Also, a stakeholder analysis has been conducted and their needs, who could influence the outcome of the project, are described. Furthermore, existing timetabling solutions and algorithms that can solve the automated timetable problem were researched to find a suitable solution for the problem. Finally, these solutions are evaluated and the choices about which solution to continue with are explained.

2.2. Problem definition and analysis

In this section, the automated timetabling problem is defined and the analysis of the problem is described. This includes concretizing deliverables, analyzing stakeholders involved in the project and determining the ethical implications of the project.

2.2.1. Problem description

Key2Soft is developing a wide range of software solutions for Egyptian schools called Key2School. The system is aimed primarily at primary, middle- and high schools. These schools make timetables for different people that are dependent on each other and should keep in mind preferences of each individual or group. Creating these timetables is currently done by hand, but this takes days or even weeks. Key2Soft aims to simplify this process by developing a system that can generate timetables automatically and thus solving the timetabling problem in a matter of seconds instead of days.

2.2.2. Deliverables

The main deliverable of this project is an automated timetable generation module, which can be integrated in the larger Key2School software package. The generated timetable has to adhere to functional (software) requirements (see section 2.5.1) and predefined constraints (see section 2.5.2) .

2.2.3. Stakeholder analysis

The stakeholder analysis in this section provides an overview of the stakeholders who are directly or indirectly influenced by this project. The stakeholders are categorized into primary, secondary and tertiary stakeholders. Primary stakeholders are the key players, who are directly influenced by this project and will actually use the system. Secondary stakeholders are only indirectly influenced: the project will influence their current situation, but they won't use the system directly. The tertiary stakeholders are not influenced by this project, but have a high interest in the success or failure of the project.

Primary stakeholders:

Key2Soft developers. The Key2Soft developers will integrate the system into the Key2School system. For this reason, the technical design and implementation has influence on how they can use our module. The developers will also continue developing and fine tuning the program.

Timetable creators / school administrators. The schools will use the Key2School system with the timetable generator integrated to generate timetables for the teachers and students, instead of doing this manually. Their requirements needs to be satisfied concerning the system on a user point of view.

Secondary stakeholders:

Students. The timetable will be generated for students. They will not use the system, but use the final output daily. Their preferences are important to be able to create a healthy educational environment. Their needs could be different for each group, or be the same for all the students. These preferences creates constraints for the creation of a timetable, as defined in section 2.5.2. The user (school administration) will specify the importance of each preference to satisfy as many as possible of the group. The students are the main focus for the user of the timetables.

Teachers. The generated timetable will also influence the teachers. The students are the first priority, but the teachers preferences are also an important aspects. For example, a teacher could prefer not to give lessons on a specific day. The module should take this into account. The user will also specify the importance of these preferences.

Tertiary stakeholders:

Key2Soft manager. The Key2Soft project manager will not use the module directly, but is the one that communicates the preferences of the school. He will eventually have the final say if the module is usable for the user or not. The manager has directly the most influence on the requirements of the module.

2.3. Ethical implications

Creating a timetable for different groups of people, such as teachers and students, does not create ethical problems per se, but requires some (ethical) decisions to be made.

First, it is important to realize that conflicting preferences can occur. When creating a timetable with soft constraints, violating some constraints is inevitable. The main ethical issue is whose preferences to satisfy and whose preferences to ignore. In consultation with the Key2Soft project manager, the decision has been made to center the system around students. This means that timetable constraints of the students have a higher priority than that of the teachers. If a constraint conflict occurs between two student sets or between two teachers, the system should randomly choose one.

Another ethical problem that can occur during the automated creation of timetables is the uneven division of workload over equally qualified teachers. One teacher could get scheduled for ten lessons, while the other would just get one. The timetable would satisfy all requirements, but would be unfair to the teachers. Fortunately, this problem is solved before this timetable system is used. In the user interface component, which is created by Key2Soft outside the scope of this project, the data is entered into the system in such a way that the user of the system explicitly defines the student sets that a teacher teaches. At this point, the decision about the teacher workload can be made beforehand to ensure a fair distribution of lessons.

2.4. Design Goals

In this section, the design goals which have been set for this project are described. A design goal is a guideline which highlights an important field or direction to focus on during the design and development of a product. During the definition of our design goals, we have attempted to find common ground between the consumer's and business needs and to find a compromise when the design vision clashes. Some goals are therefore only or mostly applicable to one side of this equation, but some are of equal importance to both.

The overarching design philosophy is based on – in their respective order of importance – compatibility, maintainability and performance. Each of these three categories is further elaborated on below.

2.4.1. Compatibility

The main deliverable of this project is a module which automatically generates a timetable based on a large amount of predefined resources, such as teachers and students (see section 4.3). The final product will be incorporated in the larger Key2School project, so it is important from the business perspective that the module is compatible with the existing code and coding practices currently in use.

Platform compatibility

Almost the entire code base of Key2Soft is written in C#. The main data storage behind the program is based on Microsoft SQL Server.

Key2Soft database integration

Key2Soft uses a MS SQL database for the Key2School system. This project will need to use this database to gain the needed information to generate a timetable.

Key2School compatibility

The output of the system will need to be used in the Key2School system. Therefore the system needs to generate output that Key2School can implement.

2.4.2. Maintainability

The system will need the possibility to change specific parts, such as adding or removing constraints. This should be easily be doable for the developers who will continue developing the system.

Code quality

The code needs to be clean and use the correct C# code style. Using continuous integration (AppVeyor) the system will always check if the newest code commits are working and correct. Furthermore, the code coverage will be checked by Coveralls to ensure each sub-part is tested. Also, Sonarqube is used to check the cleanness of the code. This will remove redundant and clumsy code usage.

Design patterns

Each part of the system needs to use the appropriate design pattern. Using these patterns creates a clear and understandable class model which makes it easier for the Key2School developers to reuse and change the code.

Submodule independence

Each part of the module needs to be usable on its own. Creating such a system gives the flexibility to reuse parts of the system.

Testability

The code needs to be structured in such a way to optimally support (unit) testing. Tests should furthermore give a clear indication of the correctness of each part and fail in a clear manner which describes the problem as well as possible for the programmer to resolve.

Documentation

All the code needs to be well documented so that the developers of Key2Soft can reuse and change our code without problems. The documentation needs to be grammatically correct, without any mistakes.

2.4.3. Performance

Fast timetable generation

The generation of the timetable should be done in a reasonable time, less than 30 minutes. The main user will not use the system if it takes to long for each generation.

Prevent overhead

While keeping code maintainability. Will have to weigh the importance of both every time we make a decision.

2.5. Requirement Analysis

The requirements for this project have been analyzed in collaboration with the Key2School project team at Key2Soft. The requirements have been divided into two categories: first, we will discuss the requirements of the module to be developed. Secondly, schools have a lot of rules for timetables (or: constraints), ranging from which teacher can teach which course, to specific room-course relations, class splitting and shared courses. These timetable constraints are detailed in section 2.5.2.

2.5.1. Module requirements

The module requirements for the project are structured following the MoSCoW method [5]. This method enables us to distinguish and prioritize the most important requirements. Each requirement is defined following the requirements of the stakeholders, and the design goals.

- The module *must* be able to produce a feasible timetable, given that the input data allows a feasible timetable to be constructed. A feasible timetable means that a timetable is generated without any predefined hard constraints violated.
- The generated timetable *must* adhere to predefined constraints as detailed in 2.5.2.
- A feasible timetable *must* be generated in less than 30 minutes.
- All classes and methods *must* be documented, explaining its use or input, behavior, and output.
- Users *should* be allowed to set the importance c.q. priority of each soft constraint by intervals of 10, where 10 is an almost hard constraint (99%) and 0 is a not needed (or inactive) constraint.
- The module *should* be able to be integrated seamlessly in Key2School.
- The module should load all relevant information to generate a timetable from Key2Soft data sources.
- The module should be able to generate a timetable without user interaction or user intervention.
- A feasible timetable *should* be generated in less than 5 minutes.
- The module *should* have at least a 90% branch coverage by using unit tests.
- The module *should* be tested with real-world timetable cases.
- Violated soft constraints with a priority of more than 50% *should* be reported to the user.
- The module *should* retain all generated timetables.
- The module *could* inform the user that, when no feasible solution is found, which constraints 2.5.2 are violated.
- The module *won't* support manual modifications to a generated timetable at this time.

2.5.2. Timetable constraints

During the first weeks of the project, the requirements with respect to the timetable have been changed multiple times. The original problem definition (see A.1) contains only a few (trivial) constraints. At the start of the project, numerous additional constraints were added. Then, after a Key2Soft meeting with an expert school timetabler, additional constraints were added as well as the distinction between constraint types.

A constraint can be either hard or soft. A hard constraint means that this constraint cannot be violated, otherwise the timetable is incorrect. An example of a hard constraint is a the weekend. During these days, there can not be a lesson scheduled. A soft constraint is a constraint that can be violated if necessary, but would be preferred not to. Soft constraints are usually based on teacher or student preference, while hard constraints are usually based on student or teacher availability. For example, teachers would like to have a evenly distributed schedule, without big empty gaps. But if the hard constraints makes it not possible to always retain this constrain, the soft constrain will be violated. The number of violated soft constraints will tell the quality of the timetable, less violated soft constraints is a better timetable.

Some of the constraints are final, which means that the constraint is always hard or soft. Other constraints can be either one of them. This can be set in the settings of the user, before generating a timetable. Each soft constraint will have a weight, ranging from 0% till 99%. The weight will represent the priority of each soft constraint, regarding which one to break if a soft constraint needs to be violated.

Terminology

The terminology, categories and description of timetable constraints are paraphrased or taken directly from the extensive survey of school timetabling research by Pillay [20].

- A class refers to a group of students that will be taught a particular subject, e.g. Mathematics.
- The term grade refers to the level of schooling, e.g. twelfth grade is the last year of secondary schooling. Each grade contains one or more classes.
- A subject refers to the actual content being taught, e.g. English or Mathematics.
- A lesson refers to a particular subject being taught to a class by a teacher. A number of lessons are taught for each subject.
- A period is a timetable slot which a lesson can be scheduled in.
- An idle or free period for a teacher is a period in which the teacher does not teach. Similarly, an idle or free period for a class is a period in which the class is not taught a lesson. A timetable without free or idle periods is described as compact.
- In some cases classes are split into subgroups and each subgroup is taught a different subject simultaneously in different venues. Similarly, it may be necessary to merge classes for lessons and teach these in one large venue.
- A tuple is comprised of a class, teacher and a room which must be scheduled in a timetable period. In some cases a tuple may consist of only a class and a teacher. In addition to this a class may be the result of a split and/or merge. There may also be more than one teacher in a particular tuple.
- A resource refers to any entity involved in a lesson. The standard resources are a class, teacher and the room in which the lesson is held.
- Type defines if the constrain is hard, soft, or could be either of them.

Problem requirement constraints

CODE	REQUIREMENT	TYPE
PR1	Classes must be scheduled for the required number of meetings for each subject.	Hard
PR2	Teachers must be scheduled for the required number of meetings with each class.	Hard

No clashes constraints

CODE	REQUIREMENT	TYPE
NC1	A class must not be scheduled more than once during a time slot.	Hard
NC2	A teacher must not be scheduled more than once during a time slot.	Hard
NC3	A room must not be scheduled more than once during a time slot.	Hard

Resource utilization constraints

CODE	REQUIREMENT	TYPE
RU1	Teachers must only be scheduled when available. Teachers may be unavailable during certain periods due to administration tasks, teaching in another school, allocated free periods or days off. Teachers must not be scheduled to teach during these periods.	Hard/Soft
RU2	Classes must only be scheduled when available.	Hard/Soft
RU3	Certain lessons require specialized rooms, e.g. science labs, computer lab, the gymnasium. These requirements must be met.	Hard

Workload constraints

CODE	REQUIREMENT	TYPE
W1	Classes for a particular grade cannot have more than a predefined number of periods per day, e.g. the timetable for lower grades may be comprised of 9 periods while that for higher grades may schedule 11 periods.	Hard
W2	A limit may be set on the maximum and minimum number of lessons of a subject a class may have per day, e.g. certain grades can not have more than three Mathematics lectures in a day.	Hard/Soft
W3	Teacher workload is defined in terms of a minimum and maximum number of teaching lessons or hours per week or per day.	Hard/Soft

Period distribution constraints

CODE	REQUIREMENT	TYPE
PD1	A limit may be set of the amount of consecutive lessons for a subject per day.	Hard/Soft
PD2	The lessons taught by a teacher should be well-spaced throughout the week.	Soft
PD3	Certain subjects may have to be taught every day.	Hard/Soft

Lesson constraints

CODE	REQUIREMENT	TYPE
L1	Classes may be merged together for a lesson. In some cases classes may have to split into subgroups with each subgroup being taught a different subject simultaneously. The split subgroups may also need to be merged differently from the original configuration.	Hard
L2	Certain lessons must take place simultaneously.	Hard

2.5.3. Analysis

The automated timetable generation problem is NP-hard. Therefore, no (polynomial) algorithm exists to find the optimal solution. A wide range of heuristic algorithms to solve the problem have been presented in literature, each with their own advantages and drawbacks. The most important algorithms are covered in section 2.6. It is critical to select the right algorithm to meet the time limit of the timetable generation and to be able to create a feasible solution at all. Also, not all algorithms are able to work with specific constraints on resource availability.

Furthermore, because this module is part of a larger system, we have to take the implementation in the final product into account. We have to either adhere to the existing project structure, or create and independent module with a compatible interface to the system.

2.6. Existing Algorithms

Extensive research has been performed in the field of automated timetable construction. The problem was first formulated in 1965 [6] and was proven to be NP-complete in 1995 [1]. Multiple different approaches using one or more heuristic algorithms have been proposed since, each with its own advantages and drawbacks. A meta-heuristic algorithm is a algorithm that makes a couple of assumptions before searching for a near optimal solution of a problem. In this section, we will present a wide range of existing algorithms to solve the automated timetabling problem.

2.6.1. Genetic Algorithm

A genetic algorithm is a heuristic approach to find a (sub-)optimal solution for problems that are not solvable using linear computation algorithms. The genetic algorithm is an evolutionary algorithm and is inspired by Darwin's human evolution theory. The algorithm encodes candidate solutions as chromosomes. Offspring of two chromosomes combines characteristics of both and random mutations can happen within a chromosome.

The fitness value of a chromosome denotes how good or feasible that solution is. First, an initial, random population of chromosomes is created, each with a different fitness. To allow the algorithm to converge to a (sub-)optimal solution, selection is used to choose the chromosomes that have the best fitness which then reproduce to form a new generation of chromosomes.

Generating new chromosomes is based on the biological principle of crossover. Crossover swaps parts of the parents' chromosomes to create new offspring. For each generation, a new selection is performed to choose the best available chromosomes.

The algorithm terminates when a predefined fitness value is reached or after a maximum number of generations. To make sure that the algorithm does not get stuck in a local minimum, the algorithm uses mutation, meaning that at a random point in a chromosome a random mutation (or: change) might occur.

Timilsina et al. [23] show that basic genetic algorithms cannot be applied to large instances of timetable solving problems. Their final solution still violates hard constraints. Their conclusion is that improved genetic operators are needed to achieve the desired result. However, Mahar et al. [15] demonstrated earlier that the timetable problem can indeed be solved if custom genetic operators and a well thought-out chromosome composition are used. They demonstrate a working approach using a genetic algorithm to generate a feasible timetable for the College of Computing and Information Technology in Egypt.

Beligiannis et al. [3] used a genetic algorithm to construct timetables for different high schools in Greece. They considered some constraints as hard, not to be broken constraints – e.g. as co-teachers and sub-classes – and at the same time tried to optimize for soft constraints – such as the teachers' course distribution over a week. The running time of the algorithm was at least 30 minutes in approximately 100 test cases. The authors note that there might be ways to improve the algorithm to run faster and more efficient.

2.6.2. Firefly Algorithm

The Firefly algorithm, as proposed by Yang [25] and applied to timetable optimization in [19], is also an evolutionary algorithm. It is based on firefly mating patterns and is similar to genetic algorithms 2.6.1 in the sense that an initial population of solutions is being evolved – with each generation producing more optimal solutions – but differs in the way new solutions are created.

To generate new candidate solutions, the Firefly algorithm chooses for every firefly (candidate) in a generation a random other firefly. If the fitness of randomly selected candidate is better than the initial candidate, the entire solution is replaced. If, however, the initial candidate has a higher fitness, two operations are performed:

1. Two randomly selected slots from the candidate are replaced with new random values.
2. One randomly selected slot from candidate is replaced with the value of the candidate with the higher fitness value.

A similar fitness function as for the genetic algorithm is used to determine the fitness of the candidate solutions and the algorithm terminates when a predefined fitness level is reached.

The firefly algorithm has only been tested on a non-representative test set as a proof of concept. In current literature, no evidence is presented that the algorithm can function on a real-world dataset and converges within a reasonable amount of time.

2.6.3. Bee Colony Optimization Algorithm

Bee Colony Optimization (BCO), as first presented by Karaboga [10] and applied to timetable optimization in [18], is another approach to search the timetable problem space. BCO is also an evolutionary algorithm and is based on the behavior when searching for food of honey bees. As with any evolutionary algorithm, the algorithm starts off with an initial population of, in this case, artificial honey bees. The honey bees gather around food sources (a subset of possible solutions) with some amount of nectar (fitness value). Three types of bees are employed:

- Employed Bees – Worker bees which each take an available solution.
- Onlooker Bees – Helper bees which try to improve the available solutions by selecting the most promising solutions using Roulette Wheel (or: fitness proportionate) selection.
- Scout Bees – To prevent getting stuck in a local optimum, Scout Bees generate new food sources when all solutions in the existing food sources have been evaluated.

The employed bees start at a random food source where their solutions are calculated. Onlooker bees then try to improve their solutions using Roulette Wheel selection, where solutions with a higher fitness have a higher probability of being selected. When a food source is exhausted, the scout bees direct all bees to another food source. The algorithm terminates after a predefined number of food sources have been explored or when the desired fitness value is obtained.

2.6.4. Hybrid Firefly / BCO

Building on their previous work in [19] and [10], Sahoo et al. have presented a hybrid approach using both Bee Colony Optimization and the Firefly algorithm [21]. However, as in the first two papers, no simulation on a representative dataset has been performed. The limited test runs function merely as a proof of concept and it is not clear whether or not their solutions (Firefly, Bee Colony as well as the hybrid approach) are feasible in practice.

2.6.5. Particle Swarm Optimization Algorithm

Particle Swarm Optimization (PSO) was developed by Kennedy and Eberhart in 1995 [4]. Each particle in the PSO is a representation of a candidate solution. By using the personal and global best, the velocity of each particle is tuned to find the optimal solution. The advantages of PSO are that PSO is easy to implement and there are few parameters to adjust. The disadvantage is that the PSO only looks at the particle itself to find the optimal. Using variations of the PSO give feasibele solutions for the timetable problem, but take time [11].

2.6.6. Tabu Search Algorithm

The Tabu Search Algorithm is a local metaheuristic search algorithm that is guided to find a near optimal solution [9]. The algorithm looks to neighbour solutions and remembers the best solution. The rest is added to the Tabu list to remember that these moves are not permitted. This is done in the short time memory, to keep the algorithm from being stuck in a local minimum. During the algorithm, each solution will be checked to the criteria. If the solution satisfies, the solution will accepted. Otherwise there will be a small swap to find a better solution. The disadvantage of this algorithm is that the short time memory is usually not enough to keep the algorithm form being stuck in a local minimum.

2.6.7. Simulated Annealing Algorithm

The Simulated Annealing algorithm is a Monte Carlo algorithm that uses the principles of atoms and temperature [17]. When atoms are heated, they have the characteristic to move freely in space at random. When cooled down, the atoms move less freely until the cooling temperature where they get frozen and do not move. The Simulated Annealing algorithm uses this characteristics to randomly move elements, depending on the simulated temperature. Cooling the elements quickly will lower the chance to find a low cost solution. A solution is accepted if the system temperature (the timetable) is lower than the initial temperature.

2.6.8. Tiling Algorithm

Kingston [12] developed a tiling algorithm where tiles where created that contains a solution for a time slot. When a tile is added in a specific place in a column, a test is run to check the compatibility of this tile. Using the alternating path algorithm a effective resource allocating algorithm is developed.

2.7. Existing timetabling software

Various free and commercial applications exist which allow the user to generate timetable automatically or semi-automatically (iteratively). Most applications feature some kind of graphical user interface to guide the user in the timetable creation process by allowing users to easily define school resources and timetable constraints. This section covers the most commonly used applications and their inner workings.

Some of the current possibilities are ASC Timetables [22] (licensed), Timetabler [14](licensed), FET [13](free open source) and Unitime [24](free open source) . Key2Soft needs to use the solution in their own Key2School system, so therefore only free open source solutions with a permissive license are usable.

2.7.1. aSc Timetables (commercial)

The aSc timetable software [22] is one of the most used commercial software solutions for the timetabling problem. During the 20 years of development, aSc created a solutions which creates a timetable in matter of seconds and dynamically usable in any school or system. The system gives the user the possibility to create any constraint that is needed and changing the timetable manually. The algorithms used is confidential, which makes this system unasable for this project.

2.7.2. Timetabler (commercial)

Timetabler [14] started as an free software solution. For 40 years, the timetabler team experimented to create a fast and friendly usable software solution. As the system progressed, the system became commercial and is now used in 80 countries. The algorithms of timetabler and the optimization they used are also propitiatory information, which makes this system also not usable for this project.

2.7.3. FET (free open source)

FET is an open source free timetabling software in C++ that is developed by Liviu Lalescu and Volker Dirr [13]. The first release of the current algorithm was on November 5, 2009 and is still updated frequently. It started in 2002 with a genetic algorithm, but this was computational slow. For this reason they created another algorithm which they called Recursive Swapping (see section 2.8.1). FET supports many different languages, such as Arabic, English and Greek. It also support many different constraints. The hardness of a constraint can be set between 0% and 100%, where 100% means a hard constraint. Bauteu et Al. [2] did a case study using FET and the results showed that the execution time is low (max 30 seconds in the demonstrated cases) compared to other algorithms. The main factor on the execution time are the (combination of) constraints used. Further research into the FET program can be found in section 2.8.

2.7.4. Unitime (free)

Unitime is a web-based automated timetabling application with a constraints solver written in Java [24]. It is based on the iterative forward search algorithm. Unitime was developed by staff from several faculties from North America and Europe. The development started in the year 2001. It uses a MySQL database to store data and works as a server-client package. The constraints are based on penalties, where -4 means strongly preferred and +4 is strongly discouraged. A penalty of ∞ represent a constraint that should always be true. The run time of Unitime differs: simple timetables take around 10 seconds, where larger and more difficult timetables take up to 15 minutes [16].

2.7.5. Conclusion

As can be seen by the wide variety of algorithms presented in this section, solving the timetable problem can be done in many different ways. However, no definitive method which always produces a solution in any given case exists (hence the NP-hardness), but there is also no method which produces a solution when it is known that the constraints, in theory, can be satisfied within a given time frame. Meta-heuristic algorithms cannot give any kind of guarantee regarding their output.

In literature, cutting-edge solutions to the timetable problem are usually only applied to a specific data set which is very suitable for that specific algorithm. Most of the time, these data sets do not truly represent real-world timetabling scenarios, which makes it hard to determine their real performance. This is another factor which makes it hard to choose a suitable approach.

We suspect that hybrid solutions are used in practice to provide specific benefits in specific scenarios. Some algorithms have disadvantages that can be overcome in a specific situation by combining it with another algorithm. Our limited experience with the timetable problem makes it difficult to select the right combination of algorithms beforehand.

For the final consideration, we have taken into account all the module requirements, the timetable constraints, the researched algorithms with their (dis)advantages, the amount of available literature and real-world applications and examples. The most promising approach seems to be based on a genetic algorithm. The decisive factor in this case was the paper of Beligiannis et al. [3] about timetabling in Greek high schools, where a genetic algorithm was successfully applied in practice in a situation similar to ours.

However, in consultation with Key2Soft, the genetic algorithm was deemed too slow to use in practice. As shown in [3], the lower bound on the run time was approximately 30 minutes. For this reason, we have decided to use an existing open source software solution which has empirically proven to be fast in most real world timetabling cases and integrate that into the system. Looking at the possibilities, two software solutions are possible candidates: Unitime and FET. Overall, FET seems to be a better fit, because of the faster run time and the provision of a standalone executable to simplify integration into a larger project. For this reason, we continued researching the inner workings of FET more in-depth.

2.8. The FET Program

Upon recommendation of the Software Lead at Key2Soft, we have devoted extra effort to researching the possibilities and inner workings of the FET timetabling program. FET was advertised to be very fast, which was a strong point of focus for Key2Soft. Because the algorithms described earlier in section 2.6 are either complex, slow or difficult to implement, being able to use an existing timetabling program would benefit the project.

The inner workings of FET are poorly documented and the program has a cluttered and unclear code base, consisting of source code files of up to 16.000 lines of code. However, a list of all supported constraints is available on their website [13] and a high level description and explanation of the algorithm is provided with the documentation of each release. The author is furthermore communicating extensively with the active timetabling community at a forum on their website, where important practical information about the workings of `FET-CL` can be found.

This section provides an overview of the algorithm in use and the possibility to use a headless version of FET as a standalone program. More information about the constraints can be found in appendix B.

2.8.1. Recursive Swapping Algorithm

FET uses a semi-greedy algorithm based on the process of how a human timetabler would construct a timetable. The algorithm is referred to as Recursive Swapping and provides in most cases a fast and efficient way to solve the timetabling problem. The pseudo-code of the algorithm, which is derived from the code, is listed in algorithm 1.

The algorithm starts by sorting the activities, placing the most difficult (or: most constrained) tasks first. The algorithm's speed greatly benefits from this approach, because less constrained activities are easier to schedule with fewer available time slots.

The main approach of the algorithm is to randomly assign all activities to a time slot. If this is not possible for some activity, e.g. there are no time slots available which satisfy the constrains on that activity, it will place the activity in a valid time slot and re-adds the conflicting activities recursively.

Algorithm 1 Recursive swapping

```
Input: A list $L$ of activities $A_1..A_n$ and the constraints
Sort activities                                   ▷ Most difficult first
for all $A_i \in L$ do
    bool ← AddActivityInTimeSlot($A_i$)           ▷ Chooses a random available slot, else returns false
    if !bool then                                 ▷ Recursive swapping
        List x                                    ▷ Conflicting activities list for each timeslot
        for all $T_j \in TimeSlots$ do
            x.addConflictingActivities($T_j$)
        end for
        $T_l$ ← lowestConflictingActivities($x$)
        List $A$ ← $T_l$.removeAll                ▷ List A contains all the activities in $T_l$ and empties the timeslot
        for all $A_x \in A$ do
            $s$ ← addRecursively($A_x$)
        end for
        if s then
            Return true
        else
            $T_l$ ← nextLowestConflictingActivities($x$)
            addRecursivelyInTimeSlot($T_l$)       ▷ Use different timeslot and redo from line 10
        end if
    end if
end for
Return false
```

2.8.2. FET-CL

A command-line only version of FET (`FET-CL`) is included with every release of the program. This version of FET functions roughly the same as the version with GUI, but has a couple of notable differences:

- The command line version does not halt on warning messages, but instead dismisses them and continues.
- The command line version does not write the amount of broken soft constraints to the console, but only to a result file (as does the GUI version). The result file, however, is more detailed than the information provided by the GUI.
- The command line version is unable to provide information about the activity that is currently being scheduled.

`FET-CL` accepts as much as 40 command line arguments, most of which alter the way the output of the final timetable is constructed. The most important arguments are described below. For a full overview, please refer to to Appendix B.

inputfile (*required*)
The XML input file, for instance "data.fet".

outputdir (*optional*)
default: current working path

The path to the results directory. In this directory, the timetables and log files are written. The timetables are available in multiple formats (including HTML and XML) and from multiple points of view (such as teachers, student groups, subgroups and rooms).

timelimitseconds (*optional*)
default: 2000000000

The time limit on the timetable generation.

language (*optional*)
default: en_US

The language of the program, especially relevant for the status and error messages produced by the program.

verbose (*optional*)
default: false

When true, makes the program output more status information and updates about the algorithm.

2.8.3. FET file structure

FET can load and store timetabling resources in an XML file, which has to use a `.fet` file extension. The internal timetabling resources are saved as individual XML elements. Each resource has its own child elements and multiple resources of the same kind are wrapped by a grouping element. Since there was limited to no documentation available about the XML structure, we have documented the `.fet` file by creating a list of all available elements and their use. Please refer to section B.3 for the documentation.

3 Design

3.1. Introduction

This chapter is devoted to the overall system architecture and design choices, which have been thought out before the actual implementation of the system commenced. Section 3.2 describes therefore the architecture of all system components.

3.2. Architecture

In this section, the architecture of the system is described. In section 3.2.1, an overview of the system is presented. Section 3.2.2 explains the data model, section 3.2.3 is devoted to the timetable resource objects, section 3.2.4 is about the XML input file creator and section 3.2.5 describes the design of the timetabling algorithm. Finally, in section 3.2.6, the design of the timetable result object is explained.

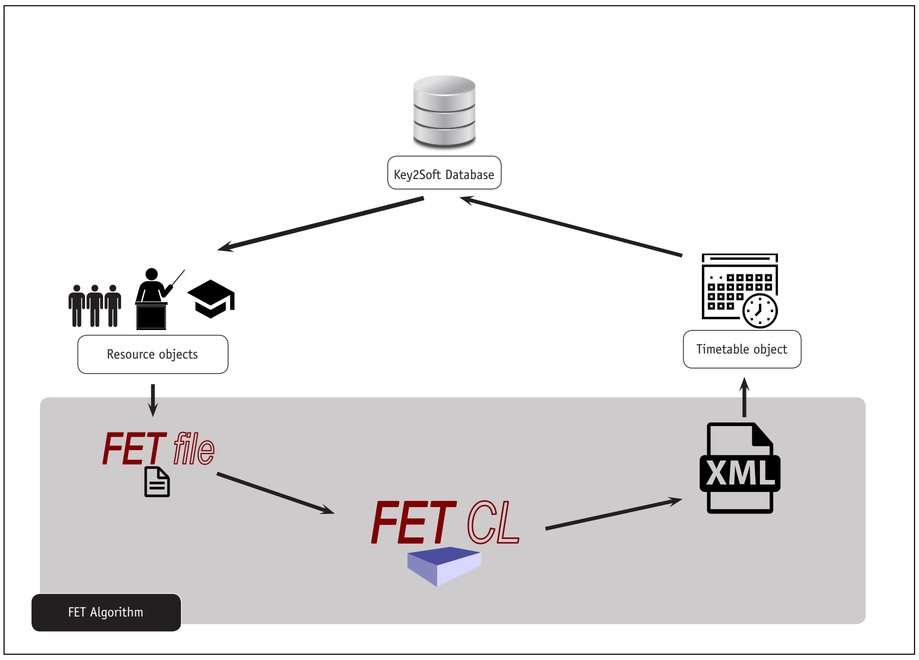


Figure 3.1: System architecture overview

3.2.1. Overview

The system for creating a timetable contains revolves mainly around the database, timetable resource objects, FET-CL and an XML creator (see figure 3.1).

First of all, there is an existing database that Key2Soft uses for their systems. This database contains all the Key2School data from a school. The data that is needed for creating a timetable is extracted from that database and converted into resource objects, such as teachers, students and subjects. These objects need to be converted to a specific file structure, a FET file. The FET file is inserted into the FET-CL system which generates an XML output file. This file contains the scheduled activities and their respective time slots. The XML file is read into a timetable object by the system and will finally be stored in the database as final output.

Figure 3.2 shows an Entity-Relationship (ER) diagram of the overall system design. In the following sections, each part will be defined and explained for a more in-depth view.

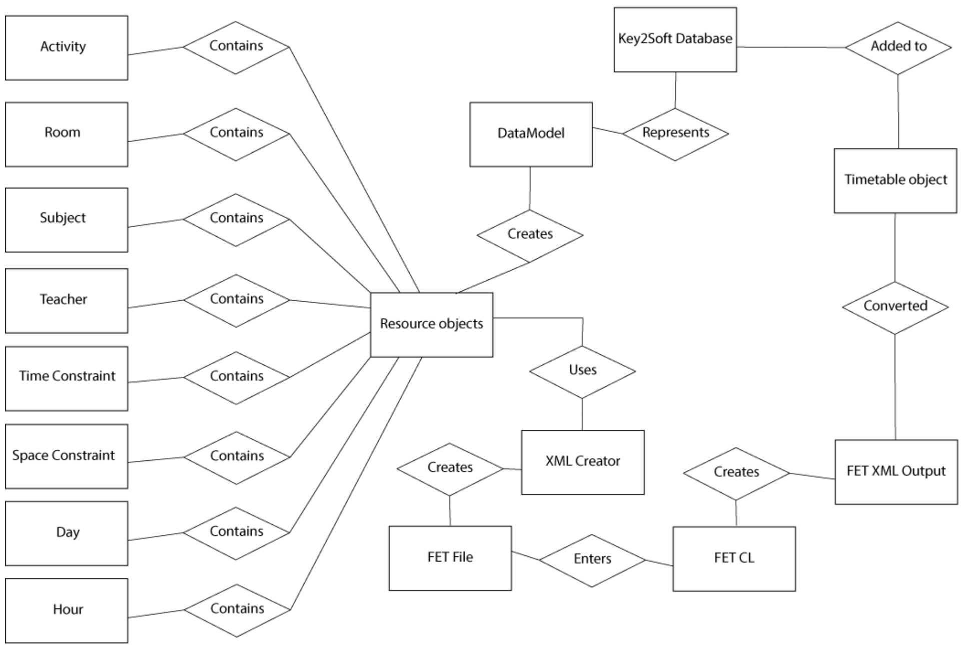
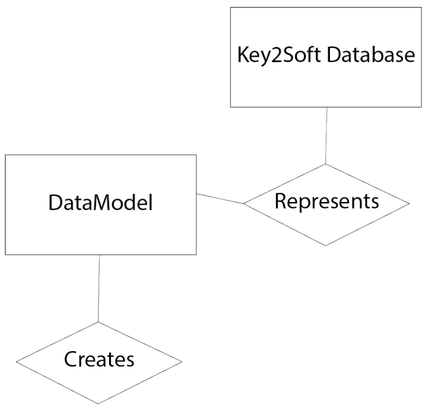


Figure 3.2: ER Diagram

3.2.2. Data Model



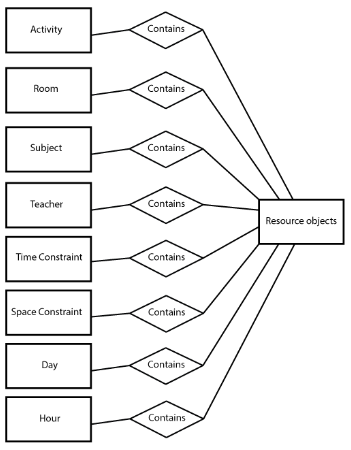
The data model is a representation of the Key2Soft Database that enables the database to be used as an ORM. An ORM (Object-Relational Mapper) maps a database to a model so that developers can use the database as objects, removing the need for additional (custom) code to access the database. It creates a simple and programmer-friendly code environment. The database becomes a virtual object model that can be used in an object-oriented system.

There are a couple of different ORMs available. The most well-known and widely used ORM is the .NET Entity Framework [8]. Another ORM that has been considered is the Dapper Framework [7], a newer and faster ORM than the .NET Entity Framework. The downside of Dapper is that it has less features and more code is needed to do the same job as the .NET Entity Framework. It is also less stable than the .NET Entity Framework.

Considering these differences, and taking into account that Key2Soft

was already using this framework, the .NET Entity Framework was
chosen. The simplicity and stability of this framework are more important for this project. Dapper is faster, but the speed-up is negligible relative to the total amount of run time of the final product (primarily caused by the timetable generation component). Furthermore, the different features of the .NET Entity Framework provide Key2Soft developers with more freedom when using the system in the future, as maintainability is one of the main design goals.

3.2.3. Resource objects

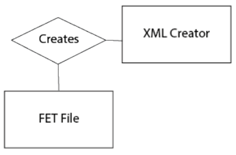


The resource objects are different lists objects containing the needed data from the DataModel to create a timetable. The different objects are:

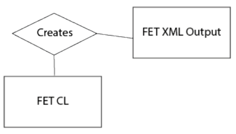
- **Activity:** Consisting all the activities with a teacher, students set, room and a subject.
- **Room:** All the different rooms and the type of the room
- **Subject:** A list of the names of the subjects
- **Teacher:** A list of the names of the teachers
- **Time Constraint:** Different time constraints objects
- **Space Constraint:** Different space constraints objects
- **Day:** The days of the week
- **Hour:** The number of hours in a week

3.2.4. XML creator

The resource objects need to be converted into a FET file. This is done by the XML creator. It uses the objects to create a XML file with the needed structure to be a FET file. Different XML parsers and creators exist, but because FET needs a specific XML structure, the simple XML Creator is made. It uses the LINQ XElements and XDocuments objects to create the file. The reason for using LINQ is that it is simple and structured in such a way that it is easy readable and fast.



3.2.5. FET-CL

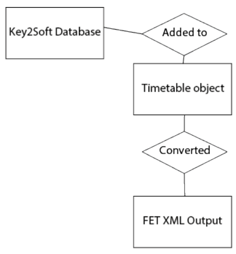


The `FET-CL` program is used as a black box module. This means that the system provides input to `FET-CL` and processes its output, without the possibility to interact with the `FET-CL` program itself. The system can only control the input file. The advantage of using such a black box module is that the developers do not need to be concerned about the `FET-CL` itself: the algorithm is contained in that program and does not need to be interfered with. The disadvantage of this design is that developers cannot see what happens during the generation and have to rely on the output of the `FET-CL` process.

3.2.6. Timetable object

The final part is returning the output into the database. The output of the FET-CL is an XML file with activities in the following structure:

- **ID:** The ID of the corresponding activity
- **Day:** Which day the activity is placed
- **Hour:** Which hour the activity is placed



- **Room:** In which room the activity is placed, if a predefined room type was provided.

This output is converted into a timetable object that adds each activity in the database with the corresponding attributes.

4 Implementation

4.1. Introduction

The implementation chapter describes how each part is implemented and used to create the timetable generator system. In section 4.2 the structure of the solution is explained. Section 4.3 give a detailed description of each resource object that is created. The way FET is used is explained in section 4.6.

4.2. Project structure

This project is part of the larger Key2School software package, as described earlier. In this section, the project structure and implementation details are discussed.

A C# software solution can contain multiple separate C# projects. These projects can have different build or compile settings and will each compile to its own assembly. This project consists of three separate projects: one for the core timetabling part, one for unit tests and one containing a sample implementation.

The core timetabling part – generating timetables from a given set of resources and constraints – can function on its own. This part of the project has therefore been structured as a C# class library. This yields a `.dll` file which can be referenced by the Key2School project and exposes only the necessary methods for timetable generation.

The unit tests are included in a separate project, which is a best practice for C# projects according to Microsoft[1]. Because the tests reside in a different project, the test code is not shipped in production which reduces the total size of the project and prevents accidental use of unit test methods by code that references this assembly. A detailed explanation of the testing methods and challenges can be found in section 5.

Furthermore, a third project with an example implementation has been included in the solution. This project demonstrates the usage of the timetabling library – explaining which methods to use and how to process the resulting timetable – and serves as a practical addition to the documentation.

4.3. Resource objects

The resource objects are stored in different lists containing the data from the database. Each list inherits the `AbstractList` class, which is a abstract class that defines the correct methods and attributes that are needed.

Each object and list has a XElement representation (see 4.4). The XElement representation is a specific representation for constructing a FET file.

[1]https://docs.microsoft.com/en-us/visualstudio/test/walkthrough-creating-and-running-unit-tests-for-managed-code

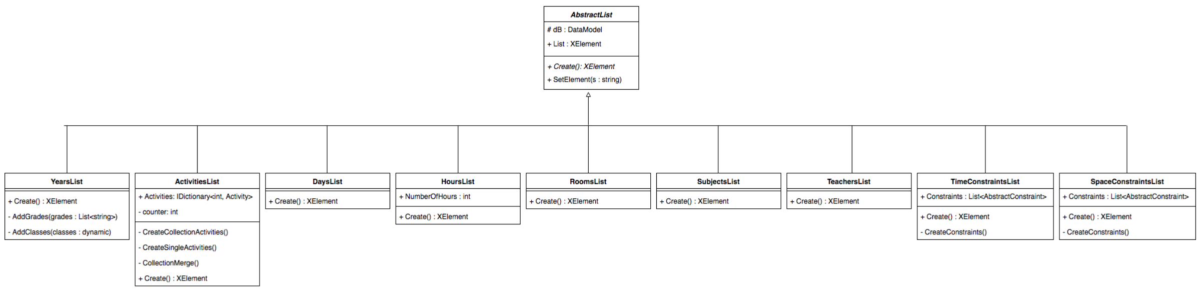


Figure 4.1: UML overview - Lists

4.3.1. AbstractList

The abstract list defines the methods for the lists of the system. Each list needs to implement the `AbstractList` class so that all the needed methods are implemented.

The constructor defines the data model that is needed to retrieve the data. The `SetElement` method makes sure that the XElement has the correct name, which is specific to the needed FET name.

The abstract `Create` method makes sure that this method is implemented by any sub classes. Using the same method ensures that the main program only needs to call one method for all the different classes, which makes adding or removing list simple and maintainable. The `Create` method of each of the list retrieves the specific data that is needed for that list and creates the needed objects. For example, the `ActivitiesList` retrieves the data from different tables and constructs `Activity` objects.

4.3.2. ActivitiesList

`ActivitiesList` creates the `Activity` objects from the datamodel. Each activity consist of a group of students, a list of teachers, the subject and the room. FET has some rules that needs to be followed for the activities that are different for the requirements of the Egyptian schools:

1. There can only be one subject in the same activity.
2. Teachers can only teach one activity in a time slot.
3. Classes can only have one activity in a time slot.

Egyptian schools have *collections*, see constraint code L2 in table 2.5.1, which are subjects that need to be scheduled in the same time slot, with different teachers and different classes. Students can choose which course they would like to follow, which is after the generated timetable. But because of the rules of FET, it is not possible to make all the activities and try to schedule them in the same time slot. For this reason, each collection is its own 'subject' in the back-end. Using this new subject, for example coll 1, all the students and teachers that need to be in the same time slot are merged into one activity. This is done by the `CollectionMerge` method in the `ActivitiesList` class.

4.3.3. ActivityBuilder

To create the `Activity` objects, the ActivitiesList creates a ActivityBuilder which construct the different activities from the inserted settings. The ActivityBuilder defines the number of activities that should be created, the duration of each of these activities, and how to group the activities. This makes sure the activities that should be together, for example the same subject for one class, have the same group ID. This is important for the constraints to know for example which activities should not be on the same day. It is also important to know the order of the activities, which is done by the `NumberLessonOfWeek` attribute in the activity.

4.3.4. Constraints

Constraints are grouped into two groups; time and space. Time constraints are constraints that have influence on time specific rules, for example the preferred time a teacher wants to teach. The space constraints

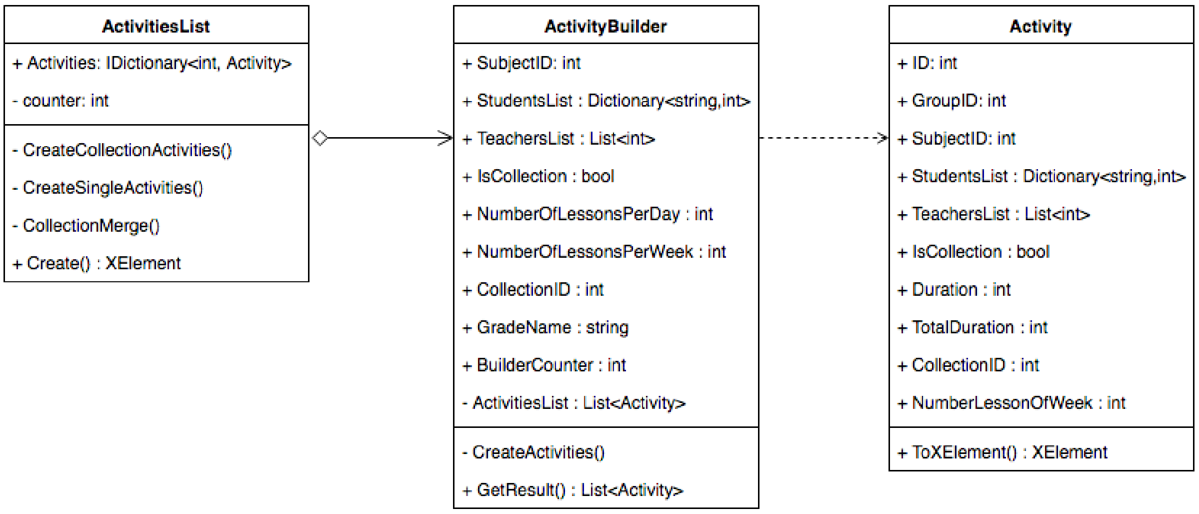

ActivitiesList
+ Activities: IDictionary<int, Activity>
- counter: int
- CreateCollectionActivities()
- CreateSingleActivities()
- CollectionMerge()
+ Create() : XElement
ActivityBuilder
+ SubjectID: int
+ StudentsList : Dictionary<string,int>
+ TeachersList : List<int>
+ IsCollection : bool
+ NumberOfLessonsPerDay : int
+ NumberOfLessonsPerWeek : int
+ CollectionID : int
+ GradeName : string
+ BuilderCounter : int
- ActivitiesList : List<Activity>
- CreateActivities()
+ GetResult() : List<Activity>
Activity
+ ID: int
+ GroupID: int
+ SubjectID: int
+ StudentsList : Dictionary<string,int>
+ TeachersList : List<int>
+ IsCollection : bool
+ Duration : int
+ TotalDuration : int
+ CollectionID : int
+ NumberLessonOfWeek : int
+ ToXElement() : XElement

Figure 4.2: UML overview - ActivityBuilder

are more specific for resource utilization, such as preferred room. The constraints that are used are based on the requirements and the data of Key2Soft.

`SpaceConstraintsList` and `TimeConstraintsList` each have a constraints list, which contains all the constraints objects.

Each constraint is a subclass of the `AbstractConstraint` class. As `AbstractList`, the `AbstractConstraint` class makes sure that all the constraints have the same fundamental base to start with. It makes sure that each constraint needs to have a XElement representation and that it creates the constraint object from the data model.

Adding or removing constraints is a part that developers will need to us if things change. For this reason, this is made simple. Implementing `AbstractConstraint` ensures all the needed methods are implemented. Furthermore, by just adding or removing the constraint object in the specific list, the program will create all the needed objects. Each constraint will only need to know where to retrieve the data from and how its XElement representation looks like. The specific XElement representation for a specific constraint in FET can be found at Appendix B.

Most constraints work as explained in Appendix B, but some needed to be different or used differently to be able to work for the schools in Egypt.

ConstraintPeriodSection

This constraints does not exist in the FET system. The constraints creates a `ConstraintStudentSetNotAvailableTimes` for the weekend of different students. Schools in Egypt contain sections, such as Britain or Arabic. This sections imply the type of schooling, and have different days as weekend. For example, the Britian section has the Saturday's and Sunday's as weekend, while the Arabic section has Friday and Saturday as weekend.

ConstraintActivitiesPreferredStartingTimes

Some subjects would prefer for example not to be scheduled in the morning. This needs to be handled. For this reason, the `ConstraintActivitiesPreferredStartingTimes` is used to remove the time slots which are not preferable. This is a kind of time off for the subjects, but in reverse. Instead of saying which time slots should not be used for a subject, it states the time slots which should be used. A not available time constraint for subjects is not possible in FET, so for that reason, this constraint is used.

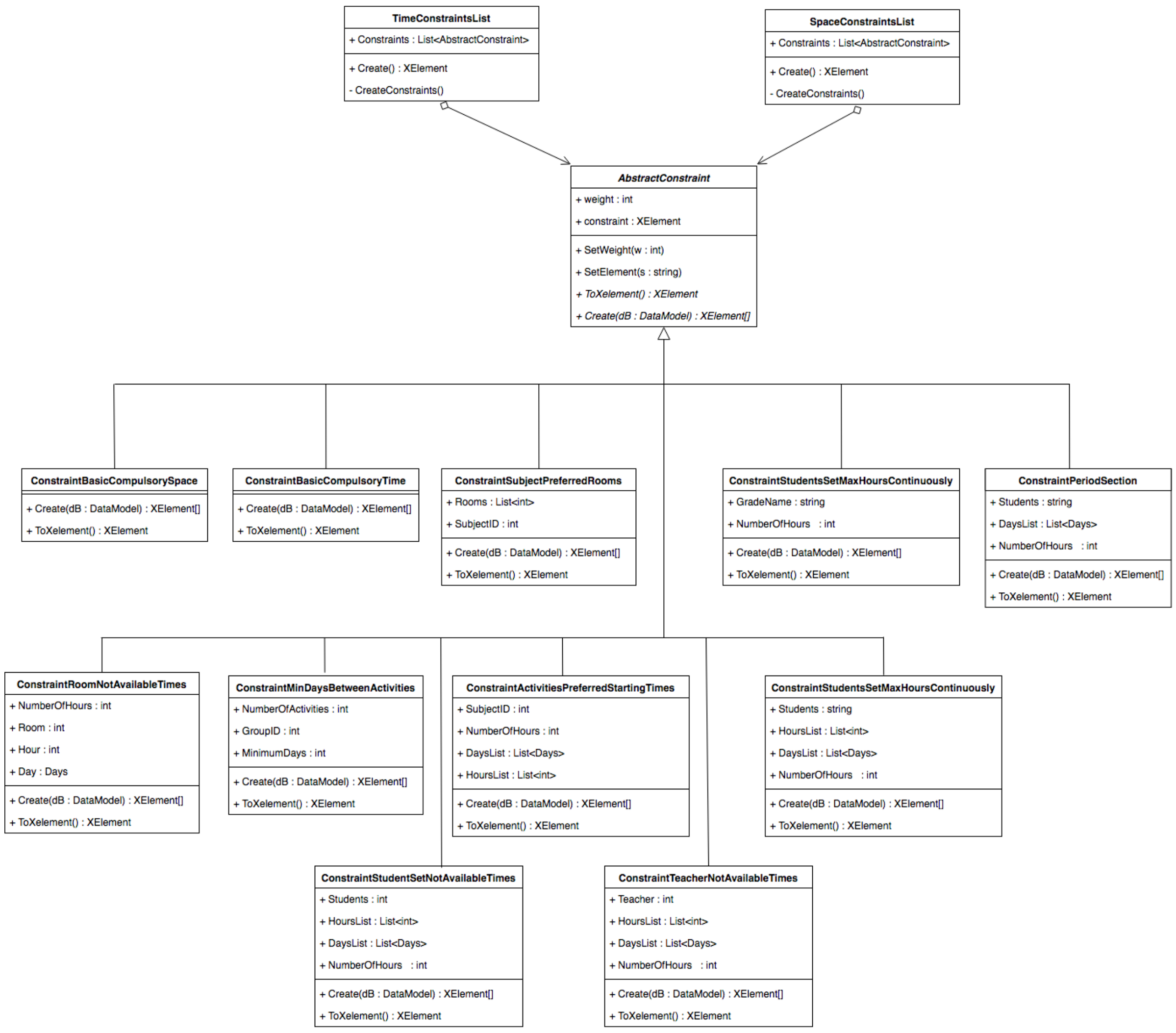

TimeConstraintsList
+ Constraints : List<AbstractConstraint>
+ Create() : XElement
- CreateConstraints()
SpaceConstraintsList
+ Constraints : List<AbstractConstraint>
+ Create() : XElement
- CreateConstraints()
AbstractConstraint
+ weight : int
+ constraint : XElement
+ SetWeight(w : int)
+ SetElement(s : string)
+ ToXelement() : XElement
+ Create(dB : DataModel) : XElement[]
ConstraintBasicCompulsorySpace
+ Create(dB : DataModel) : XElement[]
+ ToXelement() : XElement
ConstraintBasicCompulsoryTime
+ Create(dB : DataModel) : XElement[]
+ ToXelement() : XElement
ConstraintSubjectPreferredRooms
+ Rooms : List<int>
+ SubjectID : int
+ Create(dB : DataModel) : XElement[]
+ ToXelement() : XElement
ConstraintStudentsSetMaxHoursContinuously
+ GradeName : string
+ NumberOfHours : int
+ Create(dB : DataModel) : XElement[]
+ ToXelement() : XElement
ConstraintPeriodSection
+ Students : string
+ DaysList : List<Days>
+ NumberOfHours : int
+ Create(dB : DataModel) : XElement[]
+ ToXelement() : XElement
ConstraintRoomNotAvailableTimes
+ NumberOfHours : int
+ Room : int
+ Hour : int
+ Day : Days
+ Create(dB : DataModel) : XElement[]
+ ToXelement() : XElement
ConstraintMinDaysBetweenActivities
+ NumberOfActivities : int
+ GroupID : int
+ MinimumDays : int
+ Create(dB : DataModel) : XElement[]
+ ToXelement() : XElement
ConstraintActivitiesPreferredStartingTimes
+ SubjectID : int
+ NumberOfHours : int
+ DaysList : List<Days>
+ HoursList : List<int>
+ Create(dB : DataModel) : XElement[]
+ ToXelement() : XElement
ConstraintStudentsSetMaxHoursContinuously
+ Students : string
+ HoursList : List<int>
+ DaysList : List<Days>
+ NumberOfHours : int
+ Create(dB : DataModel) : XElement[]
+ ToXelement() : XElement
ConstraintStudentSetNotAvailableTimes
+ Students : int
+ HoursList : List<int>
+ DaysList : List<Days>
+ NumberOfHours : int
+ Create(dB : DataModel) : XElement[]
+ ToXelement() : XElement
ConstraintTeacherNotAvailableTimes
+ Teacher : int
+ HoursList : List<int>
+ DaysList : List<Days>
+ NumberOfHours : int
+ Create(dB : DataModel) : XElement[]
+ ToXelement() : XElement

Figure 4.3: UML overview - Constraints

4.4. XML parser

Using FET requires a specific XML file to be used as input for the system. Therefore, each object needs to have its specific XML object representation.

The XML file is constructed by using the LINQ XElements and XDocument objects. This is done because of the simplicity of the objects and the use of LINQ in the datamodel search.

The `XMLCreator` class is the main class that creates a document. Each resource object has its own XElement representation which is called by using the `ToXElement()` method. Each elements is added to the XDocument of the `XMLCreator`.

FET has different XML representations for the same attribute. For example, a subject can be `Subject` or `Subject_Name`. For that reason, each resource object needs to have its own specific representation.

An example of a XElement representation is shown in example listing 1. A more extensive example can be found in Appendix B.4.

4.5. Data model

The database is modelled by using the .NET Entity Framework. The `DataModel` class creates the main connection and builds the model from the database. This is done by extending the `DbContext`. Because of the

```xml
<Activity>
    <Id>3</Id>
    <Activity_Group_Id>3</Activity_Group_Id>
    <Duration>2</Duration>
    <Total_Duration>6</Total_Duration>
    <Students>Prim1A</Students>
    <Teacher>Teacher A</Teacher>
    <Subject>Subject A</Subject>
</Activity>
```

Listing 1: Activity FET XML representation example

.NET Entity Framework, no additional code is needed to make a connection but it suffices to define a connection string in the configuration file `connection.config` (see listing 2). The connection string requires the location of the database, the database (or: catalog) name and credentials and can optionally be configured with for example connection and query timeouts.

```
connectionString = "data source=[HOSTNAME]; initial catalog=[CATALOG NAME];
persist security info=True; MultipleActiveResultSets=True; App=EntityFramework;
User ID=[USER_ID]; Password=[USER_PASSWORD];
Connect Timeout=1;ConnectRetryInterval=1; ConnectRetryCount=1" />
```

Listing 2: Connection string

4.6. Timetable generation

The actual timetable generation is handled by the free and open source FET program (see section 2.8). The FET program does not feature an API nor can it be directly used from external code. Therefore, we have decided to manually create a process and configure it with the necessary command-line arguments. When FET completes, the output files are processed and stored in the database.

First, the methods used to create the FET input data are described in section 4.6.2. The FET process implementation is described in section 4.6.3, the output timetable processing in section 4.6.4 and finally the performance of the program in section 4.6.5.

4.6.1. Software design

Since the timetable generation library is mainly used by Key2Soft developers, creating intuitive and easy-to-use methods has been a strong point of focus (per the design goals described in section 2.4). The main class of the library, `TimetableGenerator`, has therefore only two important methods: `RunAlgorithm` and `StopAlgorithm`. The timetable generation class only needs a `TimetablingStrategy` (or: algorithm) and a compatible `DataModel` to run. The code needed to generate a timetable (in a blocking manner) is therefore as concise as the snippet in listing 3.

All algorithm classes have to be derived from the abstract `TimetablingStrategy` class. This project only includes one such implementation: the `FetAlgorithm` class. Key2Soft developers only need to create a different `TimetablingStrategy` and run the `TimetableGenerator` with this new class, might they need to use a different algorithm in the future.

4.6.2. Program input

In the final Key2School product, end-users are able to input all necessary timetable resources to generate a timetable (as described in section 4.3) via a graphical user interface. This interface has been developed by

```
using (var model     = new DataModel())
using (var generator = new TimetableGenerator())
{
    var task = generator.RunAlgorithm(new FetAlgorithm(), model);
    task.Wait();
    var timetable = task.Result;
}
```

Listing 3: Example implementation of the timetabling library

Key2Soft alongside this timetabling project. The data is stored in the Key2Soft database, which is used directly in this project.

The class responsible for generating the input file is called `FetInputGenerator`. This class creates a new instance of each resource list as described in section 4.3 and builds the XML input file in the necessary order. This file is then written to disk. The path to this input file is used later on in the process as input to `FET-CL`.

4.6.3. Process creation and management

Since the FET program will run in an external process, measures have to be taken to handle unexpected termination of the process due to e.g. user intervention or crashes. The timetable generation process has also to support manual termination, in cases where for example the run time is too long. Our implementation is designed to allow and handle these situations accordingly.

`FET-CL` can be configured by a large amount of command-line parameters, which are detailed in appendix B.2. An invalid value for a parameter causes the program to fail, but not always in the same manner or with the same error code. Instead of relying on FET to validate command-line parameters, this logic is moved to our implementation.

The `FetProcessBuilder` class safely creates a `FET-CL` process by checking the various command-line arguments before creating the process object. This class only requires the location to the `FET-CL` executable and can additionally be configured by calling a number of configuration methods. Some command-line parameters need to have a certain value for the program to function correctly, so only the parameters which are safe to be changed can be configured. After a `FetProcessBuilder` has been instantiated and configured, the resulting `Process` object can be retrieved with the `CreateProcess` method.

The resulting object is an instance of the native C# `Process` class. This class is versatile and features a lot of methods to interact with a process. This project, however, only requires the process to be started, stopped and to process its output when the process ends. The `FetProcessFacade` class is created to only expose the necessary methods. The facade class also provides otherwise complex process start- and stop functionality by the means of `StartProcess` and `StopProcess` methods.

Processes in C# run asynchronously by default. This means that starting an external process does not block the main thread, unless it is explicitly awaited. The `Process` class fires an `OnExited` event when the process has finished, which can be used to process its results. To make it easier for the developers at Key2Soft to use this project, the well-established Task-based Asynchronous Pattern (TAP)[2] has been implemented to manage the generation of a timetable. The `TimetablingStrategy` class forces all derived algorithms to implement this pattern. In the case of the included `FetAlgorithm` class, the `Task<Timetable>` is created when the algorithm is run. The task result is set when the process exits and the output has been processed.

The `Task<Timetable>` can succeed, fail or be canceled and can asynchronously be awaited by using one or more `ContinueWith` method calls, as illustrated by the code in listing 4. The final Key2School implementation can use these methods to further use or process the resulting timetable.

[2] https://docs.microsoft.com/en-us/dotnet/standard/asynchronous-programming-patterns/task-based-asynchronous-pattern-tap

```
// task is of type Task<Timetable>.
task.ContinueWith(OnSuccess,  TaskContinuationOptions.OnlyOnRanToCompletion);
task.ContinueWith(OnCanceled, TaskContinuationOptions.OnlyOnCanceled);
task.ContinueWith(OnError,    TaskContinuationOptions.OnlyOnFaulted);
```

Listing 4: Awaiting the timetable task

4.6.4. Output processing

FET-CL stores the results of its work as multiple files in an output folder. The FetAlgorithm implementation included with this project reads the relevant files in this output folder and ultimately constructs a Timetable object. The files needed to construct this timetable are:

- [input name]-activities.xml (see listing 5)
- [input name]-soft_conflicts.txt (see listing 6)

```
<?xml version="1.0" encoding="UTF-8"?>
<Activities_Timetable>
<Activity>
        <Id>120</Id>
        <Day>Monday</Day>
        <Hour>3</Hour>
        <Room></Room>
</Activity>
</Activities_Timetable>
```

Listing 5: Example FET output – activities.xml

```
Soft conflicts of tt_resources_1529594849.fet
Generated with FET 5.36.0 on 6/21/18 5:27 PM

Number of broken soft constraints: 45
Total soft conflicts: 27

Soft conflicts list (in decreasing order):

Time constraint min days between activities broken: activity with id=122 (T:44, S:28, St:Prim1A) conflicts
 ↪  with activity with id=117 (T:44, S:28, St:Prim1A), being 1 days too close, on days Sunday and Sunday,
 ↪  conflicts factor increase=0.6.
Time constraint min days between activities broken: activity with id=60 (T:79, S:3, St:Prim2E) conflicts
 ↪  with activity with id=58 (T:79, S:3, St:Prim2E), being 1 days too close, on days Wednesday and
 ↪  Wednesday, conflicts factor increase=0.6.

End of file.
```

Listing 6: Example FET output – soft-conflicts.txt

First, the XML output file is processed using the built-in C# XML deserializer. This yields a Timetable object with a list of TimetableActivity objects. The scheduled activities file contains activity reference IDs, so each activity is then matched with the original list of input activities to gain access to the resources (teacher, subject, etc.) which are scheduled. Finally, the soft conflicts file with information about the violated constraints is processed and added to the timetable as meta data.

This timetable object is the ultimate result of the Task<Timetable> as mentioned earlier and can directly be used in the Key2School code. The company, however, has also requested the timetable to be saved to the database. A dedicated DatabaseHelper class has been created to save a Timetable object to the database.

This class can be used in the `OnSuccess` handler of the timetabling task by the company. The `DatabaseHelper` class operates on a provided `DataModel` and interacts with four different tables to store the timetable, its activities, the teachers per activity and the classes (or: grades) per activity.

4.6.5. Performance

FET includes a number of example (anonymized) timetabling resource files, provided by users which have used FET at their institutions. These examples are used to demonstrate the capabilities of FET and its large variety of constraints. Some examples run in less than one second, others take as much as 30 minutes.

In order verify that our solution satisfies the run time requirement of at most 30 minutes (and preferably less than 5 minutes, see section 2.5.1), Key2Soft has provided us with real-world timetabling data from an Egyptian high school.

The total run time of FET on the provided data was one second. Taking into account the time needed to retrieve the information from the database, generating the input file, processing the output and storing the output again, the total run time of the program is less than 15 seconds. This greatly exceeds the initial expectations of FET and validates the choice for this algorithm.

4.7. Configuration

C# applications are usually configured by an XML `app.config` configuration file. To configure timetabling-specific settings, a dedicated section has been added to `app.config` named `timetablingSettings`.

```xml
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <configSections>
  (...)
    <section name="timetablingSettings" type="System.Configuration.NameValueSectionHandler" />
  </configSections>

  (...)

  <timetablingSettings>

    <!-- Maximum duration of the timetable generation in seconds. Default: 0 (unlimited) -->
    <add key="Timeout" value="0" />

    <!-- Relative path to fet-cl executable. Default: "lib/fet/fet-cl" -->
    <add key="FetExecutableLocation" value="lib/fet/fet-cl" />

    <!-- Language of FET error message output. Must be of type FetLanguage. Default:
     ↪  FetLanguage.US_English. -->
    <add key="FetLanguage" value="FetLanguage.Dutch" />

    <!-- Working directory of the program, where intermediary files are stored. Default:
     ↪  "%TEMP%/timetabling" -->
    <add key="FetWorkingDir" value="%TEMP%/timetabling" />

  </timetablingSettings>
</configuration>
```

Because the FET program requires very specific input and misconfiguration can be very difficult to debug, a dedicated class `FetConfig` to process and validate settings has been created. To enforce a valid language setting, only allowing one of the 34 predefined languages, an auxilliary `enum`-like class `FetLanguage` is used.

The `FetConfig` class contains dedicated methods for each setting, which cast the setting string to the required format and fall back to a default value when an error occurs. Additionally, the `GetSetting` method retrieves the raw value of a setting.

The `FetLanguage` has been implemented using an enum-mimicking static class structure. This enables developers to use the static fields of the class as an instantiation of that class. The upside of this approach is

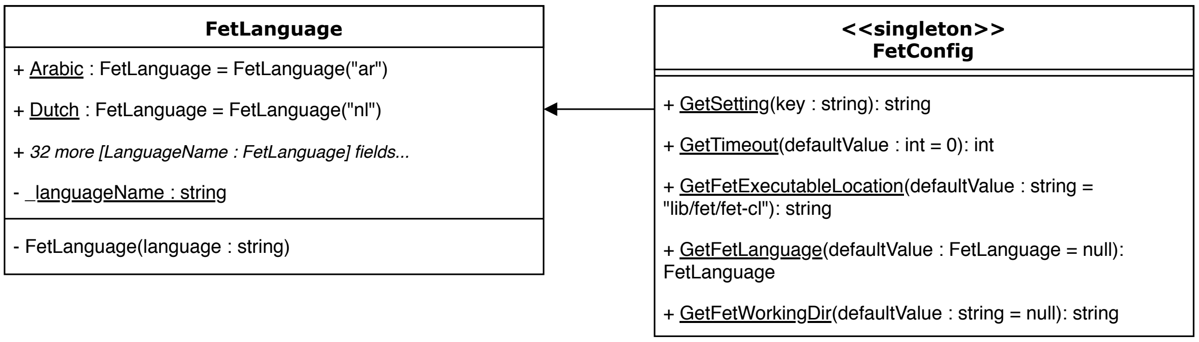

FetLanguage
+ Arabic : FetLanguage = FetLanguage("ar")
+ Dutch : FetLanguage = FetLanguage("nl")
+ 32 more [LanguageName : FetLanguage] fields...
- _languageName : string
- FetLanguage(language : string)
<<singleton>>
FetConfig
+ GetSetting(key : string): string
+ GetTimeout(defaultValue : int = 0): int
+ GetFetExecutableLocation(defaultValue : string = "lib/fet/fet-cl"): string
+ GetFetLanguage(defaultValue : FetLanguage = null): FetLanguage
+ GetFetWorkingDir(defaultValue : string = null): string

Figure 4.4: FET configuration classes

that the developer can use this object throughout the program and does not have to have knowledge of the language codes, but can instead use the human-readable language name.

```
public sealed class FetLanguage
{
        public static readonly FetLanguage Arabic = new FetLanguage("ar");
        public static readonly FetLanguage Dutch = new FetLanguage("nl");
        (..)

        private readonly string _languageName;
        private FetLanguage(string languageName) => _languageName = languageName;

        /// <inheritdoc />
        public override string ToString() => _languageName;
}
```

Listing 7: FetLanguage class implementation

5
Testing

5.1. Introduction

This chapter describes the different testing methodologies that were used during the development of the project. In section 5.2, the testing methodology is laid out, while section 5.3 details the test results.

5.2. Testing methodology

An important part of software development is verifying that the product works as intended. The product has to satisfy the functional requirements, ships with as few bugs as possible and still be maintainable to allow further modifications. By continuously testing various aspects of the product, code quality is ensured, bugs are detected as early as possible, code maintainability is improved and debugging is made easier.

In this section, the testing guidelines are described (5.2.1) and the unit test methodology (5.2.2) as well as the integration test methodology used (5.2.3) is explained.

5.2.1. Guidelines

In order to fully utilize the benefits of testing, the following guidelines have been followed throughout the development process:

- Before accepting code in the master branch, all automated (unit) tests have to succeed.
- All code pushed to the master branch has to have a sufficient coverage, aiming for at least 90%.

5.2.2. Unit testing

Unit tests are designed to test the workings of a single method in a class. They verify that a method works as intended and handles all edge cases appropriately. Unit tests can furthermore aid developers by alerting them when causing unintended side effects due to code changes elsewhere.

A number of unit testing frameworks are available for C#, among which the most popular are MSTest, NUnit an xUnit. MSTest is included with Visual Studio, NUnit is a mature testing framework inspired by its Java-counterpart JUnit and xUnit is relatively new. Due to the similarity of NUnit to JUnit, with which the project team already has experience, and the fact that NUnit is widely accepted as a robust and stable C# testing framework by the programming community, this framework was chosen to write all unit tests in.

To separate the test logic from the main codebase, a separate test project named `Timetabling.Tests` has been created in which all unit tests reside. This project follows the same directory and namespace structure as the main project and includes for (almost) all classes corresponding test classes. For example, class `Timetabling.Algorithms.FET.FetAlgorithm` has a corresponding test class named `Timetabling.Tests.`

Algorithms.FET.FetAlgorithmTest in the Timetabling.Tests project. This project structure and naming convention is chosen per Microsoft guidelines[1].

5.2.3. Integration testing

Integration tests in general exist to verify that the various components of a product work together as intended. For this project, most of the functionality was able to be verified with unit tests. The only integration test that has been performed is an all-encompassing check that the product was able to generate a timetable from a given set of timetable resources.

5.3. Test results

5.3.1. Unit tests

A total of 29 test classes comprising 124 unit tests have been executed. The results of these tests can be found in table 5.1.

Namespace	**Uncovered**	**Covered**	**Line coverage**
Timetabling	0	24	100%
Timetabling.Algorithms	37	369	90.0%
Timetabling.Helper	0	131	100%
Timetabling.Objects	5	650	99.2%
Overall	**42**	**1174**	**96.4%**

Table 5.1: Unit test results

The test coverage statistics have been generated by JetBrains dotCover[2], which has been run on the test output of the JetBrains ReSharper Test Runner[3]. The integration test (see section 5.3.2) is not included in this test coverage as to not skew the results.

As can be seen in table 5.1, almost every line of code is covered. Classes in the Timetabling.DB namespace have not been tested and are not included with the test results, because these classes are an automatically generated, Entity Framework-compatible representation of the Key2Soft database. These classes are heavily dependent on the Key2Soft database structure and a working database connection, so automatically testing them would be difficult and meaningless. The usage of these classes, however, is well tested by the use of a mock data model in tests of other parts of the system and by using the Entity Framework Efforthttps://github.com/zzzprojects/EntityFramework-Effort testing library to imitate a working database connection.

The lower than expected coverage in the Timetabling.Algorithms namespace is due to the use of anonymous functions to handle the timetable generation output task (see section 4.6.3). Because the status of C# Task-objects cannot be manually set in tests and the real-world conditions in which these code would be executed are hard to simulate, we have decided not to test this part of the system. If this part would malfunction, it would also be immediately clear to the user because the program would stop functioning altogether. This mitigates the drawbacks of this decision.

5.3.2. Integration tests

The entire functionality of the system can be summarized as generating a timetable from a given set of timetable resources. To test this functionality, just one integration test has been created: testing if a set of timetable resources can be transformed into an acual timetable. Therefore, just one integration test has been created in the FetAlgorithmTest class, where a mock data model by verifying that the output of the program is indeed a timetable when the FetAlgorithm is run on a mock data model.

[1]https://docs.microsoft.com/en-us/visualstudio/test/walkthrough-creating-and-running-unit-tests-for-managed-code
[2]https://www.jetbrains.com/dotcover/
[3]https://www.jetbrains.com/resharper/

5.4. Challenges

A number of challenges arised while creating unit tests for the system.

First, a number of common C# language patterns have been incorporated, which are difficult or meaningless to test. One of these patterns is the Dispose Pattern[4]. This pattern is designed to dispose of class resources on class destruction. Any unit tests designed to test this functionality would in fact test C# language features and the workings of its garbage collector, which usually invokes the dispose methods.

Second, the Task Based Asynchronous Pattern has been implemented to encapsulate and manage the timetable generation task. A common approach to handle task completion is to attach an anonymous, inline function to the task. It has been very difficult to test individual branches within that anonymous function, because the task completion status cannot be set by the programmer.

Third, existing (free) mocking frameworks in C# are unable to test non-virtual methods. By marking a method as virtual, you declare that that method can optionally be overridden in a derived class. Since most of the code of the system was not designed to be overridden, unnecessarily marking methods as virtual was undesirable. Therefore, a number of methods were tested indirectly or using workarounds.

Fourth, since the tests reside in a different project and therefore in a different assembly, protected functions are not visible in the test code. To be able to test these functions, internal exposer classes have been created. These classes are only visible to and usable in the test project, but inherit the original class and redefine protected methods as public. The exposer class of `FetProcessBuilder` in included as listing 8.

```csharp
internal class FetProcessBuilderExposer : FetProcessBuilder
{
    public FetProcessBuilderExposer(string executableLocation = null, IFileSystem fileSystem = null) :
    ↪   base(executableLocation, fileSystem) { }
    public new string GetArgument(string name)             => base.GetArgument(name);
    public new void SetArgument(string name, string value) => base.SetArgument(name, value);
    public new ProcessStartInfo CreateStartInfo()          => base.CreateStartInfo();
}
```

Listing 8: FetProcessBuilder exposer class in test project

Fifth, the interaction with the `FET-CL` process has been difficult to test. To stop the process gracefully, a platform specific signal has to be sent to the process. Because this signal has to be sent from the process where the `FET-CL` process originated from, most C# test runners were unable to execute this test due to their incompatible internal mechanics. We have solved this by altering the tests to make them as compatible as possible and selecting a compatible test runner.

[4]https://docs.microsoft.com/en-us/dotnet/standard/design-guidelines/dispose-pattern

6 Evaluation

6.1. Introduction

This chapter describes the evaluation of the project and requirements. In section 6.2, a table is shown with an evaluation of each requirement. Changes to the predefined requirements during the development are explained in section 6.4. Finally, the feedback that was given from the Software Improvement Group is added in section 6.5.

6.2. Requirements

The following tables represents the initial requirements we did manage to deliver and if not, with an explanation why.

Requirement	**Delivered**
The module *must* be able to produce a feasible timetable, given that the input data allows a feasible timetable to be constructed.	✓
The generated timetable *must* adhere to predefined constraints as detailed in 2.5.2.	see next table
A feasible timetable *must* be generated in less than 30 minutes.	✓
All classes and methods *must* be documented, explaining its use or input, behavior, and output.	✓
Users *should* be allowed to set the importance c.q. priority of each soft constraint by intervals of 10, where 10 is an almost hard constraint (99%) and 0 is a not needed (or inactive) constraint.	✓
The module *should* be able to be integrated seamlessly in Key2School.	✓
The module should load all relevant information to generate a timetable from Key2Soft data sources.	✓
The module *should* be able to generate a timetable without user interaction or user intervention.	✓
A feasible timetable *should* be generated in less than 5 minutes.	✓
The module *should* have at least a 90% branch coverage by using unit tests.	✓
The module *should* be tested with real-world timetable cases.	✓
Violated soft constraints with a priority of more than 50% *should* be reported to the user.	✓
The module *should* retain all generated timetables.	✓

CODE	CONSTRAINT	DELIVERED
PR1	Classes must be scheduled for the required number of meetings for each subject.	✓
PR2	Teachers must be scheduled for the required number of meetings with each class.	✓
NC1	A class must not be scheduled more than once during a time slot.	✓
NC2	A teacher must not be scheduled more than once during a time slot.	✓
NC3	A room must not be scheduled more than once during a time slot.	✓
RU1	Teachers must only be scheduled when available.	✓
RU2	Classes must only be scheduled when available.	✓
RU3	Certain lessons require specialized rooms, e.g. science labs, computer lab, the gymnasium. These requirements must be met.	✓
W1	Classes for a particular grade cannot have more than a predefined number of periods per day.	✓
W2	A limit may be set on the maximum and minimum number of lessons of a subject a class may have per day, e.g. certain grades can not have more than three Mathematics lectures in a day.	X
W3	Teacher workload is defined in terms of a minimum and maximum number of teaching lessons or hours per week or per day.	✓
PD1	A limit may be set of the amount of consecutive lessons for a subject per day.	✓
PD2	The lessons taught by a teacher should be well-spaced throughout the week.	✓
PD3	Certain subjects may have to be taught every day.	✓
L1	Classes may be merged together for a lesson.	✓
L2	Certain lessons must take place simultaneously.	✓

One constraints was not implemented, constraint W2. The FET program does not allow to set a maximum on the number of hours of a specific subject on a day. We have requested this constraint from FET, and this is discussed with the developers and managers of Key2Soft.

Nonetheless, all other constraints and requirements are met.

6.3. Timetable evaluation

To be sure that the timetable generated a plausible real world solution, a generated timetable was manually evaluated by the Key2Soft manager against the timetable that school was currently using. The generated timetable was almost the same as the manually created one, which verified the requirements of the timetable generator.

6.4. Requirements changes during the process

During the development of the system, the following requirements where changed or added by the Key2Soft manager. This was primarily done due to meetings with timetable creators of schools. All of these changes are implemented and met as they are explained below.

6.4.1. Collections

One of the changes was collections. At the beginning, the schools needed shared classes, which where groups of the classes that have the same lesson from the same teacher during the same time slot. So for example, part of class 1A and part of 1B had French, while the others had German, all in the same time slot. This was changed to collections, meaning that both groups have the same collection subject, for example 'Second Language', in the same time slot with two teachers, but not specifying which course. This removed all the groups from classes. The implementation of this can be found in section 4.3.2.

6.4.2. Stages

The timetable generator generated the timetables for the entire school. During the last days of the project, the Key2Soft managers preferred that the generation was done for each specific stage, or a list of stages. A stage is for example the primary stage, which consisted of the first three grades.

6.5. Software Improvement Group

During the development, the Software Improvement Group evaluated the maintainability of the project. This has been done in two phases: one evaluation moment at around 75% project completion and one for the final product.

The maintainability score of the first evaluation was 4.7 out of 5 stars. The highest score was not attained because of lower scores for Unit Size and Module Coupling. Because of this high score, the evaluation team did not have any concrete feedback for improvement of the project, but rather encouraged us to maintain the high score until the end of the project.

The (translated) SIG feedback can be found below, the original (Dutch) feedback can be found in appendix C.1.

> **Feedback:** *"The code of the system scores 4.7 out of 5 stars in our maintainability model, which means that the maintainability of the code is above average. The highest score has not been attained due to lower scores for Unit Size and Model Coupling.*
>
> *At this moment, the score is so high that we have no concrete recommendations for future improvements, congratulations! However, it is important to maintain this level throughout the project, especially when the deadline gets closer."*

The project scored 4.5 out of 5 stars during the evaluation of the final product. The slight decrease was to be expected due to the increase in size of the codebase. Also, the test code increased, which was received well by SIG. In summary, SIG has concluded that their recommendations after the first evaluations were successfully taken into account during the development of the final product.

The (translated) SIG feedback can be found below, the original (Dutch) feedback can be found in appendix C.2.

> **Feedback:** *"In the second upload, we have seen that the project has grown quite a lot. The score for maintainability is compared to the first upload slightly lower. The first score was very high, so a decrease could be expected. However, you still score around 4.5 stars, so there is no reason to worry.*
>
> *Besides the increase in production code, it is good to see that you have also added new test code. The amount of tests still looks well.*
>
> *From these observations, we can conclude that the recommendations from the feedback on the first upload have been incorporated during the development process."*

7 Process and Recommendations

7.1. Introduction

The chapter reflects on the development methodology that was used in section, SCRUM, in section 7.2.2. Furthermore, the development resources used during this project are explained in section 7.3. Finally, some recommendations are made for developers using the system in section 7.4.

7.2. Development methodology

The key components of the development methodology used over the course of this project relied on well-established principles in software engineering. An efficient and transparent development methodology enables a team to manage expectations and progress, streamline collaboration and ensure the quality of the work delivered. The size of a project team is an important factor in determining the right methodology. Large teams might require more effort to keep everyone up to date, while small teams might know what team members are working on at all times.

From our previous experience with software projects, an agile methodology – being able to adapt quickly to changing requirements – is often the key to success. Changing requirements should be viewed as a necessity rather than a problem. Iteratively working on a product and continuously incorporating feedback from the client enables a team to fulfil new wishes and requirements with as little technical debt as possible as customer understanding evolves.

At the start of the project, the overarching goal of the project was apparent: to be able to generate a timetable from a given set of resources. This goal, however, relies on many underlying requirements, e.g. constraints on time and resource availability, execution time, etc. This exact problem definition was unclear at the beginning of the project.

Because of the small size of our team and the unknowns in terms of specific requirements, we have chosen to use an adapted version of the SCRUM methodology.

7.2.1. SCRUM

The SCRUM model is based on a series of development sprints where small, encapsulated parts of a program are developed. In full-fledged development teams, a SCRUM team usually consists of four to eight members with three roles: Scrum Master, Product Owner and Developer. Because the limited size of our team – just two – and the fact that we have worked next to each other during the entire project, we have not explicitly assigned the SCRUM roles, but instead worked collaboratively to maintain a sensible product backlog and determine the deliverables of a sprint.

Due to the limited duration of the project, keeping in mind the time needed to perform research and write a detailed report, we have set the sprint duration to one week and carried out eight sprints in total. Each sprint started with an evaluation of the previous sprint and by determining a sprint goal and backlog. A weekly meeting with our project supervisor at Key2Soft was scheduled to discuss the progress and next steps. Furthermore, every day the team would have a short meeting to discuss that day's planning.

7.2.2. Reflection

We had a weekly sprint meeting at the beginning of the week discussing the goals and any complex choices we needed to make. This worked, as we each knew what to focus on and what to expect to be finished for the next week. This also created a clear planning that we each needed to fulfill to finish the project on time.

Furthermore, before we started, we did arrange with one of the Key2Soft managers to weekly evaluate the status of the project, Unfortunately, this was not done due to his busy schedule. This would however have been useful, especially with the changes in the requirements that were made at the end of the project.

7.3. Development Resources

7.3.1. Slack

Slack was our main communication source with Key2Soft. Any file that was shared or communication with developers that where not in the office that day was done on Slack. This made it easy to contact people, but was not as fast as we hoped for.

7.3.2. GitHub

We used GitHub as source control environment for the project. An issue was made for anything that was needed to be implemented or created. The master branch always has a working solution. Each feature or change that was made needed to be on its own branch. Only when all the testing guidelines where met (see section 5.2.1), a pull request could be merged by the other person. This made sure that each one reviewed the others code.

7.3.3. Waffle

Waffle is a project management tool that uses boards to keep track of issues. This is linked to the issues in GitHub, so that labels changed automatically. This created a better overview. Combining this with the SCRUM method created a simple tool to keep track of each issue.

7.4. Recommendations

As discussed with Key2Soft, the system currently creates a basic timetable. This means that the basic constraints are met, but it is not the ideal solution. To find a more fitting solution for any specific school, manual fine-tuning is a key element. Therefore, a simple manual editor will benefit the user, especially if the editor could show if a constraint is broken during the editing of a timetable.

Furthermore, implementing more constraints will also generate a more ideal solution. But, there is a trade-off, which is speed. Implementing more constraints will create a slower generation. For now, the generation of a timetable takes less than 1 sec in the test data, so some delay is not a problem.

Another recommendation when using this system is to try to check the input data before generating a timetable. For example, if a teacher has only 4 free time slots due to his time off settings, but needs to give 5 lessons, the system will not find a solution. This can easily be caught while the user adds the data into the system.

8 Conclusion and Discussion

The automated generation of timetables is a well known and extensive researched problem. Finding a near-optimal solution is possible in several ways. The best fitting solution depends on the functional system requirements and the required timetable constraints. The main problem of this project was creating a system that generates a near optimum timetable for schools in Egypt. Considering the different constraints and requirements Key2Soft had in mind, such as collections of activities, the free and open source timetabling program "FET" has been selected as the best solution to fulfil the requirements.

One of the main concerns of Key2Soft was the time the system would need to generate a timetable. This project demonstrated that, with real-world sample data, the system could generate a timetable in less than one second.

All the *must have* and *should have* requirements that where determined before starting the project were met, as can be seen in chapter 6.

As discussed with Key2Soft, this project generates only an initial version of the final timetable. End-users of the system will need to fine-tune timetables to create a better solution. Our recommendation is to create a user-friendly and intelligent interface that will support end-users to edit timetables by hand by automatically providing insights into broken constraints when moving an activity.

This bachelor project has been conducted on the premises of Key2Soft in Cairo, Egypt, which presented a lot of different challenges to the team. This meant that not only the project itself needed research, but also everything around that. Egypt has, for example, different living conditions and a totally different culture as opposed to the Netherlands, which impacted our everyday life. This showed us the differences that exist in the world and broadened our knowledge about the Arabic culture. One of the challenges was, for example, that most employees of Key2Soft were not fluent in English. This made communication more difficult than we anticipated. Fortunately, one of the team members was able to speak Arabic, but this was still something we had to adjust to. Another culturally different moment was the Ramadan, which is a month where the entire Egyptian system changes. Work hours change – less and more flexible hours – and people are living more during the evening and night instead of the morning.

It has been a special and very educational experience during which we have learned a lot we will take with us during the remainder of our time at the Delft University of Technology, our career and our lives.

Bibliography

[1] Tim B. Cooper and Jeffrey H. Kingston. “The Complexity of Timetable Construction Problems”. In: 1153 (Mar. 1995).

[2] Andrei Bautu and Elena Bautu. “PRACTICAL ASPECTS ON AUTOMATIC GENERATION OF UNIVERSITY TIMETABLES–A CASE STUDY”. In: (2015).

[3] Grigorios N Beligiannis, C Moschopoulos, and Spiridon D Likothanassis. “A genetic algorithm approach to school timetabling”. In: *Journal of the Operational Research Society* 60.1 (2009), pp. 23–42.

[4] Ruey-Maw Chen and Hsiao-Fang Shih. “Solving university course timetabling problems using constriction particle swarm optimization with local search”. In: *Algorithms* 6.2 (2013), pp. 227–244.

[5] Dai Clegg and Richard Barker. *Case Method Fast-Track: A Rad Approach (Computer Aided System Engineering)*. Addison-Wesley, 1994. ISBN: 020162432X. URL: https://www.amazon.com/Case-Method-Fast-Track-Approach-Engineering/dp/020162432X.

[6] J. Csima. *Investigations on a Time-table Problem*. Thesis–University of Toronto, 1965. URL: https://books.google.com.eg/books?id=1-rDtwAACAAJ.

[7] Dapper. *Dapper*. Retrieved on May 30, 2018. URL: https://github.com/StackExchange/Dapper.

[8] Entity Framework. *.Net Entity Framework*. Retrieved on May 30, 2018. URL: https://docs.microsoft.com/en-us/ef/.

[9] Fred Glover and Eric Taillard. “A user’s guide to tabu search”. In: *Annals of operations research* 41.1 (1993), pp. 1–28.

[10] Dervis Karaboga. “An idea based on honey bee swarm for numerical optimization”. In: (Oct. 2005).

[11] I. V. Katsaragakis, I. X. Tassopoulos, and G. N. Beligiannis. “A comparative study of modern heuristics on the school timetabling problem”. English. In: *Algorithms* 8.3 (2015), pp. 723–742.

[12] J. H. Kingston. *A tiling algorithm for high school timetabling*. English. Vol. 3616 LNCS. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). 2005, pp. 208–225.

[13] L. Lalescu. *FET - Free Timetabling Software*. Retrieved on May 9, 2018. URL: https://lalescu.ro/liviu/fet.

[14] October Resolutions Ltd. *Timetabling Software For Schools | TimeTabler*. Retrieved on May 9, 2018. URL: https://www.timetabler.com/.

[15] Khaled Mahar. “Automatic generation of University timetables: an evolutionary approach”. In: *IADIS International Conference Applied Computing*. 2006, pp. 570–574.

[16] Tomáš Müller. “Real-life examination timetabling”. In: *Journal of Scheduling* 19.3 (June 2016), pp. 257–270. ISSN: 1099-1425. DOI: 10.1007/s10951-014-0391-z. URL: https://doi.org/10.1007/s10951-014-0391-z.

[17] K. Nguyen et al. “Simulated Annealing-based algorithm for a real-world high school timetabling problem”. English. In: *Proceedings - 2nd International Conference on Knowledge and Systems Engineering, KSE 2010*. 2010, pp. 125–130.

[18] Deeptimanta Ojha, Rajesh Sahoo, and Satyabrata Das. “Automated Timetable Generation using Bee Colony Optimization”. In: *International Journal of Applied Information Systems* 10 (May 2016), pp. 38–43.

[19] Deeptimanta Ojha, Rajesh Sahoo, and Satyabrata Das. “Automatic Generation of Timetable Using Firefly Algorithm”. In: 6 (Apr. 2016), pp. 589–593.

[20] Nelishia Pillay. “A survey of school timetabling research”. In: *Annals of Operations Research* 218.1 (2014), pp. 261–293.

[21] Rajesh Sahoo et al. “Automatic generation and optimization of course timetable using a hybrid apraoch”. In: *Journal of Theoretical and Applied Information Technology* 95 (Jan. 2017), pp. 68–77.

[22] aSc Timetables. *aSc TimeTables - School Scheduling. Best timetable software to create school timetable.* Retrieved on May 9, 2018. URL: https://www.asctimetables.com/.

[23] Sandesh Timilsina et al. “Genetically Evolved Solution to Timetable Scheduling Problem”. In: *International Journal of Computer Applications* 114.18 (2015).

[24] UniTime. *UniTime | University Timetabling.* Retrieved on May 9, 2018. URL: http://www.unitime.org/.

[25] Xin-She Yang. “Firefly algorithm, stochastic test functions and design optimisation”. In: *International Journal of Bio-Inspired Computation* 2.2 (2010), pp. 78–84.

A Initial problem definition by Key2Soft

A.1. Project: Automated system to generate optimal school time table

The school timetabling problem is a typical scheduling problem that appears to be a stressful job in every school. Usually, timetable scheduling has done manually with a single person or group of individuals involved in the task of scheduling it manually. Planning of timetable is one of the most complex and depend on individual experience.

There are many factors involved to generate timetable (e.g. teacher availability, school grade, courses and their workload and classes).

Based on related researches, it is possible to claim that in the most of the institutions, the solution is done by manual work, which requires more time and effort. Although the search for the best solution is infeasible, it is possible to find near optimal solutions using heuristics. Most current solutions provide answers optimizing administrative factors, i.e. considering mainly the factors related to the disciplines, classrooms and professors, not considering students needs, e.g. reducing time gaps between non-consecutive classes. In this work, we focus on the timetable improvement for the students and assess our results using real data from a Brazilian university.

These constraints are of two types Hard and Soft constraints. Hard constraints include those constraints that cannot be violated while a timetable is being computed. For example, for a teacher to be scheduled for a timeslot, the teacher must be available for that time slot. A solution is acceptable only when no hard constraint is violated. On the other hand soft constraints are those that are desired to be addressed in the solution as much as possible. For example, though importance is given to a teacher's scheduling, focus is on setting a valid timetable and this can lead to a teacher going free for a time slot. Thus, while addressing the timetabling problem, hard constraints have to be adhered, at the same time effort is made to satisfy as many soft constraints as possible. Due to complexity of the problem, most of the work done concentrates on heuristic algorithms which try to find good approximate solutions

This algorithm is designed to solve and generate school time tables. The following is a list of assumptions made while developing this algorithm:

- The algorithm produces optimum outputs in a five-day week.
- The number of subjects (s_1, s_2, ..., s_n) need to be finalized before the algorithm begins execution.
- Number of teachers (t_1, t_2, ..., t_n) entered before execution of the algorithm are assumed to be constant and cannot be changed during or after the algorithm has been executed.
- Any change in the above two assumptions will require a new generation of Timetable for the changed data.
- In each time table, all time-slot is filled with, a unique combination of subjects without any repetition of subjects.

- Any teacher is allowed at most k number of lectures in a week. The value of k is accepted before execution of the algorithm.
- It is assumed that a teacher cannot take more than one lecture for the same class in a day.
- Timeslots ts_1, ts_2, ..., ts_n once entered at the beginning cannot be changed throughout the execution.
- Every day in the week is assumed to have equal number of time slots.
- Classrooms for any batch id fixed throughout the day.

B

FET-CL

FET-CL is a command-line version of the FET program, without the Graphical User Interface.

B.1. Supported timetable constraints

The following constraints on time and resource allocation are available in FET as of version 5.35.6, provided by the FET online documentation.

B.1.1. Time constraints

The following time-related timetable constraints can be put on the various timetable resources.

- Break periods
- For a single or every teacher:
  - Not available periods
  - Max/min days per week
  - Max gaps per day/week
  - Max hours daily/continuously
  - Max span per day
  - Min hours daily
  - Max hours daily/continuously with an activity tag
  - Respect working in an hourly interval a max number of days per week
  - Min resting hours
- For a single or every student set:
  - Not available periods
  - Max days per week
  - Begins early (specify max allowed beginnings at second hour)
  - Max gaps per day/week
  - Max hours daily/continuously
  - Max span per day
  - Min hours daily
  - Max hours daily/continuously with an activity tag
  - Respect working in an hourly interval a max number of days per week
  - Min resting hours
- For a single activity or a set of activities:
  - A single preferred starting time
  - A set of preferred starting times

  - A set of preferred time slots
  - Min/max days between them
  - End(s) students day
  - Same starting time/day/hour
  - Occupy max time slots from selection (a complex and flexible constraint, useful in many situations)
  - Consecutive, ordered, grouped (for 2 or 3 (sub)activities)
  - Not overlapping
  - Max simultaneous in selected time slots
  - Min gaps between a set of (sub)activities

B.1.2. Space constraints

The following space-related timetable constraints can be put on the various timetable resources.

- Room not available periods
- For teacher(s):
  - Home room(s)
  - Max building changes per day/week
  - Min gaps between building changes
- For students (sets):
  - Home room(s)
  - Max building changes per day/week
  - Min gaps between building changes
- Preferred room(s):
  - For a subject
  - For an activity tag
  - For a subject and an activity tag
  - Individually for a (sub)activity
- For a set of activities:
  - Have the same room if they are consecutive
  - Occupy a maximum number of different rooms

B.2. Command-line arguments

`FET-CL` accepts the following command-line arguments.

version Prints the current FET version.

inputfile The FET input file.

outputdir The path to results directory.

timelimitseconds Maximum duration of the program.

htmllevel Represents the detail level for the generated HTML timetables.

language The output language of the program.

writetimetable[...] Whether or not the corresponding timetables are written to the disk.

printactivitytags Whether or not the activity tags are to be present in the final HTML timetables.

printnotavailable In which style to print the unavailable slots in the generated timetables.

printbreak In which style to print the break slots in the generated timetables.

```
fet-cl --inputfile=x
       [--outputdir=d]
       [--timelimitseconds=y]
       [--htmllevel=z]
       [--language=t]
       [--writetimetableconflicts=wt1]
       [--writetimetablesstatistics=wt2]
       [--writetimetablesxml=wt3]
       [--writetimetablesdayshorizontal=wt4]
       [--writetimetablesdaysvertical=wt5]
       [--writetimetablestimehorizontal=wt6]
       [--writetimetablestimevertical=wt7]
       [--writetimetablessubgroups=wt8]
       [--writetimetablesgroups=wt9]
       [--writetimetablesyears=wt10]
       [--writetimetablesteachers=wt11]
       [--writetimetablesteachersfreeperiods=wt12]
       [--writetimetablesrooms=wt13]
       [--writetimetablessubjects=wt14]
       [--writetimetablesactivities=wt15]
       [--printactivitytags=a]
       [--printnotavailable=u]
       [--printbreak=b]
       [--dividetimeaxisbydays=v]
       [--duplicateverticalheaders=e]
       [--printsimultaneousactivities=w]
       [--randomseedx=rx --randomseedy=ry]
       [--warnifusingnotperfectconstraints=s]
       [--warnifusingstudentsminhoursdailywithallowemptydays=p]
       [--warnifusinggroupactivitiesininitialorder=g]
       [--warnsubgroupswiththesameactivities=ssa]
       [--printdetailedtimetables=pdt]
       [--printdetailedteachersfreeperiodstimetables=pdtfp]
       [--exportcsv=ecsv]
       [--overwritecsv=ocsv]
       [--firstlineisheadingcsv=flhcsv]
       [--quotescsv=qcsv]
       [--fieldseparatorcsv=fscsv]
       [--verbose=r]

fet-cl --version
```

Listing 9: Command-line usage of FET-CL

dividetimeaxisbydays Whether or not the HTML timetables with time-axis are divided by days.

duplicateverticalheaders Whether or not the HTML timetables have duplicate vertical headers to the right of the tables, for easier reading.

printsimultaneousactivities Whether or not the HTML timetables are to show related activities which have constraints with same starting time.

randomseedx, randomseedy The random seed X and Y component. The same timetable is generated if the same seeds and FET-version are used with the same input file.

warnifusingnotperfectconstraints Whether or not a message box with a warning to be is shown if the input file contains imperfectly implemented constraints.

warnifusingstudentsminhoursdailywithallowemptydays Whether or not a message box with a warning is to be shown if the input file contains nonstandard constraints "students min hours daily" with "allow empty days".

warnifusinggroupactivitiesininitialorder Whether or not a message box with a warning is to be shown if the input file contains nonstandard timetable generation options to group activities in the initial order.

warnsubgroupswiththesameactivities Whether or not a message box with a warning is to be shown if the input contains subgroups which have the same activities.

printdetailedtimetables Whether or not to output extra details for the years and groups timetables.

printdetailedteachersfreeperiodstimetables Whether or not to output extra details for teachers free periods timetables.

exportcsv Whether or not to export the CSV file and timetables.

overwritecsv Whether or not to overwrite any existing old CSV files.

firstlineisheadingcsv Whether or not to output headings as first line of the CSV files.

quotescsv The quotation style used for CSV exports.

fieldseparatorcsv The field separator used for CSV exports.

verbose Whether or not to output additional (generation) messages.

B.3. FET XML structure

`FET-CL` accepts an .fet file, which contains an XML file structure. This file contains all the necessary information to construct a timetable. The structure of the file is as follows.

B.3.1. Meta elements

fet Root element.
Attribute: `version` – version used to generate the FET file.

Institution_Name
Name of the institution for which the timetable is generated. Appears in the HTML output.

Comments
When used as a direct child of `fet`: comments to be added to the timetable. Appears in the HTML output. When used in other places: contextual comments regarding its parent element.

B.3.2. Grouping elements

Days_List
Group for elements regarding the structure of a week. Contains an arbitrary number of `Day` elements and one `Number_of_Days` element.

Hours_List
Group for elements regarding the structure of a day. Children can be: `Number_of_Hours`, `Hour`.

Subjects_List
Group for elements regarding the subjects taught. Children can be: `Subject`.

Activity_Tags_List
Group for elements regarding grouped activities. Children can be: `Activity_Tag`.

Teachers_List
Group for elements regarding teachers. Children can be: `Teacher`.

Students_List
Group for elements regarding students and student groups. Children can be: `Year`.

Activities_List
Group for elements regarding activities (a teacher, student group and class combination). Can be empty. Children can be: `Activity`.

Buildings_List
Group for elements regarding the available building. Can be empty. Children can be: `Building`.

Rooms_List
Group for elements regarding available classrooms. Children can be: `Room`.

Time_Constraints_List
Group for time constraints. Please refer to section B.3.5 for the available child elements.

Space_Constraints_List
Group for space constraints. Please refer to section B.3.6 for the available child elements.

B.3.3. Resource elements

Day Describes a day. Contains one element `Name`, whose value is the name of the week.

Hour
Describes a day. Contains one element `Name`, whose value is the name or number of the hour.

Subject
Describes a subject. Contains one element `Name`, whose value is the name of the subject, and one element `Comments`, with optional comments about the subject.

Activity_Tag
Describes an activity tag. Contains one element `Name`, whose value is the name of the tag, one element `Printable`, with a value of *true* if the tag included in the output, and one element `Comments`, with optional comments about the subject.

Teacher
Describes a teacher. Contains one element `Name`, whose value is the name of the teacher, one element `Target_Number_of_Hours`, which contains the preferred number of assigned hours per week, one element of `Qualified_Subjects`, which can contain zero or more `Qualified_Subject` elements, and one element `Comments`, with optional comments about the subject.

Year Describes a group of students, also referred to as grade. Contains a `Name`, `Number_of_Students` and a `Comments` element, and an arbitrary number of `Group` elements.

Group
Describes a student set. Contains a `Name`, `Number_of_Students` and a `Comments` element, and an arbitrary number of `Subgroup` elements.

Subgroup
Describes a student subgroup. Contains a `Name`, `Number_of_Students` and a `Comments` element.

Activity
Describes an activity. Contains a `Teacher`, `Subject`, `Students`, `Duration`, `Total_Duration`, `Id`, `Activity_Group_Id`, `Active` and a `Comments` element.

Building
Describes a building. Contains a `Name` and a `Comments` element.

Room
Describes a room. Contains a `Name`, `Building`, `Capacity` and a `Comments` element.

B.3.4. Reoccurring elements

Each constraint always contains these elements:

ELEMENT	VALUE TYPE	DESCRIPTION
`Weight_Percentage`	integer	Constraint weight. Ranges between 0 and 100.
`Active`	boolean	Whether or not the constraint is active.
`Comments`	string	Optional comments.

B.3.5. Time constraint elements

ConstraintBasicCompulsoryTime

The basic compulsory time constraints, referring to time allocation for any timetable. The basic time constraints try to avoid to assign teachers to more than one activity simultaneously and to assign students to more than one activity simultaneously.

ConstraintBreakTimes

The break times constraint. Allows an arbitrary number of periods to be designated as breaks.

ELEMENT	VALUE TYPE	DESCRIPTION
`Teacher`	string	Name of the teacher to which this constraint applies.
`Number_of_Break_Times`	integer	Amount of `Break_Time` sibling elements.
`Break_Time`	–	Period designated as break. Element contains one `Day` and one `Hour` child element.

Single teacher constraints

ConstraintTeacherNotAvailableTimes

The teacher not available constraint. Allows an arbitrary number of periods to be designated for a teacher to be unavailable.

ELEMENT	VALUE TYPE	DESCRIPTION
`Teacher`	string	Name of the teacher to which this constraint applies.
`Number_of_Not_Available_Times`	integer	Amount of `Not_Available_Time` sibling elements.
`Not_Available_Time`	integer	Period in which the teacher is not available. Element contains one `Day` and one `Hour` child element.

ConstraintTeacherMaxDaysPerWeek

The maximum working days for a teacher constraint.

ELEMENT	VALUE TYPE	DESCRIPTION
`Teacher_Name`	string	Name of the teacher to which this constraint applies.
`Max_Days_Per_Week`	integer	Maximum working days per week for a teacher.

ConstraintTeacherMinDaysPerWeek

The minimum working days for a teacher constraint.

ELEMENT	VALUE TYPE	DESCRIPTION
`Teacher_Name`	string	Name of the teacher to which this constraint applies.
`Minimum_Days_Per_Week`	integer	Minimum working days per week for a teacher.

ConstraintTeacherMaxGapsPerDay

The teacher max gaps per day constraint. A teacher must respect the maximum number of gaps per day (breaks and teacher not available not counted).

ELEMENT	VALUE TYPE	DESCRIPTION
`Teacher_Name`	string	Name of the teacher to which this constraint applies.
`Max_Gaps`	integer	Maximum number of gaps per day for a teacher.

ConstraintTeacherMaxGapsPerWeek

The teacher max gaps per week constraint. A teacher must respect the maximum number of gaps per week (breaks and teacher not available not counted).

ELEMENT	VALUE TYPE	DESCRIPTION
`Teacher_Name`	string	Name of the teacher to which this constraint applies.
`Max_Gaps`	integer	Maximum number of gaps per week for a teacher.

ConstraintTeacherMaxHoursDaily

The teacher max hours per day constraint. A teacher must respect the maximum span in hours per day.

Element	Value type	Description
Teacher_Name	string	Name of the teacher to which this constraint applies.
Maximum_Hours_ Daily	integer	Maximum hours per day for a teacher.

ConstraintTeacherMaxSpanPerDay

The teacher max span per week constraint. A teacher must respect the maximum number of gaps per week (breaks and teacher not available not counted).

Element	Value type	Description
Teacher_Name	string	Name of the teacher to which this constraint applies.
Max_Span	integer	Maximum span per week.

ConstraintTeacherActivityTagMaxHoursDaily

The teacher max hours per activity tag per day constraint.

Element	Value type	Description
Teacher_Name	string	Name of the teacher to which this constraint applies.
Activity_Tag_Name	integer	Activity tag.
Maximum_Hours_ Daily	integer	Maximum hours per day.

ConstraintTeacherMinHoursDaily

The teacher min hours per day constraint.

Element	Value type	Description
Teacher_Name	string	Name of the teacher to which this constraint applies.
Minimum_Hours_ Daily	integer	Minimum hours per day.
Allow_Empty_Days	boolean	Whether or not to allow teachers to have empty days.

ConstraintTeacherMaxHoursContinuously

The teacher max hours continuously constraint.

Element	Value type	Description
Teacher_Name	string	Name of the teacher to which this constraint applies.
Maximum_Hours_ Continuously	integer	Maximum consecutive hours.

ConstraintTeacherActivityTagMaxHoursContinuously

The teacher max hours continuously constraint. **N.B.** implementation not perfect according to author.

Element	Value type	Description
Teacher_Name	string	Name of the teacher to which this constraint applies.
Maximum_Hours_ Continuously	integer	Maximum consecutive hours.

ConstraintTeacherIntervalMaxDaysPerWeek

The teacher max hours continuously constraint.

Element	Value type	Description
Teacher_Name	string	Name of the teacher to which this constraint applies.
Interval_Start_ Hour	string	Name of the start period.
Interval_End_Hour	string|void	Name of the end period. Void (empty) means end of the day.
Max_Days_Per_Week	integer	Maximum days per week.

ConstraintTeacherMinRestingHours

The teacher max hours continuously constraint.

Element	Value type	Description
Teacher_Name	string	Name of the teacher to which this constraint applies.
Minimum_Resting_Hours	integer	Minimum number of resting hours.
Circular	boolean	???

Constraints for all teachers

ConstraintTeachersNotAvailableTimes

The teacher not available constraint. Allows an arbitrary number of periods to be designated for a teacher to be unavailable.

Element	Value type	Description
Number_of_Not_Available_Times	integer	Amount of Not_Available_Time sibling elements.
Not_Available_Time	integer	Period in which the teachers are not available. Element contains one Day and one Hour child element.

ConstraintTeachersMaxDaysPerWeek

The maximum working days for all teachers constraint.

Element	Value type	Description
Max_Days_Per_Week	integer	Maximum working days per week for each teacher.

ConstraintTeachersMinDaysPerWeek

The minimum working days for all teachers constraint.

Element	Value type	Description
Minimum_Days_Per_Week	integer	Minimum working days per week for each teacher.

ConstraintTeachersMaxGapsPerDay

The all teachers max gaps per day constraint.

Element	Value type	Description
Max_Gaps	integer	Maximum number of gaps per day for each teacher.

ConstraintTeachersMaxGapsPerWeek

The all teachers max gaps per week constraint.

Element	Value type	Description
Max_Gaps	integer	Maximum number of gaps per week for each teacher.

ConstraintTeachersMaxHoursDaily

The all teachers max hours per day constraint.

Element	Value type	Description
Maximum_Hours_Daily	integer	Maximum hours per day for each teacher.

ConstraintTeachersMaxSpanPerDay

The all teachers max span per week constraint.

Element	Value type	Description
Max_Span	integer	Maximum span per week.

ConstraintTeachersActivityTagMaxHoursDaily

The all teachers max hours per activity tag per day constraint.

Element	Value type	Description
Activity_Tag_Name	integer	Activity tag.
Maximum_Hours_Daily	integer	Maximum hours per day.

ConstraintTeachersMinHoursDaily
The all teachers min hours per day constraint.

Element	Value type	Description
`Minimum_Hours_Daily`	integer	Minimum hours per day for each teacher.
`Allow_Empty_Days`	boolean	Whether or not to allow teachers to have empty days.

ConstraintTeachersMaxHoursContinuously
The all teachers max hours continuously constraint.

Element	Value type	Description
`Maximum_Hours_Continuously`	integer	Maximum consecutive hours.

ConstraintTeachersActivityTagMaxHoursContinuously
The all teachers max hours continuously constraint.

Element	Value type	Description
`Maximum_Hours_Continuously`	integer	Maximum consecutive hours.

ConstraintTeachersIntervalMaxDaysPerWeek
The all teachers max hours continuously constraint.

Element	Value type	Description
`Interval_Start_Hour`	string	Name of the start period.
`Interval_End_Hour`	string|void	Name of the end period. Void (empty) means end of the day.
`Max_Days_Per_Week`	integer	Maximum days per week.

ConstraintsTeachersMinRestingHours
The all teachers max hours continuously constraint.

Element	Value type	Description
`Minimum_Resting_Hours`	integer	Minimum number of resting hours.
`Circular`	boolean	???

Single student constraints

ConstraintStudentNotAvailableTimes The student not available constraint. Allows an arbitrary number of periods to be designated for a student to be unavailable.

Element	Value type	Description
`Student`	string	Name of the student to which this constraint applies.
`Number_of_Not_Available_Times`	integer	Amount of `Not_Available_Time` sibling elements.
`Not_Available_Time`	integer	Period in which the student is not available. Element contains one `Day` and one `Hour` child element.

ConstraintStudentMaxDaysPerWeek The maximum working days for a student constraint.

Element	Value type	Description
`Student_Name`	string	Name of the student to which this constraint applies.
`Max_Days_Per_Week`	integer	Maximum working days per week for a student.

ConstraintStudentMinDaysPerWeek The minimum working days for a student constraint.

Element	Value type	Description
`Student_Name`	string	Name of the student to which this constraint applies.
`Minimum_Days_Per_Week`	integer	Minimum working days per week for a student.

ConstraintStudentMaxGapsPerDay The student max gaps per day constraint. A student must respect the maximum number of gaps per day (breaks and student not available not counted).

ELEMENT	VALUE TYPE	DESCRIPTION
`Student_Name`	string	Name of the student to which this constraint applies.
`Max_Gaps`	integer	Maximum number of gaps per day for a student.

ConstraintStudentMaxGapsPerWeek The student max gaps per week constraint. A student must respect the maximum number of gaps per week (breaks and student not available not counted).

ELEMENT	VALUE TYPE	DESCRIPTION
`Student_Name`	string	Name of the student to which this constraint applies.
`Max_Gaps`	integer	Maximum number of gaps per week for a student.

ConstraintStudentMaxHoursDaily The student max hours per day constraint. A student must respect the maximum span in hours per day.

ELEMENT	VALUE TYPE	DESCRIPTION
`Student_Name`	string	Name of the student to which this constraint applies.
`Maximum_Hours_Daily`	integer	Maximum hours per day for a student.

ConstraintStudentMaxSpanPerDay The student max span per week constraint. A student must respect the maximum number of gaps per week (breaks and student not available not counted).

ELEMENT	VALUE TYPE	DESCRIPTION
`Student_Name`	string	Name of the student to which this constraint applies.
`Max_Span`	integer	Maximum span per week.

ConstraintStudentActivityTagMaxHoursDaily The student max hours per activity tag per day constraint.

ELEMENT	VALUE TYPE	DESCRIPTION
`Student_Name`	string	Name of the student to which this constraint applies.
`Activity_Tag_Name`	integer	Activity tag.
`Maximum_Hours_Daily`	integer	Maximum hours per day.

ConstraintStudentMinHoursDaily The student min hours per day constraint.

ELEMENT	VALUE TYPE	DESCRIPTION
`Student_Name`	string	Name of the student to which this constraint applies.
`Minimum_Hours_Daily`	integer	Minimum hours per day.
`Allow_Empty_Days`	boolean	Whether or not to allow students to have empty days.

ConstraintStudentMaxHoursContinuously The student max hours continuously constraint.

ELEMENT	VALUE TYPE	DESCRIPTION
`Student_Name`	string	Name of the student to which this constraint applies.
`Maximum_Hours_Continuously`	integer	Maximum consecutive hours.

ConstraintStudentActivityTagMaxHoursContinuously The student max hours continuously constraint. **N.B.** implementation not perfect according to author.

ELEMENT	VALUE TYPE	DESCRIPTION
`Student_Name`	string	Name of the student to which this constraint applies.
`Maximum_Hours_Continuously`	integer	Maximum consecutive hours.

ConstraintStudentIntervalMaxDaysPerWeek The student max hours continuously constraint.

Element	Value type	Description
`Student_Name`	string	Name of the student to which this constraint applies.
`Interval_Start_Hour`	string	Name of the start period.
`Interval_End_Hour`	string|void	Name of the end period. Void (empty) means end of the day.
`Max_Days_Per_Week`	integer	Maximum days per week.

ConstraintStudentMinRestingHours The student max hours continuously constraint.

Element	Value type	Description
`Student_Name`	string	Name of the student to which this constraint applies.
`Minimum_Resting_Hours`	integer	Minimum number of resting hours.
`Circular`	boolean	???

Constraints for all students

ConstraintStudentsNotAvailableTimes The student not available constraint. Allows an arbitrary number of periods to be designated for a student to be unavailable.

Element	Value type	Description
`Number_of_Not_Available_Times`	integer	Amount of `Not_Available_Time` sibling elements.
`Not_Available_Time`	integer	Period in which the students are not available. Element contains one `Day` and one `Hour` child element.

ConstraintStudentsMaxDaysPerWeek The maximum working days for all students constraint.

Element	Value type	Description
`Max_Days_Per_Week`	integer	Maximum working days per week for each student.

ConstraintStudentsMinDaysPerWeek The minimum working days for all students constraint.

Element	Value type	Description
`Minimum_Days_Per_Week`	integer	Minimum working days per week for each student.

ConstraintStudentsMaxGapsPerDay The all students max gaps per day constraint.

Element	Value type	Description
`Max_Gaps`	integer	Maximum number of gaps per day for each student.

ConstraintStudentsMaxGapsPerWeek The all students max gaps per week constraint.

Element	Value type	Description
`Max_Gaps`	integer	Maximum number of gaps per week for each student.

ConstraintStudentsMaxHoursDaily The all students max hours per day constraint.

Element	Value type	Description
`Maximum_Hours_Daily`	integer	Maximum hours per day for each student.

ConstraintStudentsMaxSpanPerDay The all students max span per week constraint.

Element	Value type	Description
`Max_Span`	integer	Maximum span per week.

ConstraintStudentsActivityTagMaxHoursDaily The all students max hours per activity tag per day constraint.

Element	Value type	Description
`Activity_Tag_Name`	integer	Activity tag.
`Maximum_Hours_Daily`	integer	Maximum hours per day.

ConstraintStudentsMinHoursDaily The all students min hours per day constraint.

Element	Value type	Description
`Minimum_Hours_Daily`	integer	Minimum hours per day for each student.
`Allow_Empty_Days`	boolean	Whether or not to allow students to have empty days.

ConstraintStudentsMaxHoursContinuously The all students max hours continuously constraint.

Element	Value type	Description
`Maximum_Hours_Continuously`	integer	Maximum consecutive hours.

ConstraintStudentsActivityTagMaxHoursContinuously The all students max hours continuously constraint.

Element	Value type	Description
`Maximum_Hours_Continuously`	integer	Maximum consecutive hours.

ConstraintStudentsIntervalMaxDaysPerWeek The all students max hours continuously constraint.

Element	Value type	Description
`Interval_Start_Hour`	string	Name of the start period.
`Interval_End_Hour`	string|void	Name of the end period. Void (empty) means end of the day.
`Max_Days_Per_Week`	integer	Maximum days per week.

ConstraintsStudentsMinRestingHours The all students max hours continuously constraint.

Element	Value type	Description
`Minimum_Resting_Hours`	integer	Minimum number of resting hours.
`Circular`	boolean	???

Student set constraints

ConstraintStudentsSetNotAvailableTimes The teacher not available constraint. Allows an arbitrary number of periods to be designated for a teacher to be unavailable.

Element	Value type	Description
`Students`	string	Name of the student set to which this constraint applies.
`Number_of_Not_Available_Times`	integer	Amount of `Not_Available_Time` sibling elements.
`Not_Available_Time`	integer	Period in which the teacher is not available. Element contains one `Day` and one `Hour` child element.

All students constraints

B.3.6. Space constraint elements

ConstraintBasicCompulsorySpace
The basic compulsory space constraints, referring to space allocation for any timetable. The basic space constraints try to avoid rooms assigned to more than one activity simultaneously and activities with more students than the capacity of the room.

Element	Value type	Description
`Weight_Percentage`	integer	Constraint weight. Ranges between 0 and 100.
`Active`	boolean	Whether or not the constraint is active.
`Comments`	string	Optional comments.

ConstraintSubjectPreferredRoom
The preferred room for a specific subject.

Element	Value type	Description
`Subject`	string	Subject name.
`Room`	string	Room name.

ConstraintSubjectPreferredRooms
A list of preferred rooms for a specific subject.

Element	Value type	Description
`Subject`	string	Subject name.
`Number_of_ Preferred_Rooms`	integer	Number of preferred rooms
`Room`	string	A room element for each room name.

ConstraintActivityPreferredRoom
The preferred room for a specific activity.

Element	Value type	Description
`Activity_Id`	integer	Activity id.
`Room`	string	Room name.

ConstraintActivityPreferredRooms
A list of preferred rooms for a specific activity.

Element	Value type	Description
`Activity_Id`	integer	Activity id .
`Number_of_ Preferred_Rooms`	integer	Number of preferred rooms
`Room`	string	A `Room` element for each room name.

ConstraintActivitiesSameRoomIfConsecutive
A list of activities which can be in the same room if they are consecutive.

Element	Value type	Description
`Number_of_ Activities`	integer	Number of activities
`Activity_Id`	integer	A `Activity_Id` element for each activity.

ConstraintActivityTagPreferredRoom
The preferred room for a specific activity tag.

Element	Value type	Description
`Activity_Tag`	string	Activity tag name.
`Room`	string	A room element for each room name.

ConstraintActivityTagPreferredRooms
A list of preferred rooms for a specific activity tag.

Element	Value type	Description
`Activity_Tag`	string	Activity tag name.
`Number_of_ Preferred_Rooms`	integer	Number of preferred rooms
`Room`	string	A room element for each room name.

ConstraintTeacherHomeRoom
The preferred home room for a specific teacher.

Element	Value type	Description
`Teacher`	string	Teacher name.
`Room`	string	Room name.

ConstraintTeacherHomeRooms
A list of preferred home rooms for a specific teacher.

Element	Value type	Description
Teacher	string	Teacher name.
Number_of_ Preferred_Rooms	integer	Number of preferred rooms
Room	string	A room element for each room name.

ConstraintTeacherMaxBuildingChangesPerDay
The maximum number of building changes a day a specific teacher can have.

Element	Value type	Description
Teacher	string	Teacher name.
Max_Building_ Changes_Per_Days	integer	Number of max building changes

ConstraintTeacherMaxBuildingChangesPerWeek
The maximum number of building changes per week a specific teacher can have.

Element	Value type	Description
Teacher	string	Teacher name.
Max_Building_ Changes_Per_Week	integer	Number of max building changes

ConstraintTeacherMinGapsBetweenBuildingChanges
The minimum number of free time between building changes.

Element	Value type	Description
Teacher	string	Teacher name.
Min_Gaps_Between_ Building_Changes	integer	Number of min time slots

ConstraintTeachersMaxBuildingChangesPerDay
The maximum number of building changes a day all teachers can have.

Element	Value type	Description
Max_Building_ Changes_Per_Days	integer	Number of max building changes

ConstraintTeachersMaxBuildingChangesPerWeek
The maximum number of building changes a week all teachers can have.

Element	Value type	Description
Max_Building_ Changes_Per_Week	integer	Number of max building changes

ConstraintStudentsMaxBuildingChangesPerDay
The maximum number of building changes a day all students can have.

Element	Value type	Description
Max_Building_ Changes_Per_Day	integer	Number of max building changes

ConstraintStudentsMaxBuildingChangesPerWeek
The maximum number of building changes a week all students can have.

Element	Value type	Description
Max_Building_ Changes_Per_Week	integer	Number of max building changes

ConstraintStudentsSetHomeRoom
The specific home room of a specific student group or class.

Element	Value type	Description
Students	string	Students group or class name.
Room	string	Room name.

ConstraintStudentsSetHomeRooms

The list of home rooms of a specific student group or class.

ELEMENT	VALUE TYPE	DESCRIPTION
`Students`	string	Students group or class name.
`Number_of_Preferred_Rooms`	integer	Number of preferred rooms
`Room`	string	A `Room` element for each room name.

ConstraintStudentsSetMaxBuildingChangesPerDay

The maximum number of building changes a day specific students groups/classes/years can have.

ELEMENT	VALUE TYPE	DESCRIPTION
`Students`	string	A `Students` element with the group name for each group.
`Max_Building_Changes_Per_Day`	integer	Number of max building changes

ConstraintStudentsSetMaxBuildingChangesPerWeek

The maximum number of building changes a week specific students groups/classes/years can have.

ELEMENT	VALUE TYPE	DESCRIPTION
`Students`	string	A `Students` element with the group name for each group.
`Max_Building_Changes_Per_Week`	integer	Number of max building changes

ConstraintStudentsSetMinGapsBetweenBuildingChanges

The minimum number of free time for specific students groups/classes/years between building changes.

ELEMENT	VALUE TYPE	DESCRIPTION
`Students`	string	A `Students` element with the as group name.
`Min_Gaps_Between_Building_Changes`	integer	Number of min time slots

ConstraintStudentsMinGapsBetweenBuildingChanges

The minimum number of free time for all students between building changes.

ELEMENT	VALUE TYPE	DESCRIPTION
`Min_Gaps_Between_Building_Changes`	integer	Number of min time slots

ConstraintRoomNotAvailableTimes

The time slots a room is not available.

ELEMENT	VALUE TYPE	DESCRIPTION
`Room`	string	Room name
`Number_of_Not_Available_Times`	integer	Number of not available time slots
`Not_Available_Time`		Element contains one `Day` and one `Hour` child element.

ConstraintSubjectActivityTagPreferredRoom

The preferred room for a specific activity tag

ELEMENT	VALUE TYPE	DESCRIPTION
`Subject`	string	Subject name
`Activity_tag`	string	Activity tag name
`Room`	string	Room name

ConstraintSubjectActivityTagPreferredRooms

A list of preferred rooms for a specific activity tag

ELEMENT	VALUE TYPE	DESCRIPTION
Subject	string	Subject name
Activity_tag	string	Activity tag name
Number_of_Preferred_Rooms	integer	The number of preferred rooms.
Preferred_Room	string	A Preferred_Room element for each room.

B.3.7. Miscellaneous elements

Number_of_Days
Describes the number of Day siblings of this element.

Number_of_Hours
Describes the number of Hour siblings of this element.

Name
Contains the name associated with its parent element.

B.4. Basic FET XML file example

Please find an example FET file with one teacher, one subject, one activity, one room and a week of five days with five periods in listing 10.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<fet version="5.36.0">
        <Institution_Name>Institution Name</Institution_Name>
        <Comments>Example FET file.</Comments>
        <Days_List>
                <Number_of_Days>5</Number_of_Days>
                <Day><Name>Monday</Name></Day>
                <Day><Name>Tuesday</Name></Day>
                <Day><Name>Wednesday</Name></Day>
                <Day><Name>Thursday</Name></Day>
                <Day><Name>Friday</Name></Day>
        </Days_List>
        <Hours_List>
                <Number_of_Hours>8</Number_of_Hours>
                <Hour><Name>09:00</Name></Hour>
                <Hour><Name>10:00</Name></Hour>
                <Hour><Name>11:00</Name></Hour>
                <Hour><Name>12:00</Name></Hour>
                <Hour><Name>13:00</Name></Hour>
        </Hours_List>
        <Subjects_List>
                <Subject>
                        <Name>Bachelor End Project</Name>
                        <Comments></Comments>
                </Subject>
        </Subjects_List>
        <Activity_Tags_List>
        </Activity_Tags_List>
        <Teachers_List>
                <Teacher>
                        <Name>T.E.A. Cher</Name>
                        <Target_Number_of_Hours>0</Target_Number_of_Hours>
                        <Qualified_Subjects></Qualified_Subjects>
                        <Comments></Comments>
                </Teacher>
        </Teachers_List>
        <Students_List>
                <Year>
                        <Name>2018</Name>
                        <Number_of_Students>0</Number_of_Students>
                        <Comments></Comments>
                        <Group>
```

```
                                <Name>Bachelor Year 3</Name>
                                <Number_of_Students>2</Number_of_Students>
                                <Comments></Comments>
                        </Group>
                </Year>
        </Students_List>
        <Activities_List>
                <Activity>
                        <Teacher>T.E.A. Cher</Teacher>
                        <Subject>Bachelor End Project</Subject>
                        <Students>Bachelor Year 3</Students>
                        <Duration>3</Duration>
                        <Total_Duration>3</Total_Duration>
                        <Id>1</Id>
                        <Activity_Group_Id>0</Activity_Group_Id>
                        <Number_Of_Students>2</Number_Of_Students>
                        <Active>true</Active>
                        <Comments></Comments>
                </Activity>
        </Activities_List>
        <Buildings_List>
                <Building>
                        <Name>Key2Soft Office</Name>
                        <Comments></Comments>
                </Building>
        </Buildings_List>
        <Rooms_List>
        </Rooms_List>
        <Time_Constraints_List>
                <ConstraintBasicCompulsoryTime>
                        <Weight_Percentage>100</Weight_Percentage>
                        <Active>true</Active>
                        <Comments></Comments>
                </ConstraintBasicCompulsoryTime>
        </Time_Constraints_List>
        <Space_Constraints_List>
                <ConstraintBasicCompulsorySpace>
                        <Weight_Percentage>100</Weight_Percentage>
                        <Active>true</Active>
                        <Comments></Comments>
                </ConstraintBasicCompulsorySpace>
        </Space_Constraints_List>
</fet>
```

Listing 10: Example FET file

C
Software Improvement Group (SIG) feedback

C.1. First evaluation

Reviewer: Dennis Bijlsma (d.bijlsma@sig.eu)
Date: June 11, 2018
Language: Dutch

"De code van het systeem scoort 4.7 sterren op ons onderhoudbaarheidsmodel, wat betekent dat de code bovengemiddeld onderhoudbaar is. De hoogste score is niet behaald door lagere scores voor Unit Size en Module Coupling.

Op dit moment is de score dusdanig hoog dat we geen concrete aanbevelingen voor verdere verbetering hebben, hulde! Wel is het zaak om ervoor te zorgen dat jullie dit niveau tijdens het vervolg van het project vast weten te houden, en al helemaal op het moment dat de deadline in zicht komt.

Als laatste nog de opmerking dat er geen (unit)test-code is gevonden in de code-upload. Het is sterk aan te raden om in ieder geval voor de belangrijkste delen van de functionaliteit automatische tests gedefinieerd te hebben om ervoor te zorgen dat eventuele aanpassingen niet voor ongewenst gedrag zorgen.

Over het algemeen scoort de code dus bovengemiddeld, hopelijk lukt het om dit niveau te behouden tijdens de rest van de ontwikkelfase."

C.2. Second evaluation

Reviewer: Dennis Bijlsma (d.bijlsma@sig.eu)
Date: July 2, 2018
Language: Dutch

"In de tweede upload zien we dat het project een stuk groter is geworden. De score voor onderhoudbaarheid is in vergelijking met de eerste upload iets gedaald. Jullie zaten qua score erg hoog, dus die daling was in dat opzicht te verwachten. Jullie zitten echter nog steeds rond de 4,5 ster, dus geen reden om je zorgen te maken.

Zoals jullie per email hebben aangegeven hadden jullie in de eerste upload wel degelijk testcode, dus die opmerking uit de feedback op de eerste upload komt te vervallen. Naast de toename in de hoeveelheid productiecode is het goed om te zien dat jullie ook nieuwe testcode hebben toegevoegd. De hoeveelheid tests ziet er dan ook nog steeds goed uit.

Uit deze observaties kunnen we concluderen dat de aanbevelingen uit de feedback op de eerste upload zijn meegenomen tijdens het ontwikkeltraject."

Information sheet

- **Project Title:** Automated timetable generation for Egyptian schools
- **Client Organization:** Key2Soft
- **Final Presentation:** July 12, 2018 at 11:00 AM

Description

Creating a timetable for primary-, middle- and high school is a time-consuming process which is currently usually be done by hand. During the course of this project, an automated system to generate timetables for schools in Egypt has been implemented, taking into account the various requirements and timetable constraints of these schools. During the research phase, existing solutions that would fulfill the requirements of Key2Soft have been analyzed. The biggest challenge was finding a solution that finds a near optimal solution and is still fast enough for the client. From the analysis of the research phase, the free and open source timetable generator FET has been selected to generate timetables in this project, mainly due to its speed. A SCRUM variant was used to efficiently work together as a project team on this project.

The final product creates an initial timetable which satisfies all the requirements that were defined. To create a better or optimal timetable, the timetable generated by the system will need to be manually fine-tuned.

Members of the project team

- **Name:** Sven van Hal
- **Interests:** Software Engineering, customer interaction, system integration.
- **Contribution:** Timetabling component integration, timetable output processing, quality assurance, testing and refactoring, open-source.

- **Name:** Karim Osman
- **Interests:** Software Engineering
- **Contribution:** Connection with database, DataModel and Resource objects creator

Client

- **Name:** Mahmoud ElHefnawy
- **Affiliation:** CEO, Key2Soft

Coach

- **Name:** Felienne Hermans
- **Affiliation:** Software Engineering Department, Delft University of Technology

Contact information

Team members:

- **Sven van Hal** (sven@svenvanhal.nl)
- **Karim Osman** (karimosman1@hotmail.com)

The final report for this project can be found at: https://repository.tudelft.nl