Bachelor Thesis

August 23, 2017

DayO
Optimizing Workdays Based on Time and Productivity Constraints

Lada Murchych

of Belgorod, Russia, Ukraine (13-743-448)

supervised by

Prof. Dr. Thomas Fritz
Andre Meyer
Jürgen Cito

University of Zurich^UZH

s.e.a.l. software evolution & architecture lab

Bachelor Thesis

DayO
Optimizing Workdays Based on Time and Productivity Constraints

Lada Murchych

Bachelor Thesis
Author: Lada Murchych, lada.murchych@uzh.ch
Project period: 18.03.2017 - 24.08.2017

Software Evolution & Architecture Lab
Department of Informatics, University of Zurich

Acknowledgements

I would like to express sincere gratitude to all people who have helped with this thesis and supported me during this project. I am very grateful to my supervisors who have given valuable advice, made interesting suggestions for application features as well as research, and always provided qualified feedback. Moreover, I would like to thank the participants of the evaluation who have agreed to spend one week of their lives using the DayO application and answering endless questionnaires. My deepest gratitude goes to my husband who has supported me during this project, giving valuable advice regarding implementation and helped to proof-read this thesis.

Abstract

Knowledge workers have multiple competing responsibilities, ranging from answering emails, attending meetings to performing highly demanding tasks. To be successful at work and get things done, workers have to continuously plan ahead or determine what's best to work on right now. Planning is vital for completing projects and accomplishing longer tasks, since keeping a high-level picture in mind while simultaneously concentrating on the solution is challenging. Productivity of the knowledge worker depends on many different factors, ranging from environmental like interruptions to personal like sleep duration and quality. However, none of the existing approaches for automated task scheduling takes personal productivity factors into account.

To alleviate the planning effort and help knowledge workers in allocating their personal resources of time and energy to achieve effective task execution, we have devised a concept of constraint-based workday planning and implemented as a prototype, DayO. Taking the list of tasks created by the user, DayO generates schedule suggestions based on task characteristics, appointments scheduled for today and productivity factors. The evaluation of DayO with knowledge workers has shown that suggestions of workday plan display high accuracy and that concept presented in this thesis can be used for task scheduling at work.

Zusammenfassung

Wissensarbeiter haben mehrere konkurrierende Verantwortlichkeiten, von der Beantwortung von E-Mails, Teilnahme an Sitzungen, bis zur Erfüllung von anspruchsvollen Aufgaben. Um auf der Arbeit erfolgreich zu sein und die gestellten Aufgaben zu erledigen, müssen Wissensarbeiter kontinuierlich planen oder jederzeit festlegen, welche Aufgabe zur Zeit am wichtigsten ist. Die Planung ist entscheidend für die Abwicklung von Projekten und die Erfüllung von längeren Aufgaben. Die Aufrechterhaltung einer Perspektive von mehreren Aufgaben und gleichzeitiger Konzentration auf einer Lösung ist eine Herausforderung. Die Produktivität des Wissensarbeiters hängt von vielen verschiedenen Faktoren ab, von externen Faktoren wie Unterbrechungen bis hin zu persönlichen Produktivitätsfaktoren wie Schlafdauer und Schlafqualität. Allerdings berücksichtigt keines der vorhandenen Werkzeuge für die automatisierte Aufgabenplanung die persönlichen Produktivitätsfaktoren.

Um den Planungsaufwand zu reduzieren und den Wissensarbeitern bei der Zuteilung ihrer persönlichen Ressourcen von Zeit und Energie zu helfen, die eine effektive Aufgabenausführung zu erreichen erlaubt, haben wir ein Konzept der bedingungsbasierten Arbeitstagplanung entwickelt und in einem Prototyp, DayO, umgesetzt. Aus der Liste der vom Benutzer erstellten Aufgaben generiert DayO Zeitplanvorschläge für den Arbeitstag basierend auf der Analyse von Aufgabenmerkmalen, für heute geplanten Terminen und Produktivitätsfaktoren. Die Auswertung von DayO in einer Studie mit Wissensarbeitern hat gezeigt, dass die generierten Vorschläge für den Arbeitstagplan eine hohe Genauigkeit aufweisen, und dass das in dieser Arbeit vorgestellte Konzept für die Aufgabenplanung bei Wissensarbeitern verwendet werden kann.

Contents

1 Introduction **1**

2 Background **5**
- 2.1 Planning Problem . . . 5
- 2.2 Optimization Problem . . . 6
- 2.3 Constraint-Satisfaction Problem (CSP) . . . 6
  - 2.3.1 Formal Definition . . . 7
  - 2.3.2 Example: N-Queens . . . 7
  - 2.3.3 Solving CSP . . . 8
  - 2.3.4 Applications . . . 9

3 Related Work **11**
- 3.1 Planning Productive Day . . . 11
- 3.2 Automatic Task Scheduling . . . 13
  - 3.2.1 Task Scheduling Concepts and Approaches . . . 13
  - 3.2.2 Existing Applications with Automatic Task Scheduling . . . 14
- 3.3 Factors Influencing Personal Productivity . . . 16
  - 3.3.1 Sleep Duration and Quality . . . 17
  - 3.3.2 Morningness and Eveningness as Factors of Personal Productivity . . . 18
  - 3.3.3 Stress Level and Productivity . . . 19
  - 3.3.4 Other Productivity Factors . . . 20

4 Constraint-based Workday Planning **23**
- 4.1 Overview . . . 23
  - 4.1.1 Prerequisites . . . 25
- 4.2 CSP Formulation . . . 27
  - 4.2.1 Rules for Assignment Optimization . . . 29

5 DayO Prototype **35**
- 5.1 Overview . . . 35
- 5.2 System Architecture . . . 36
- 5.3 Client . . . 38
  - 5.3.1 Components . . . 38
  - 5.3.2 User Interface Design . . . 40
  - 5.3.3 Interaction and Navigation . . . 43
- 5.4 Server . . . 46
  - 5.4.1 Domain Classes . . . 47
  - 5.4.2 Application Logic Classes . . . 51
  - 5.4.3 Controllers . . . 54

5.4.4 Data Access Classes and DTOs . . . 55
5.5 Data Model . . . 55
5.6 Incorporated APIs . . . 57

6 Evaluation **59**
6.1 Evaluation Method . . . 59
6.2 Results . . . 61
6.2.1 Quantitative Results . . . 61
6.2.2 Qualitative Results . . . 63
6.2.3 User Feedback . . . 67
6.3 Further Improvements . . . 68
6.4 Limitations and Conclusions . . . 69

7 Discussion **71**
7.1 Discussion . . . 71
7.2 Future Work . . . 73

8 Conclusions **75**
8.1 Summary . . . 75
8.2 Contributions of This Thesis . . . 76

9 Appendix **77**
9.1 Additional UI Screens . . . 77
9.2 Code Listings . . . 78
9.2.1 CSP Solving Execution . . . 78
9.2.2 Score Calculation . . . 79
9.2.3 Rules . . . 82
9.2.4 Solver Configuration . . . 84
9.3 Diagrams . . . 86
9.4 Evaluation . . . 87

List of Figures

4.1 Overview of schedule generation prerequisites and process 24
4.2 Relationship between task attributes and rules 26
4.3 Score distribution for sleep quality from good to very bad and sleep duration from 0 to 9.5 hours 32
4.4 Urgency score distribution by the workdays until deadline 33

5.1 Architecture overview 37
5.2 Packages of frontend and backend module 38
5.3 Overview of the structure of dashboard package on the frontend 39
5.4 Screen with daily form 41
5.5 Schedule and task list view with tool-tips 42
5.6 Screen with generated schedule options 43
5.7 View of the dashboard when user has no tasks 44
5.8 Activity diagram of user interaction and navigation 45
5.9 Domain classes which are used for optimization with OptaPlanner 49
5.10 Hard constraints implemented in DayAssignmentScoreCalculator class 53
5.11 Entity-relationship model of the entities stored in the database 56

6.1 Hard and soft scores of all generated schedules for study participants 63
6.2 The perceived quality of the chosen schedule assessed after the schedule was chosen 64
6.3 Statistics of following schedule during the day assessed in the evening of the same day 65
6.4 The perceived productivity assessed at the end of workday 66

9.1 Settings form 77
9.2 Completing and viewing completed tasks 78
9.3 UML class diagram of classes in domain_classes package 86
9.4 Sequence diagram of the schedule generation process 87
9.5 Table of schedules generated during user evaluation with scores and user choice . 89

List of Tables

6.1 Questionnaires used during user evaluation 60
6.2 Tasks statistics 61
6.3 Events statistics 62

List of Listings

5.1 An example data class with extension functions 46
5.2 Extension property for the TaskImportance enum 50
5.3 Example of calling of score extension property on enum 50
9.1 Asynchronous call to schedule generating method 78
9.2 Method for persisting generated schedules 79
9.3 DayAssignmentScoreCalculator class 79
9.4 Function for finding overlapping events 82

9.5 TirednessRule implementation as example of rule 82
9.6 Configuration for OptaPlanner planning engine 84

Chapter 1

Introduction

Knowledge workers[1] have multiple competing responsibilities, ranging from answering emails, attending meetings to performing highly demanding tasks. To be successful at work and get things done, workers have to continuously plan ahead or determine what's best to work on right now. When faced with multiple things to do — the complexity of choosing the task to work on right now increases. Planning is vital for completing projects and accomplishing longer tasks, since keeping high-level picture in mind while simultaneously concentrating on the solution is challenging. To make it possible, tools for planning are widely used. Knowledge workers plan appointments using calendar, track projects' progress using project management tools, manage tasks — personal as well as work-related — in a to-do list.

Our review of available research on personal productivity shows that there are factors which influence efficiency of performing cognitive tasks regardless of knowledge worker's base productivity level and other personal characteristics. For example, sleep restriction has been shown to have negative effect on cognitive performance [7], the same is true for high level of stress [24, 31]. According to some research people may feel more or less productive depending on the time of the day [6, 42, 45] which is supported by the research of different chronotypes [49].

> What factors influence knowledge worker's productivity and how can they be used as constraints for scheduling productive workday?

This question is examined in detail in chapter 3, introducing findings in the field of personal productivity constraints[2], and in chapter 4, in which the incorporation of the productivity factors into the schedule optimisation is discussed in detail.

Taking these considerations into account, we believe task scheduling is an important daily problem for most knowledge workers. We see the solution to be a tool which alleviates the planning effort and would help people in allocating their personal resources of time and energy to achieve effective task execution. The tool must be able to provide realistic schedule to the user, taking in account planned appointments, expected task duration, time boundaries of the working day, etc. The tool must do it automatically, minimizing the cost of usage, leaving more time for knowledge workers to concentrate on their tasks and not on the tool itself.

> We examine the possibility to create a tool for automated task scheduling based on available time, task characteristics and current productivity constraints.

[1]persons whose capital is his/her own knowledge

[2]We define the productivity factors (or constraints) as factors which have influence on productivity — increasing the productivity or, more often, decreasing it.

Many applications (for example, Todoist [4], Microsoft To-Do [1]) with functionality to create and manage the task lists in various forms exist, and some of them offer automatic selection of tasks for today and task scheduling. However, none of the reviewed approaches for automatic task scheduling has taken into account the current level of user's personal productivity.

The problem addressed in this thesis is schedule generation for workday taking into account current state of user productivity. The goal is to create a proof-of-concept application which would assist people in planning by suggesting possible schedules for a workday given list of tasks and information about productivity factors. Having tasks scheduled, users don't need to spend time and energy on task planning, instead concentrating on accomplishing tasks. Additionally, task schedule helps to reduce context switching overhead if task execution is interrupted.

We want to focus on maximizing expected effectiveness over longer time by helping the knowledge worker keep the stress level under control by suggesting schedules which correspond to the current productivity level. For example, severe sleep restriction (reducing sleep duration to less than 7 hours) has clear negative impact on the productivity level next day [7], thus, it would be better to concentrate on easier tasks, avoiding unnecessary stress and possible mistakes. On the other hand, being rested, feeling productive and having slept enough, it is desirable to concentrate first on the most difficult tasks.

To solve the problem described above and to answer the research questions we developed a tool which can generate suggested schedules for a day based on the timing and productivity constraints of the user. The DayO tool was developed in several stages:

1. research available literature on personal productivity;
2. develop the set of constraints and formulate the scheduling of tasks as a constraint-satisfaction problem;
3. design and create a web-based application for interaction with the user;
4. implement the rules and algorithms for schedule generation and test them;
5. deploy the application for pilot user evaluation.

Schedule is generated based on task characteristics, appointments scheduled for today and productivity factors — sleep quality and duration, subjective feeling of stress level and chronotype (expected higher productivity in the morning or in the afternoon). Information about fixed appointments is retrieved from user's Google Calendar account. The tasks are scheduled for the remaining time. User maintains a list of tasks which he can add, edit, complete or delete at any time. After opening the application in the morning, user is presented with daily form, where he enters values of daily productivity factors mentioned above. After the daily form is filled in, the system generates a workday schedule allocating tasks to the available time of the working day and presents the user with the choice of up to 3 suggested schedules. During the day user sees the unified view of schedule (with fixed appointments and scheduled tasks) and the task list, he then marks tasks as completed when necessary, or creates the new ones.

Once implementation of DayO was finished, we conducted pilot user evaluation to understand how accurate the suggested schedules were from the user's perspective and to gather feedback about the tool and the general concept of automatic schedule generation, as well as suggestions regarding possible improvements. Pilot user evaluation has been conducted with 4 participants during 1 week. They were requested to use the tool and provide daily feedback in questionnaires.

This thesis makes following contributions:

- it reviews the available literature about the factors influencing personal productivity;
- it introduces a tool which is capable of generating a schedule for a day from given tasks using time constraints to position tasks around existing events in a workday and productivity constraints with task characteristics to optimize the schedule regarding user's productivity and task priority;
- it presents the results of the evaluation of DayO application with users.

Chapter 2 discusses planning, optimization, and constraint-satisfaction problem (CSP). In chapter 3 we focus on strategies for workday planning, existing approaches and applications for automatic schedule generation, look into existing research on the factors of personal productivity. Chapter 4 provides an overview of the DayO application concept, formulates the task scheduling problem as a CSP and describes the rules for schedule generation and optimization. In chapter 5 the overview of the DayO application is provided including the architecture definition, description of server and client modules as well as data model. In chapter 6 the method and results of pilot user evaluation are described. Chapter 7 includes the discussion of the DayO concept and implementation as well as possible future work. The thesis is concluded by chapter 8 where the contributions are summarized.

Chapter 2

Background

In this chapter the planning and optimization problem is defined, as well as detailed definition of the constraint-satisfaction problem (or CSP), its applications and solution approaches is provided. The order of the definitions is chosen due to relations between the described phenomenons. Optimization problems can be seen as representing a small subset of planning problems, further optimization problems can be formulated and solved as sequence of CSPs which provides valuable methods for solving discrete optimization problems.

2.1 Planning Problem

Planning is a term that means different things to depending on the field. In robotics planning addresses the automation of mechanical systems that have sensing, actuation, and computation capabilities. In artificial intelligence, the term planning relates to discrete rather than continuous problem. “Instead of moving a piano through a continuous space, as in the robot motion planning problem, the task might be to solve a puzzle, such as the Rubik’s cube or a sliding-tile puzzle, or to achieve a task that is modelled discretely, such as building a stack of blocks.” [40, p. 3 - 9]

In this thesis we are more interested in the AI aspect of planning problem, thus the goal will be to find solution for a puzzle. More specifically, the planning problem for DayO application is to optimize goals (schedule) given limited resources (time) under number of constraints (productivity factors). The planning goals may be very different depending on the actual problem being solved: maximizing profit, minimizing energy consumption, maximizing productivity of the individual worker, etc. The resources may also be very different, starting with monetary resources (e.g. budget), to time and space; specific constraints related to this resources need to be taken into account when solving the planning problem (as working times, cost, amount of space or energy consumption). [1]

Thus, when we are talking about planning problem we mean an optimization problem which we need to solve to find the ’optimal’ plan (or solution). The mathematical optimization problem is defined in the next section.

[1] https://docs.optaplanner.org/7.0.0.Final/optaplanner-docs/html_single/index.html

2.2 Optimization Problem

"A *mathematical optimization problem*, or just *optimization problem*, has the form

$$minimize f_0(x)$$

$$subject to fi(x) \leq bi, i = 1, ..., m.$$

Here the vector $x = (x_1, ..., x_n)$ is the *optimization variable* of the problem, the function $f_0 : \mathbb{R}^n \rightarrow \mathbb{R}$ is the *objective function*, the functions $f_i : \mathbb{R}^n \rightarrow \mathbb{R}, i = 1, ..., m$, are the (inequality) *constraint functions*, and the constants $b_1, ..., b_m$ are the limits, or bounds, for the constraints.
A vector $x^\star$ is called optimal, or a solution of the problem, if it has the smallest objective value among all vectors that satisfy the constraints: for any z with $f_1(z) \leq b_1, ..., f_m(z) \leq b_m$, we have $f_0(z) \geq f_0(x)$." [10, p. 1]

The optimization problem is a problem of making the best possible choice of from a set of candidate choices. The variable x represents the choice made; the constraints $fi(x) \leq b_i$ represent requirements limiting the possible choices, $f_0(x)$ represents the cost of choosing x. ($-f_0(x)$ can be thought of as representing the utility of choosing x.) "A solution of the optimization problem corresponds to a choice that has minimum cost (or maximum utility), among all choices that meet the firm requirements." [10, p. 2]

Optimization problems have many real-world applications, such as portfolio optimization where variables are investments in the different assets and objective function may represent the risk of the portfolio with budget as constraint; another example is data fitting where the desired solution is a model which would describe the observed data and prior information and variables are given as parameters of the model with constraints implied by the desired qualities of the model or limits of parameters. Many practical problems from fields of engineering, investment, chemistry, operations, etc. can be formulated as optimization problems. Generally, the mathematical optimization can be applied to the different decision or design problems from variety of fields which can be formulated as a mathematical optimization problem (e.g. objective functions, variables, and constraints). [10, p. 2 - 3]

"Optimization problem can be expressed as a sequence of CSPs. By setting a threshold value on the objective function value, an 'objective' constraint can be added. Successive adjustments to the threshold value according to whether there are values of the variables that satisfy all constraints, allow the optimal value of the objective function value to be obtained." [12] In the next section we will define CSP and describe some of the algorithms of solving it.

2.3 Constraint-Satisfaction Problem (CSP)

Constraint satisfaction problems (CSPs) are mathematical problems defined as a set of objects whose state must satisfy a number of constraints or limitations.[2] More specifically, a finite domain CSP requires a set of variables, a finite set of possible values that can be assigned to the variables and the list of constraints which must be satisfied. [12] In the process of solving the problem a value from set of possible values is assigned to the variable in such a way that all, or at least "unbreakable" (also known as hard) constraints are satisfied, depending on the way the problem was modelled initially.

[2] `https://en.wikipedia.org/wiki/Constraint_satisfaction_problem`

2.3.1 Formal Definition

CSP consists of set of variables $X = x_1...x_n$, D_i — a finite set of possible values (domain of the variable), set of *constraints* that restrict the values that the variables can take simultaneously. Values must not be consecutive integers, and may not be numeric at all, but it is important that the values are discrete, since continuous values require different definitions and solution approaches.

Constraints. "The constraints of a CSP are usually represented by an expression involving the affected variables, e.g. $x_1 \neq x_2, 2x_1 = 10x_2 + x3$ and $x_1x_2 < x3$. Formally, a constraint $C_{ijk}...$ between the variables x_i, x_j, x_k is any subset of the possible combinations of values of x_i, x_j, x_k, i.e. $C_{ijk}... \subseteq D_i \times D_j \times D_k \times$....The subset specifies the combinations of values which the constraint allows.

For example, if variable x has the domain $\{1, 2, 3\}$ and variable y has the domain $\{1, 2\}$ then any subset of $\{(1, 1), (1, 2), (2, 1), (2, 2), (3, 1), (3, 2)\}$ is a valid constraint between x and y. The constraint $x = y$ is equivalent to the subset $\{(1, 1), (2, 2)\}$." [12]

The constraints of the real-world problems are usually represented differently in practice often being more complex than could be expressed by linear inequalities or equations. A constraint can affect any number of variables from 1 to n, where n is the number of variables in the problem. The number of affected variables is the *arity* of the constraint [12].

2.3.2 Example: N-Queens

In the n-queens problem, you want to place n queens on an $n \times n$ chessboard (square grid). Each queen occupies one square on a grid and no two queens share the same square. Two queens are attacking each other if one of them can travel horizontally, vertically, or diagonally and hit the square the other queen is on. The problem is to place the queens such that no two queens are attacking each other. Formally the problem could be defined as follows.

Variables: Let $Q = \{q_1, ..., q_n\}$ denote the queens to be placed. Further $X = \{x_1...x_n\}$ denoting the horizontal position, $Y = \{y_1...y_n\}$ — the vertical position on the chessboard.
Domain: If the chessboard is defined as $[1..n] \times [1...n]$ matrix with n columns and n rows, then the set of possible possible values for each of the q_i will be $D = \{(1, 1), (1, 2), ..., (n, n)\}$
Constraints:

- Use a chessboard of n columns and n rows meaning that the assignment is valid, if $x \in [1..n]$ and $y \in [1..n]$
- Place n queens on the chessboard: for each q of $\{q_1...q_n\}$ $\nexists q_i$ with $(x, y) \notin [1..n] \times [1...n]$
- No two queens can attack each other (no two queens are on the same row, column, or diagonal): for each q of $\{q_1...q_n\}$, let q_i denote the queen assigned to (x_i, y_i), then $\nexists q_j$ with $(x_j, y_j) \wedge (x_i = x_j \vee y_i = y_j \vee x_i = y_i \wedge x_j = y_j)$; further let k denote some constant then $\nexists q_j$ with $(x_j, y_j) \wedge x_j = x_i + k \wedge y_j = y_i + k$

The N-Queen problem can be solved by utilizing one of the common pattern in CP solvers is an alternation between *propagation* and *backtracking*.
"**Propagation** consists of taking some constraint — which might be a constraint of the original problem, or a constraint learned or hypothesized along the way — and applying it to variables.

Backtracking occurs when the solver, having made a wrong guess, reverts back to an earlier state and tries something else." [3]

2.3.3 Solving CSP

There are different types of solutions to the CSP problem which include possible, feasible, and optimal solutions.[4] When possible solution is just any assignment to the variables of values which belong to the value domain and may break any number of constraints, a feasible solution is an assignment of a value from its domain to every variable, in the way that every constraint is satisfied. If problem has a feasible solution it is called "satisfiable", otherwise the problem is "unsatisfiable".

In process of solving a CSP we might want to find just one solution without preferences to the particular qualities of the solution, all feasible solutions, an optimal (or at least a good as optimal) solution which requires the definition of the objective function in terms of some or all of the variables [12].

Algorithms for solving CSPs are usually aimed at simply finding a feasible solution, but they can also be adapted to finding an optimal solution. "For instance, an objective variable can be created to represent the objective function, an initial solution is found, and then a new (objective) constraint is introduced specifying that the value of the objective variable must be better than in the initial solution. This is done repeatedly, tightening the constraint on the objective variable as each solution is found, until the problem becomes unsatisfiable: the last solution found is then an optimal solution. The number of iterations, and therefore the computation time, depends on the quality of the initial solution. A common practice is to apply a heuristic method for generating an initial solution." [12] "Constraint satisfaction problems on finite domains are typically solved using a form of search. The most used techniques are variants of backtracking, constraint propagation, and local search."[2]

Backtracking is a recursive algorithm. The state of the search (partial assignment of the variables) is maintained at all times and the consistency checked after each new assignment was made the new assignment is checked against the partial solution; if any constraint is broken, then assignment is discarded and new value from the domain is assigned. If the current variable cannot be assigned to any value from the domain in such a way that no constraint is broken, then algorithm backtracks to the previous variable and searches for new assignment, repeating the process described above. After either all possibilities have been considered, or the solution has been found, the algorithm finishes execution, in first case no solution was found. This algorithm is not used in practice because of its inefficiency in many cases.[2] [12]

Forward checking is a lookahead algorithm that checks the constraints not only between the current and past variables, but also considers the future variables. When a value is assigned to the current variable, any value in the domain of a future variable which conflicts with this assignment is (temporarily) removed from the domain. Therefore, if the domain of a future variable becomes empty, we can be certain that the current partial solution is inconsistent, and either another value for the current variable is chosen, or the algorithm backtracks to the previous variable. Forward checking allows detecting the branches of the search tree that will lead to failure earlier than backtracking [12].

MAC algorithm does even more forward-checking, considering the future variables against each other after current assignment is made. Thus, both the values which are not consistent with

[3] `https://developers.google.com/optimization/puzzles/queens`

[4] `https://docs.optaplanner.org/7.0.0.Final/optaplanner-docs/html_single/index.html`

the current assignment, and those which are not supported in domain of the future variable are deleted from the possible values. This makes the domains of the future variables event smaller, saving the computation time [12]. There are ways to improve the efficiency of the above mentioned algorithms by using techniques as constraint propagation or improve the quality of the solution (if searching for optimal solution) by using local search method.

Constraint propagation techniques are used to modify a constraint satisfaction problem, enforcing a form of local consistency which are conditions related to the consistency of a group of variables or constraints. This technique turns a CSP into an equivalent but is usually simpler to solve problem, and provides a tool for proving satisfiability or unsatisfiability of problems. "The most known and used forms of local consistency are arc consistency, hyper-arc consistency, and path consistency."[2] Local search methods work by iteratively improving a complete assignment over the variables. At each step, some values assignments are changed to increase the number of constraints satisfied by this assignment. It is possible to integrate search algorithm with local search creating a hybrid algorithm which could find more optimal solutions by considering more different assignments.[2 4]

2.3.4 Applications

Location Problem involves location of facilities, such as warehouses, to supply demand to customers. A set $\{1, ..., m\}$ mg of potential facility locations is given, and a fixed cost f_i is incurred if a facility is established at location $i(i = 1, ..., m)$. There is a set of customers $\{1, ..., n\}$, and the cost of supplying customer $j(j = 1, ..., n)$ from a facility at location i is c_{ij}. The location problem is to choose a subset of the potential locations at which to establish facilities, and then to assign each customer to one of these facilities, so that the total cost is minimized [12].

Scheduling Problem has many examples in production industries that require jobs to be scheduled on machines. One of the most general and challenging is the job shop scheduling problem in which n jobs are to be scheduled on m machines. Each job $j(j = 1, ..., n)$ comprises a set Oj of operations that must be executed in a specified order. The machine on which any operation o must be performed is m_o and the corresponding processing time is p_o. Since machine capacity constraints prevent any machine from processing more than one operation at the same time, a sequence of operations must be specified for each machine. A common objective is to find a schedule that minimizes the makespan, which is the completion time of the last job. The job shop problem is very difficult to solve. Local search heuristics are successful in generating near to optimal solutions at moderate computational expense [12].

Other CSP problems are: car sequencing problem in car production industry (planning for the sequence in which cars need to be processed to install all necessary options which may be different for each model); vehicle routing for supplying the customers with different requirements from one depot (the route for the vehicle needs to be planned such that the it starts and ends at the depot, visiting each of the customers on the route); timetabling problem (creating a timetable for example for lectures or exams having restrictions of the earliest starting time, latest ending time, duration, rooms, professors' preferences, etc.) There are many other examples of constraint-satisfaction problems in the industry as well as in research and education.[4]

Chapter 3

Related Work

In this chapter we review existing research about the factors of personal productivity and investigate concepts and currently available applications for automated task planning. The goals of this chapter are: to understand how the concept of DayO compares to the other concepts and applications, and to review the existing scientific literature about personal productivity factors. Based on the review of the available research the productivity constraints are defined for modelling the constraint-satisfaction problem.

3.1 Planning Productive Day

Planning is defined as "process of thinking about and organizing the activities required to achieve a desired goal."[1] Another term tightly bound to planning is time management defined as "the process of planning and exercising conscious control over the amount of time spent on specific activities - especially to increase effectiveness, efficiency or productivity."[2] Both definitions have found application in the technique of task lists (so called to-do lists) and applications for task tracking and management. In this section we seek understanding of the strategies used for day planning and principals influencing the choice of tasks for a day.

Planning could be performed either explicitly or implicitly, according to the research by Classens [16] in which 18% of respondents answered that they did not plan the tasks they were going to accomplish during the day, whereas other respondents planned tasks occasionally (12%), a week ahead (29%), or even daily before starting to work (41%). However, the respondents who reported that they did not plan their day might not be aware that they employed some strategy for choosing tasks to be performed right away.

Regardless of the planning strategy the question of task choice remains open, since usually there are more tasks than it is possible to complete during the day. According to the diary study by Classens [16], the perceived importance and urgency of the task were predictive of the task priority, which means that more urgent and important tasks would receive higher priority. However, the diary experiment has shown that urgent tasks with low importance were completed earlier than other tasks. Generally, only 73% of all the tasks planned for the day were actually accomplished; the reasons for not accomplishing the planned tasks were interruptions, unplanned tasks, and lack of time. Further, the "unplanned tasks were completed to a higher extend than planned tasks and were perceived as more urgent and less attractive than planned tasks", which

[1] https://en.wikipedia.org/wiki/Planning

[2] https://en.wikipedia.org/wiki/Time_management

was attributed to the recency effect (if task arrived more recently, it was better remembered and, therefore, perceived as more urgent) [16].

Another aspect of planning the productive workday is the order in which tasks are executed during the day. In the interview study by Classens 59% of respondents reported to start their working day from large and difficult tasks, motivating this choice with being more productive at the start of the day; 18% of respondents chose to start from small, enjoyable tasks; 18% of participants reported to start their day from tasks which were important for others, and 24% of respondents reported to not have any structured execution behaviour. Author has named these differences "preference for structured versus non-structured way of working"; as was described above, the planning strategies might range from planning as far ahead as one week to not planning at all, preferring to adapt to the current work situation [16].

One of the widely used and acknowledged approaches for planning is the GTD approach, which is described in the following subsection.

Getting Things Done Approach

The Getting Things Done (GTD) approach which was first described by the author David Allen in his book [5], has started new way of thinking about tasks and planning, giving the foundation for many productivity applications as we know them today, especially the programs for management of tasks (to-do's). In the article by Heylighen and Vidal [33] authors emphasize that in the professional life people are confronted with always changing plans and new information, therefore, the use of GTD system would be justified to provide enough flexibility to adapt to the constraints and affordances of individual situation by choosing the appropriate task. Moreover, the idea of organising work proceeding bottom-up, as opposed to the traditional top-down approach, would also free time by reducing the planning overhead and provide additional flexibility.

One of the main ideas of the GTD is the suggested work-flow, the first stage of which, named "capture" or "collect", consists of gathering all tasks (ideas) into the in-box, thus externalizing the tasks from one's mind. Following stages include "clarifying" and "organizing" the items in the in-box, by first filtering out the tasks which are not actionable and storing them in proper way; other tasks can either be done immediately, if they take very little time to accomplish, delegated, or assigned to the waiting list, additionally given a specific context (time, place, other dependencies, etc.) The concluding stages of the work-flow suggested by D. Allen are: "plan" and "do" (or "reflect" and "engage" in the newer edition).[3]

In this thesis we would rely on people having only actionable items which are related to work context in their to-do lists, which means that we can assign any task from the list to any time during the work day and would only need to consider the inherent properties of the task (importance, urgency, deadline, etc.) to influence the individual task assignment. The rules according to which tasks will be assigned in the dayO application are described in detail in the Concept section 4.1. Based on those inherent properties of the tasks and other constraints the application will suggest the plan for the day, thus automating the "plan" stage of the work-flow, so that user can continue to doing the planned tasks ("engage" stage).

[3]https://en.wikipedia.org/wiki/Getting_Things_Done

3.2 Automatic Task Scheduling

Automated task planning (or scheduling) is not something entirely novel. There is research about different systems and approaches for automatic scheduling of tasks in industry [32, 44]. Though, since knowledge work is different, this systems and concepts may not be applicable to planning the workday of the knowledge worker.

3.2.1 Task Scheduling Concepts and Approaches

The concepts and approaches described in this section were found as patents, thus, what we discuss in this section are concepts and methods and not a concrete implementation. Applications providing similar functionality are discussed in section 3.2.2. The approaches for automated task planning which our research has revealed could be categorised into two different groups:

1. project planning and management [13, 17, 47]
2. scheduling tasks of an individual worker [26, 41, 52]

The difference between two groups is that approaches for project planning and management are defined for much higher level of abstraction (tasks are much larger and normally are assigned to the whole team rather than to a single person); the approaches from the second group can be used for planning the tasks for a workday of an individual worker.

Project Planning and Management Approaches. In the category of the project management concepts and approaches we find the methodology describing the generic task management strategy for managing multiple tasks distributed among plurality of personnel [47], the concept of the computerised system which could automatically track the real-time progress of the project and reduce the risks from improper estimation of the task duration by individual employees [17], or explain the rationale behind the planning decision provided by the system by storing the consideration in accordance with which the task was originally scheduled [13].

Scheduling Tasks of an Individual Worker

The concepts for automatic planning which could be used for task scheduling for individual worker described in found patents include the "method and apparatus for arranging and displaying task schedule in a calendar view format" [26], "method for automatically developing suggested optimal work schedules from unsorted group and individual task lists" [52], and "intelligent planning and calendaring system with cueing feature and floating tasks" [41].

Method for arranging and displaying task schedule in a calendar view format. This method relates generally to software applications that schedule and display a number of tasks defined by the user. Given tasks with the relevant information about each individual task (start and end, necessary resources, dependencies on other tasks, etc.), the system should be able to quickly position the tasks in the calendar representation, putting each of the tasks into the day to which it was scheduled and giving each task a "movable task-bar" as visual representation. Task bar can span more then one day, if task duration is defined for multiple days. The inventors argue that such representation of the tasks could be more informative than other representations such as Gannt chart or a table. [26] In DayO a calendar representation for each of the assigned tasks is created. However, compared to the system described above, which only positions task bar into a day without start or end time, our application will generate task bars positioned in a day at their respective starting time with height of the bar representing the duration of the task as it is usual in the calendar representations.

Methods for automatically generating schedules. From the found concepts in group of task scheduling for an individual there are two methods which describe systems for automatic generation of workday schedule:

1. method for automatically developing suggested optimal work schedules from unsorted group and individual task lists [52]
2. intelligent planning and calendaring system with cueing feature and floating tasks [41]

Both of these methods describe analogous systems which could generate a schedule for a day given set of unordered tasks with the earliest starting time, latest ending time and task duration provided in the task description. Tasks are then ordered by the system depending on the time which is left to complete the task (which is usually equal to deadline of the task minus this task duration). Finally tasks are allocated in the specified order to the "bins" of free time defined for the user. (2) also takes a reward value for completing the task as one of the parameters, (1) differentiates between "fixed" and "floating" tasks, where fixed tasks are the ones for which the difference between the earliest start and the latest finish is equal to the task duration.

The concepts of the systems described above are very similar to the DayO concept, but there are some differences:

- due date (or latest ending time) is an optional parameter of the task and there is no earliest starting time specified, since all tasks are considered to be active and independent;
- since tasks have different relative importance and complexity, these parameters also influence the scheduling decision;
- generated schedule is also influenced by the productivity constraints of the user on each individual day (more about the productivity factors could be read in the following section 3.3 and the chosen productivity constraints are described in the 4.1).

Thus, our application uses a different approach to scheduling tasks which is accounting not only for the task priority, but also for the user's productivity.

3.2.2 Existing Applications with Automatic Task Scheduling

According to our research of existing applications in the field of productivity tools, there are two applications which provide automatic scheduling of the tasks to particular time: Purp [3], Microsoft To-Do [1], and Todoist [4]. However, the implementations, exact approaches and algorithms are not publicly available, thus, our understanding of these applications' functionality is based on information provided on their public websites.

Purp. This application offers the "brain" feature [3] which can suggest the time when to do the tasks based on the task duration and the "goal" to which it belongs. However, very little information is available about the "brain" feature except that it "calculates which task fits the given time of the day and which fits the actual moment" and "provides the list of today, the tomorrow and the list of uncompleted, missed tasks" [3], no information is provided on how exactly this feature makes the planning decisions and which parts of the task description are accounted for.

Microsoft To-Do. This application is providing the "Intelligent Suggestions" feature which allows user to start the day ("My Day" list) without having any tasks in it to reduce the feeling of being overwhelmed by the tasks and then quickly add all the tasks which need to be done today. "Intelligent Suggestions" feature prioritizes the tasks for user, showing the ones with the highest priority. It is, however, unclear how the algorithm works and based on which aspects of the task the decision about task priority is made. DayO does not only choose the tasks which need to be done today, but also suggests a schedule for a day.

Todoist. This application introduces "smart schedule" feature which "uses predictive modelling to help you easily plan out your tasks for the day and week to come. It learns your personal productive habits, and takes into account patterns across all Todoist users, to predict the best possible due dates for your tasks." [4] According to the given description of the smart schedule, it can infer the habits of the users which would help planning specific tasks to some times when user commonly accomplishes such kind of tasks; there is also some mechanism for determining the urgency of the task and generating different schedules for the week days and weekends. However, we were not able to find research which would give the scientific background to the mentioned scheduling algorithm or provide information about the user evaluation of this feature.

Existing applications offer features for automatic tasks planning which are taking into account the task parameters (such as priority or deadline), user-defined goals (in case of Purp), or inferred habits of the user (in case of Todoist). To our best knowledge neither of them explicitly includes the information about the user's productivity on a given day into their planning decision or accounts for existing events when generating a suggestion. Thus, additionally to choosing which tasks to complete today based on task parameters and productivity constraints, DayO provides generated schedule(s) in which chosen tasks are positioned in the way corresponding to user's daytime productivity preference and accounts for events already planned during this day. The factors influencing the schedule optimization are discussed in detail in 4.2.

To our best knowledge there is no research of the equivalent approach, application or concept which would describe system automatically generating day schedule optimized using both time and productivity constraints (choosing the tasks which need to be accomplished today, fitting them around existing event, optimizing schedule regarding user's productivity constraints), and include the evaluation of the system with users, therefore the DayO concept might be considered novel in the field of productivity research and automatic day planning.

3.3 Factors Influencing Personal Productivity

The commonly used definition of productivity is the one from the field of economic studies, where productivity is defined as something which "describes various measures of the efficiency of production." [4], one of the measures of economic productivity is "a ratio of the value of the product produced to the cost of the resources used to produce it." [14] More generally we could define the productivity measure as a "...ratio of outputs produced to resources consumed." [14]

Further in same article David Card discusses that productivity can be measured very differently for different organizations, starting from differences in the definition of the output (which might be measured as functionality or as a product) and the resources (measured in cost or effort), further there are factors which may influence the interpretation of the productivity measures (such as change in the requirements, or quality at delivery), coming to the conclusion that one must not only carefully select the measures of the product size and input (or resources), but also consider influence of the other factors; thus, "No single measure of productivity is likely to be able to serve all the different needs of a complex software organization..." [14]

Similarly, the personal productivity of the knowledge worker could be defined as ratio of the output (solved tasks) divided by invested resources (time). Due to the nature of tasks which knowledge workers are solving, it is difficult to devise universal efficiency measures. Individual's productivity might also depend on personal qualities, such as for example stress resilience, creativity, etc. besides knowledge and experience which worker has. Thus, the baseline of the productivity may vary greatly when comparing different individuals. In this section I will describe the factors which influence one's productivity independently of differences between individuals. Which means that, if individual would have a level of productivity which can be described as X, then after experiencing the influence of the factor F the original productivity will be either increased or decreased (depending on the nature of the factor) by some δ. Mainly we are interested in the sign of the effect and not in measurement of the magnitude of the impact.

There are many factors which may influence worker's productivity. In the article about impact of office environments on employee performance [53] author names following factors affecting productivity: job stress and dissatisfaction, motivation, and factors of the indoor environment (such as light, sound, air quality, etc.) Having researched the topic of the factors of productivity to gain better understanding of its possible constraints, I have read about the following factors: general level of stress (which also may be described as psychological well-being) [20, 24, 31, 65], health risks [9], personal time-management strategies [16], quality and duration of sleep [6, 7, 54, 60], chronotype of the person [6, 34, 46], email strategy [11, 39, 43], interruptions [21, 22, 28, 36, 64], working environment [18, 23, 53], information overload [15, 25, 38, 62], and happiness [66]. This list might not be exhaustive and some of the factors are certainly interconnected, nevertheless, it would be enough to understand those factors to be able to design the application which would plan the day depending on the productivity constraints for individual days. The following factors will not be considered and discussed in detail, since they are either relatively stable when considering one worker and his/her workplace or cannot be controlled or changed by individual workers: working environment, personal time-management strategies, and health risks.

We describe the following factors in detail: sleep duration and quality in section 3.3.1, mornigness/eveningness in section 3.3.2 and stress in section 3.3.3, since these are the factors which will be considered in the dayO application. Other factors, such as information overload, email strategy, interruptions, etc. will be briefly described in the subsection 3.3.4 to provide a better understanding of the productivity factors.

[4] `https://en.wikipedia.org/wiki/Productivity`

3.3.1 Sleep Duration and Quality

The research shows that quality of sleep has economical [8, 54], health-related [59], behavioural and physiological implications [6, 7] and also might affect academic success [27, 63].

The cost of poor sleep is very high as it was clearly shown in the research by Rosekind et al [54] analysing more than 4000 participants employed at different companies with various job descriptions. At-work productivity loss cost related to poor sleep was estimated at approximately 3000$ per employee yearly for insomnia group and at 1200$ for good-sleep group. The research has also shown that the average productivity loss was between 2.5 and 6 percent for good-sleep and insomnia group respectively. Sleep loss and tiredness was reported to have the largest impact on time management and work with mental demands with approximately 10% of time with limited performance for the good-sleep group. The economical model described in the study by Biddle and Hamermesh shown that sleep affects the wages through its impact on productivity. In the same study authors have also shown that there is a significant partial correlation between the sleep time and the work time [8].

Meta-research on behavioural and physiological effects of sleep restriction shows that restricting the duration of sleep below the individual's optimal time in bed causes a range of neurobehavioral deficits, including lapses of attention, slowed working memory, reduced cognitive throughput, depressed mood, and perseveration of thought. It was also shown that these effects accumulate and after some time can reach the level of dysfunction comparable with the complete sleep deprivation. Even though the degree to which the negative effects on the neurobehavioral aspects were varying depending on individuals, the study claims that restriction of sleep to less than 7 hours a night for healthy adults caused adverse effects on range of cognitive tasks and daily behaviours such as driving [7] .

In the article by Gaultney [27] the weak correlation between the quality of sleep habits that students have reported and their academic success was revealed. The study has also shown that significant number of students who were at risk of sleep disorders were also having academic problems and that the individuals at risk of at least one sleeps disorder were overrepresented in the group of students with the GPA below 2.

From the economical research [8, 54] we can see that influence of sleep quality and duration on personal productivity at work is tangible and can be quantified. Sleep can be also seen as part of economical model [8]. For this thesis it will imply that there is a reason to believe that people who sleep enough will be more productive than those who did not sleep enough or had disturbed sleep. Therefore, the sleep length and quality of sleep will be two of the constraints influencing optimal suggestion for the day plan. It was also shown that sleep may impact the academic success [27] and significantly impair the cognitive abilities [7] and generally be linked to the lower performance of the tasks requiring mental strain and increased attention. [6]

For this thesis we will assume that productivity of knowledge workers would decrease in case of sleep restriction below 7 hours a night or insufficient sleep quality. As study [7] also suggests, there are the cumulative effects of sleep restriction which theoretically could be calculated and used as a factor increasing the productivity loss. It would be interesting to calculate the exact sleep debt and then design the "pay-back" model, but this would require additional research into sleep debt and productivity loss which for now would be left out of this project for the lack of time to research the question in more detail.

Sleep habits and individual differences in sleep behaviour are related to the topic of circadian rhythms, also known as morningness-eveningness. The influence of the chronotype on the personal productivity will be discussed in the next section.

3.3.2 Morningness and Eveningness as Factors of Personal Productivity

It feels very natural that people may be feeling more or less productive during particular times of the day. Widely known terms of "lark" and "owl" referring to people feeling more productive in the morning or later during the day respectively [6] are generally referred to as chronotype in sleep research. Morningness/eveningness — other definition often used for the chronotype — could be considered a stable characteristic, since it was shown in the study by S.-J. Pain [49] that the chronotype was largely independent of ethnicity, gender or social situation. The possible connection between morningness/eveningness and the daily schedules was reported, but it is not clear whether it is a correlation or a causation.

Further we know from the paper about mornigness/eveningness and the need for sleep by Thilliard et al. [60] that evening types would ideally need more sleep and that they were able to sleep only 75 % of what was perceived necessary. Whereas morning types were associated with lower sleepiness during the day, though no strong prove was provided towards it. Study by Horne and Olsberg in which the oral temperature was measured to determine the peak times for the different chronotypes [34] has shown that the temperature rises significantly faster for the morning types than for the evening types, the former having the peak time one hour earlier than the later, though no connection was made to the productivity.

The morningness-eveningness questionnaire [35] designed by Horne and Ölsberg for self-assessment of the chronotype includes the questions about the perceived productivity during the day, but there was no research to provide strong evidence for connection of the two phenomenons. Recent study with more than 31000 participants based on measurement of the sleep duration and productivity while performing search tasks by Althoff et al. [6] claims that there is a connection between the chronotype and performance during particular times of the day, but the suggestion is that "early types tend to be higher performing earlier in the day while late types are higher performing later" [6] which is unfortunately not very precise.

Analysis of the knowledge workers' activity rhythms based on the observation of the patterns of use of the productivity software in the in situ study by Mark et al. [42] provides information about the daytime bound productivity, showing that workers were most productive in the mid-afternoon and at around 11am. No connection to the chronotype of the individuals was made. Authors claimed that participants were considering themselves most productive when working on challenging and engaging tasks (focused work), on the other hand participants reported to be happiest when doing rote work which corresponds to high engagement and low challenge as compared to the focused work [42]. These findings lead to the assumption that the most challenging tasks should be planned when the person perceives himself or herself the most productive, distributing tasks according to the individual productivity time, if possible.

Similarly to [42] the article on perception of productivity by software developers by Meyer et al. [45] also shows that there could be common trends of the daytime based productivity. The results of this study based on self-reported productivity were allowing to group developers into three different groups: morning, afternoon and low-at-lunch. Even though the periods of productivity are distributed differently then it was described in [42], the study allows to assume that there are times of the day when people consider themselves more productive then at other times. Surely the results are based on the perceived productivity and self-reports, but since there are very few methods which allow precise and stable assessment of the productivity, personal judgement could be a valid option.

3.3.3 Stress Level and Productivity

Persons can experience stress due to different factors perceived by the individual to be adverse or threatening. Stress factors can originate from situations related to any aspect of life, such as family, relationships or daily routines, but possibly the most deeply analysed source of stress is work-related stress. "Workplace stress can be defined as the change in one's physical or mental state in response to workplaces that pose an appraised challenge or threat to that employee." [20] For this thesis it is important to understand how stress, including workplace stress, influences the personal productivity.

Further Colligan analyses the factors which can contribute to the experiencing of stress at work, including organizational climate, role conflict, relationships, but not limited to it. Authors provide thorough research of the stress, concluding that besides significant negative implications for physical and mental health, workplace stress leads to reduced productivity, decreased morale and absenteeism. [20]

The study conducted using the ASSET dataset by Johnson et al. [37] analysing the difference in experiencing stress of across 26 occupations in UK points out that "...up to five million people feel "very" or "extremely" stressed by their work and work-related stress costs society about £3.7 billion every year." The three stress-related variables (psychological well-being, physical health and job satisfaction) were analysed to reveal the differences between occupations in the stress level.

"Highly stressed employees are subject to greater health risks, increased cost, and productivity losses than those with normal stress levels." [65] In this study researchers tested two different programs designed for reducing the amount of stress at the workplace and shown that compared to the control group both mindfulness programs have helped to significantly reduce perceived stress and improve the quality of sleep, the positive changes in mood and the workplace productivity assessed with WQL index were observed, but were not shown to be statistically significant.

A large study of workplace environment, productivity and stress revealed that the strongest predictor of productivity was the perceived psychological well-being of the worker. Researchers did not find a significant predictive power for any of the analysed work stressors individually which was attributed to the different coping capabilities of the individuals. Psychological well-being was explained to be a measure of general stress level and, given that "Burnout and emotional exhaustion are conceptualized as extreme forms of stress..." [24], decrease in the psychological well-being may be understood as one of the stages below the extreme stress. [24]

"The effect of stress is seen in reduction of available attentional capacity (psychological adaptability) and an increase in physiological strain that has to be compensated for by regulatory systems (physiological adaptability)." [31] Further in this article Hancock and Warm introduce the dynamic model of stress and show the effects it has on sustained attention. Authors underline that even though all individuals might be different in their capability to adapt to stress, when stress is at the extreme level, "...common behaviour across individuals may be expected.. Arising from the understanding of dynamic stress model and of the solution paths leading to resolving the task, a very interesting idea is mentioned that the less demand for attention a task shows, the smaller will be the impact of stress on the performance of this task, researchers called this type of task processing "automatic.. [31]

Assessing the Level of Stress

There are many different aspects of stress, as there are many ways for the assessment of the current stress level. Some indices concentrate on assessing the stress level by aggregating the stressful situations in scope of the entire life, others concentrate on the current perception of stress. The indices for measuring the level of stress include Effort-Reward-Imbalance scale or ERI [55, 56] and The Standard Stress Scale [30]. In the Perceived Stress Scale questionnaire by Cohen [19] consisting of 10 different questions all of which start with "In last the month, how often..." [19]. All of the methods described above tackle the question of more global stress level, as perceived by the person over the longer period of time. It is common for the stress assessment questionnaires to use the 5-point Likert scale for denoting the severity of the condition or the level of experienced stress.

However, for the application of this thesis it would be enough to understand the current level of stress person is experiencing when the suggestion for the day is being designed, which certainly would implicitly include the general feeling of stress. The user, therefore, will be asked daily to enter the perceived level of stress into the application using 5-point system to denote the current level of stress, before the optimization of the schedule for the day has started. Only very high levels of stress will influence the suggested plan for the day, having greater impact on more difficult tasks as compared to the easier tasks.

3.3.4 Other Productivity Factors

There are many factors which may influence the productivity, but we will concentrate on the few factors which could be at least to some extend changed or controlled by the individual. In the subsections below we will describe the following productivity factors: information and technology overload, interruptions, and email. These factors could be perceived as closely related, since the overload may be caused by many interruptions or increase in incoming information (for example through means of email). Generally, those factors can be seen as characteristics of the modern working place.

Information Overload and Productive Work

Having information is crucial for making informed decisions and almost all of knowledge workers' daily tasks are connected to information - finding, gathering, analysing, classifying and so on. But is there such a thing as too much information? Can more information cause the loss of productivity? These are the questions which many researchers have asked since 1980s in the organizational, IT and marketing research. According to this research "performance (i.e., the quality of decisions or reasoning in general) of an individual correlates positively with the amount of information he or she receives — up to a certain point. If further information is provided beyond this point, the performance of the individual will rapidly decline" [25]. The phenomenon of excessive information income can be described as information overload.

Based on their research and analysis of literature on information overload Eppler and Mengis explain that information overload can happen in many different situations, such as for example decision making, information retrieval and analysis, or even communication, leading to inefficient work, loss of control, demotivation, ignoring of information, and greater tolerance of error. There causes of this phenomenon include the inherently information-related factors (such as ambiguity, uncertainty, diversity, etc.), task complexity, innovations, time pressure, as well as personal limitations like experience and cognitive abilities. Information technology contributes to the information overload with push-notifications, speed of access, low duplication cost, etc. [25]

Studies by Karr-Wisniewski et al. introduced the term of technology overload (or crowding), of which information overload is one aspect, alongside with feature and communication overload. [15, 38] In the article about the technology overload Karr explains that the different aspects of technology overload are grounded in the human limitations. According to the model of bounded rationality individuals presented with more information than they have either time or ability to process, will experience decrease in the decision-making performance being at the same time sure that more information leads to better decisions. [38] Thus, the effectiveness of the decision-making (e.g. productivity) of the knowledge worker might decrease when information overload occurs. Since the work of the modern knowledge worker is highly dependent on different information systems, technology overload or crowding is also an important aspect to consider.

According to the study done by Karr-Wisniewski et al. [15] 47% of the respondents experienced feature overload, 55% - information overload and 86% communication overload. Also specifically perceived communication overload was found to be negatively correlated with the perceived productivity, though other aspects also had negative correlation with productivity, especially, if technology dependence was high [38] . The communication overload was explained by the human interruption theory and we will discuss the interruptions' influence on the productivity in detail in the next subsection.

Interruptions and Productive work

Interruption is defined in the dictionary [5] as *"an occasion when someone or something stops something from happening for a short period"*. More specific to the context of knowledge worker's task *"an interruption is "externally generated, randomly occurring, discrete event that breaks continuity of cognitive focus on a primary task" (Corragio, 1990, p. 19) and typically "requires immediate attention" and "insists on action""* [57].

Different types of interruption are defined, including the active (also known as self-interruption) and passive when the source of the interruption is either some external system (email) or another worker. In the article on interruptions' cost following types of interruptions are defined: total — which completely occupy the attention (meeting, active phone conversation), dominant — activities which largely occupy the mind, but let the thought about original task develop in the background (walking or recreational browsing in internet), distractions — activities which drain the attention from the main task, reducing the accuracy or causing the loss of speed (using instant messaging applications), and finally the background activities which only divert the portion of attention (like listening to music) [58].

The annual cost of the interruptions to the knowledge workers' work was estimated at $588 billion in the article by Spira and Goldes published in 2005. The working time consumed by the interruptions was claimed to be 28% of the total working time. [58] Earlier study of teams of software engineers by Solingen et al. estimated the time spent on interruptions at 15-20% of the total working time, where single interruption would take between 15 and 20 minutes on average, if accounted for the interruption's occurrence, time for handling on it and recovering back to the interrupted task [64].

According to Perlow's observational research of the team of software engineers in late 90s *"large proportion of the time spent uninterrupted on individual activities was spent in very short blocks of time,*

[5] http://dictionary.cambridge.org/dictionary/english/interruption

sandwiched between interactive activities. Seventy-five percent of the blocks of time spent uninterrupted on individual activities were one hour or less in length, and, of those blocks of time, 60 percent were a half an hour or less in length." [50] In more recent research it was found out that the activity was interrupted on average every 3 minutes [29, 38, 45], but this might also include the self-interruption. According to Dabbish et al. *"Self-interruptions account for a significant portion of task switching in information-centric work contexts."* People are mostly interrupting themselves to come back to the tasks belonging to the main working sphere for which workers are accountable, which may imply that self-interruption could have a positive effect on the productivity [22].

It was shown that task interruptions influence the individual decision making performance (the quality and the time to make a decision). According to the findings of this study, *"interruptions facilitate performance on simple tasks while hindering performance on complex tasks"* , while the frequency of the interruptions was associated with decrease the decision accuracy and increase in time required to form a decision [57]. Controversially, the experimental study by Monk found that more frequent interruptions were associated with the faster task resumption than less frequent interruptions, explaining it with more aggressive goal maintenance when interruptions occurred more frequently [48]. Though, both the primary task chosen for the experiment and the interruption task did not require special knowledge or expertise and could be defined as simple, in which case the findings of both papers could be seen as similar. A diary study of the daily work of 11 knowledge workers has also shown that *"task complexity, task duration, length of absence, number of interruptions, and task type influence the perceived difficulty of switching back to tasks"*, suggesting that information technology could help workers recover from task interruptions [21].

Thus, interruptions to the original tasks of the knowledge workers play significant role in daily work routines and can affect the productivity not only by taking away the time from original task, but also by influencing the decision making (increasing decision time or decreasing decision quality) [57]. Task's inherent characteristics (such as complexity) may determine the impact of the interruptions [57] and influence the perceived difficulty of recovering back to performing the original task [21]. One of the sources of interruptions is email [38, 64]. In the next section we will discuss the influence of email processing strategies on productivity and perceived stress.

Influence of Email Processing Strategies on Perceived Productivity and Stress

"Using email is one of the most common online activities in the world today." [39] According to the statistics published by technology market research firm [6] the number of emails send daily worldwide was around 205 billion in 2015 and as agency predicts, will continue to grow with the growing number of users, both business users and individuals.

Emails in the work environment represent a very important source of information and one of the most used tools for communication inside corporation [43]. It was shown that even software developers spend a significant amount of time (around 14%) on managing emails, whereas the time spent on development is only slightly more than 23% [45]. The results of experimental research show that not only the longer time which was spent on managing email [43], but also the frequency of checking emails [11, 39] were linked to lower productivity [11, 43] and higher levels of stress [39, 43].

Thus, adopting the email strategy which would limit the frequency of checking the emails (also known as batch-processing) could increase the productivity of processing the emails [11, 43] and reduce stress, which would lead to increase the personal well-being and "other positive outcomes including higher mindfulness, self-perceived productivity, and sleep quality." [39]

[6]The Radicati Group, Inc., `http://www.radicati.com/?p=12964`

Chapter 4

Constraint-based Workday Planning

In this chapter the developed concept for automatic schedule generation will be described, including the rules which will be used for calculating the productivity constraints and general rules for task scheduling. The problem solved with the DayO application will be formulated as a CSP. The rules for assignment optimization will be described in detail.

4.1 Overview

The concept of the constraint-based workday planning can be described as consisting of following parts: constraints (timing constraints as well as productivity rules), CSP description, and the procedure of schedule generation. Further, the concept relies on the tasks defined in the specific form, the events which are planned for the day, and the information about user's workday and the productivity factors on each individual day. Constraint-based workday planning can be defined as a procedure of generating and optimizing the schedule for a given day during which tasks for the day are chosen and assigned to the particular time in the schedule based on productivity and timing constraints.

The algorithm for automatic task scheduling used in this thesis receives the tasks which can be planned for a day and the free time slots with their respective starting times as input and considers different options of scheduling the tasks during the day, whereas the score for each task allocation is calculated signifying how 'optimal' this assignment might be. For each candidate schedule the total score is calculated and the option with highest score is considered the most optimal. There are several factors influencing the choice of the assignment:

- soft score calculated for each of the tasks based on the task attributes and on user information (rules are described in section 4.2.1);
- the feasibility of a solution is represented by negative hard score (for example, if two tasks overlap, the solution is considered unfeasible);
- number of tasks planned for the day (we want to maximize the assignment, ideally occupying each free slot in a day, but due to the rule implementation there is the tendency for soft score to be negative, therefore, fewer assigned tasks would be considered better option).

The procedure of schedule generation consists of the following steps:

1. Generate the set of free time slots for a day based on the events for the day;
2. Choose active tasks from all tasks which user has saved in the system (tasks are considered active when they are neither completed nor deleted);
3. Provide the list of active tasks and free time slots to the optimization algorithm;
4. Call the OptaPlanner's method for solving the CSP, providing it with the problem created in the previous step;
5. Save up to three best solutions to the provided problem.

On the problem solving stage, the solver algorithm calculates the soft score for each task of the assignment as it searches for the best solution. The soft score for task assignment is then considered to be weighted sum of each rule's scores.

Figure 4.1 represents the concept of automatic schedule generation. Blue items represent the prerequisites of the scheduling (next subsection reviews each of these prerequisites in detail); green items represent variables and values necessary for definition of the CSP (detailed definition of the scheduling problem as CSP is given in the next section 2.3; orange items are responsible for soft score calculation (schedule is optimized based on soft score); the violet item represents the algorithm of solving the CSP which is partially defined in DayO prototype and partially provided by the CSP solver library. More details about the implementation is given in chapter 5; the detailed description of CSP is provided in section 4.2.

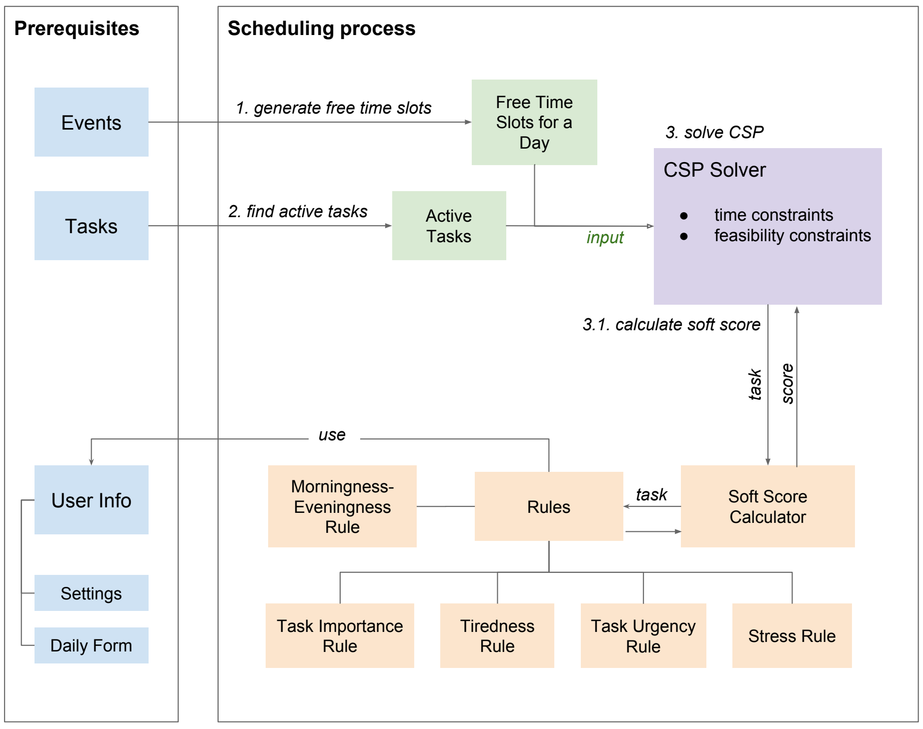


Figure 4.1: Overview of schedule generation prerequisites and process

4.1.1 Prerequisites

In this section the necessary preconditions for running the automatic scheduling algorithm will be described. In order to run the automatic schedule generation we need:

- tasks which can be planned (all tasks in the list can be completed at any time and are largely independent from one another);
- slots for planning the tasks;
- fixed appointments for the day;
- start and end of the workday.

In the following subsections each of the preconditions will be described individually.

Tasks

The procedure of generating a schedule relies on tasks having importance, difficulty and duration defined; and, optionally, due date. Additionally tasks have descriptions. We will review each parameter in detail in the following paragraphs.

In figure 4.2 the relationship between the task attributes and the rules is presented. The arrows from the rule blocks to the attribute blocks mean that the attribute is used for score calculation by the rule.

Importance. The task importance describes how important the task is compared to other tasks; it is vital to understand that importance is not the same as priority of the task. In the research of time management and personal effectiveness at work it was claimed that *"...a distinction should be made between the importance and urgency of tasks, because they are factors that affect why one task receives a higher priority than another task"* [16]. In our concept we would rely more on importance of the task to define task priority, leaving urgency (given by defining a due date of a task) as an optional parameter, since not all tasks might have a defined deadline. However, keeping in mind the relevance of urgency in task prioritization we would score the tasks with deadline as having higher priority than tasks without due date (more about this consideration can be read in the definition of the urgency rule in the following section).

Difficulty. The relative complexity of the task as compared to other tasks will be defined as difficulty of the task. It is an important parameter of the task, since different productivity factors will be influencing the efficiency of solving the task depending on task complexity. Solving complex tasks requires more attention, greater concentration and more mental capacity; therefore, the factors draining attention or concentration resources will be influencing more complex tasks to greater extent than easier ones [21, 29, 42, 57]. In our application we use task difficulty defined at three levels: challenging (for the most complex tasks requiring highest involvement and a lot of mental resources, for example planning a new project), regular (tasks which are usual in their demand for attention and mental resources, for example designing a form for the UI designer), and easy (routine or trivial tasks which almost do not require special concentration or straining one's mental abilities, as for example checking email might be). Depending on the difficulty of the task, some productivity factors such as sleep and chronotype will have greater or lesser influence on the total score of the schedule (more about the influence of task difficulty on score calculations can be read in the sections about tiredness rule and morningness-eveningness rule).

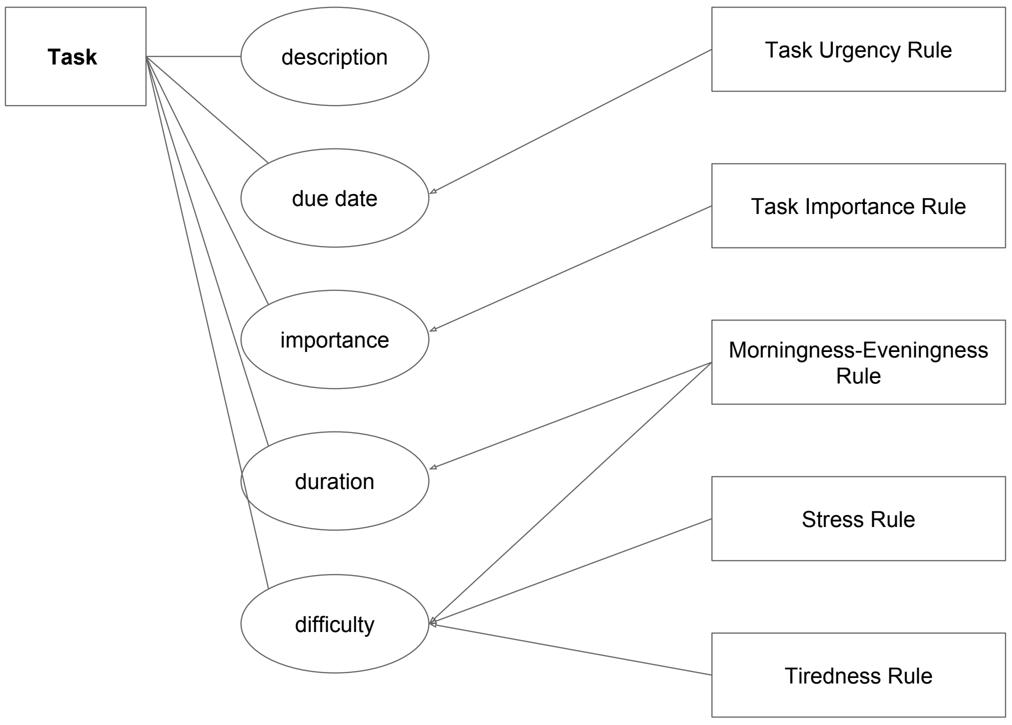


Figure 4.2: Relationship between task attributes and rules

Duration, Description, and Due Date. Task duration is a vital parameter for creating a schedule, since it defines how much time in the schedule the particular task will occupy. The estimation of the task duration as all other parameters of the task is a perceived value, therefore, the precision of the created schedule will significantly depend on the quality of the estimation. For example, if a task was estimated at 3 hours, and actually took 4 hours of time, then the schedule will not anymore represent the realistic situation. Task description is not required for scheduling of the tasks, but it is important for the user to be able to recognize and differentiate tasks. Tasks have due dates, by which they should be completed. The urgency of the task will be defined based on how soon the due date of the task is (more about the due date's influence on the score of the schedule can be read in the section about the urgency rule).

Events

We define the fixed appointments in the user's schedule as events. Events are important for planning of the day, since they define the amount of time available for the tasks to be accomplished during the workday. Based on the events of the day the free time slots for planning will be generated to position tasks "around" events in the schedule. The length of the event and the free time available before the event will also influence how the tasks are planned. More about the influence of the events on task scheduling can be read in the section describing the scheduling process.

User Information

To be able to do the planning, we need to have some information about the user, such as start and end of the workday. Since normally the start and end of the workday are fixed, they will be defined in the user settings and can be adjusted at any time. To use the morningness-eveningness rule we need the user to assess their chronotype from the point of view of their daytime productivity. As it is usual in the morningness-eveneningness research [34, 35, 49, 61], we ask the user to choose one of three types: morning type, neither type, and evening type. More about the influence of the chronotype on the task planning will be said in the section describing the productivity rules in 4.2.1, specifically the morningness-eveningness rule.

Daily Form Information. To be able to calculate the scores for the tasks, the information about the current state of the user regarding the considered productivity factors is required. For calculating the tiredness rule's score, the duration of sleep last night and the perceived quality of sleep are necessary inputs. The duration of sleep is entered by user between 0 and 10 hours with the step of 0.5h. The quality of sleep is defined as four levels: very bad, bad, normal, and good. The sleep quality and duration will influence the tiredness score given to each task (the detailed description of the tiredness rule will be given in the next section 4.2.1). The information about the perceived level of stress is also entered on the daily basis using the Likert scale ranging from low to very high level of stress. The current level of stress is used to calculate the stress score (the exact calculation strategy will be described under stress rule in next section 4.2.1). Since the information from daily form is required to generate the schedule, users will be asked daily to fill in the form and the schedule generation will be started after the form has been submitted.

4.2 CSP Formulation

In this section we will formulate the schedule generation for a day as a constraint-satisfaction problem. Each rule defined for calculating the productivity score is described individually.

As it was described in the background chapter 2.3, to define the CSP we need to provide the variables, the values with their respective domains, and describe the constraints. The CSP can have two types of constraints: hard constraints — the conditions which must be satisfied for a feasible solution, and soft constraints — conditions for optimization of a solution. First the problem of task scheduling is defined as a CSP, then the hard and soft constraints defined for restricting the solution space for the problem are described.

The scheduling problem is represented by the set of variables, the free time slots:

$$TS = \{ts_1, ts_2, ..., ts_n\}$$

each of time slots has the starting time attribute:

$$S = \{s_1, s_2, ..., s_n\}.$$

Tasks represent the set of values which are assigned to the time slots:

$$T = \{t_1, t_2, ..., t_m\}$$

each of the tasks has a duration attribute:

$$D = \{d_1, d_2, ..., d_m\}$$

Thus, the scheduling problem is a constraint-satisfaction problem of assigning a subset of tasks T' to the subset of free time slots TS'. Since the duration of the time slot is set to 15 minutes, probably not all time slots will be assigned a task, unless the duration of each of the assigned tasks is exactly 15 minutes. The same is true for the tasks, since in the normal case , we would be choosing from more tasks than the number of tasks which would fit into an individual day. Therefore, we could formulate the scheduling problem more precisely as follows: choose a subset of tasks to be assigned to the subset of the time slots in such a way that the constraints (hard) are satisfied and schedule is optimized regarding the productivity constraints (soft constraints).

Hard Constraints. The hard constraints of the problem are the constraints that must not be broken. They define the qualities a feasible solution must have. Thus, we will define the qualities of a valid schedule (or feasible task assignment) as follows:

1. Task-Event Collision Constraint — tasks planned for a day must not occupy the same time as the events of the same day;
2. Task Overlapping Constraint — tasks must not overlap with other planned tasks;
3. Event Overlapping Constraint — the general overlapping of the task with the event is possible, but the portion of the task before the event and portion of the task after the event must be large enough;
4. Workday Duration Constraint — all tasks must start at or after the start of the workday and end at or before the end of the workday.

More precisely the hard constraints can be formulated as follows. Let $E = \{e_1, e_2, ..., e_k\}$ denote the set of events for a day and $TS = \{ts_1, ts_2, ..., ts_n\}$ denote the set of the time slots which are the planning variables for the schedule of the same day. Let $T = \{t_1, t_2, ..., t_m\}$ denote the set of starting times of each of the time slots. Let further se denote the starting time of the event and ee denote the end time of the event (each event has a starting time and end time). Then the first constraint is:

for all events in $e_j \in E$ and all time slots in $ts_i \in TS$, there does not exist the time slot ts_k for which t_k denoting the starting time of the time slot is in the interval of $[se_j, ee_j)$.

(Task-Event Collision Constraint) Satisfied, if there are no such time slots defined for a day which would overlap with the events of the same day.

Let the task t_i with duration d_i be assigned to the time slot ts_j which has a starting time s_j, then for all tasks $t \in T$ there is no such task t_k with duration of d_k assigned to the time slot ts_m with the starting time s_m, such that

$$(s_j > s_m \wedge s_j < s_m + d_k) \vee (s_j < s_m \wedge s_m < s_j + d_i) \vee (s_j < s_m \wedge s_m + d_k < s_j + d_i)$$

(Task Overlapping Constraint) For all tasks assigned to the time slots, there should be no such assignment that the next task would start before the previous one has ended, meaning that there must be no overlapping tasks in the valid assignment.

(Event Overlapping Constraint) Tasks are not allowed to be separated by the event into the portions which are too small, since the interruptions have negative influence on productivity [21, 57, 58]. The constraint is implemented in such a way that the tasks with $d < 60$ are not allowed to be interrupted and the portion of the task p should be larger than a defined constant (now defined at 30 minutes).

Let t denote a task with duration d and e denote an event which overlaps with the task with the starting time of the event denoted by se and end of the event denoted by ee, further let the minimum portion duration be denoted by D_{min}, then for any task $t \in T$ assigned to the time slot $ts \in TS$ with starting time s the following must be true:

$$|s - se| \geq D_{min} \wedge |d - (|s - se|)| \geq D_{min}$$

(Workday Duration Constraint) All feasible schedules must fit into the workday, meaning that for all assigned tasks, the start must be after the start of the workday and end of the task must be before end of the workday.

Let S_d denote the start of the day and E_d — the end of the day, then for all assigned tasks $t \in T'$ and all time slots with task assignment $ts \in TS'$, the following must be true:

$$\nexists ts \in TS' \wedge s < S_d \vee (s + d) > E_d, \forall t \in T'$$

Soft constraints. In our definition of creating a task assignment for a day, the soft constraints are the rules based on which the more optimal regarding the user's productivity assignment of tasks could be chosen. The soft constraints can be defined as negative, or positive constraints, therefore, the score given by the soft constraints may be either negative or positive and the absolute difference between the score will be representing how much more or less preferable some particular assignment of the task is. In the current implementation of the soft constraints, they are represented as rules which give the score to each task based on the inherent qualities of the task (difficulty, importance, or due date), and/or the particular assignment, and the productivity factors of the user (morningness-eveningness, tiredness, or stress). The score given to each task by the rule shows how much this particular task (assigned to particular time) is preferred compared to the other tasks. In the following section each of the rules implemented in the DayO prototype will be described and the scores given by the rule will be explained.

4.2.1 Rules for Assignment Optimization

To create a schedule which will be adjusted to the user's productivity and optimized regarding the tasks' inherent parameters, the set of rules for calculating the soft score for each task is used. Two of the rules are related to task parameters only (the importance rule and urgency rule) and three other rules are related to user's productivity (tiredness rule, morningness-eveningness rule, and stress rule). The later three rules were chosen based on the research of the available literature on the factors of workers productivity which are described in detail in the related work chapter 3.3. Each of the rules returns a score for a given task, which in the current implementation ranges between [-100,100] points, the score showing the preference for the task assignment as compared to other tasks. Based on the score of individual tasks and the total score of the assignment the best (more optimal than other) option(s) is chosen. In the current implementation all rules must implement a method for calculating the score, which is called whenever next task is scheduled, and each rule has parameter *weight* showing how important it is compared to other rules.

Three productivity factors (sleep quality and duration, stress, and morningness-eveningness) were chosen, since according to existing research these factors have influence on the person's productivity independent from the baseline value of the productivity (when compared to their usual level of productivity). For example, workers are less productive than usual, if the duration or quality of sleep were insufficient, [6,7,51,54], or experience high levels of stress [20,24,31]. It was also shown that the chronotype is a stable characteristic [49] and some research argues that people perceive themselves to be more productive during a particular time of the day [6,42,45]. Other productivity factors such as work environment or interruptions were not chosen, since they are relatively stable or independent for making them a part of the factors which influence the planning of the schedule.

Two rules related to tasks' characteristics are there to show the preference between different tasks. For example, the task with the higher priority is strongly preferred to the task with lower priority. The more urgent tasks are preferred to the less urgent ones. In the following sections we will discuss each of the rules and the details of their implementation.

Generally, the rules related to productivity are not trying in any way to measure or predict the size of the productivity loss (or gain), but rather define the sign of impact on efficiency (either negative or positive) and some rules differentiate between the tasks based on the task difficulty. For example, if the person is very stressed, or very tired, we assume that more complex tasks would suffer greater impact than easier ones. Thus, knowing that measurement of productivity of the knowledge worker is a challenging task [14] and our research has not revealed methods which can be applied reliably to many types of the knowledge work, in the rules we will rely on the "baseline" productivity of the person which can be either improved or impeded by the influence of the productivity factors.

Tiredness Rule

Tiredness rule uses information from the daily form which user fills in (sleep duration and sleep quality) and the difficulty of a given task as variables influencing the resulting score. Generally, the larger sleep debt (hours slept – suggested sleep duration) the lower the productivity (and thus the given score); the worse quality of sleep the lower is the productivity and the score. The complexity of the task influences the extent of positive or negative impact of the rule (the magnitude of the score), thus, the harder the task is, the more impact the previous two factors will have (positive as well as negative) on the score. For example, if the sleep quality or sleep duration is low, then the easy tasks are strongly preferred to the difficult ones, and tasks with normal difficulty are preferred to the ones with high difficulty.

This rule is mostly for penalizing the difficult tasks, if the sleep duration or quality were insufficient. If sleep quality and duration were good, then score will be positive, showing the boost to productivity, but the value of maximal positive score as compared to the absolute value of minimal negative score is much smaller, since the evidence found for the negative impact of insufficient sleep on productivity is more convincing [6,7]. Further, according to the available research the normal sleep duration for otherwise healthy adults is somewhere between 7 and 8 hours [6,7]. The sleep quality can be one of four different values: good, normal, bad, or very bad. The combination of the sleep debt and sleep quality define the sign (positive or negative) and basic score, afterwards being multiplied by the score for the task difficulty to incorporate the influence of the task difficulty.

The base score is calculated based on the sleep debt, which is the deviation from the normal sleep duration. In the current implementation the normal sleep duration is defined at 8 hours with the tolerance of 0,5 hours. Thus, if sleep duration falls into the interval $[7.5, 8.5]$, no penalties or boosts for the sleep duration would apply. Also the cutoff for the sleep debt was introduced to normalize the score to a desirable range. All sleep duration below 4 hours will be considered equally bad and will get the lowest sleep (base) score, but the quality of sleep and the task difficulty would still influence the final score. Further, each of the defined sleep qualities has a score representing the impact of the sleep quality; the scores are distributed as follows: good = 2, normal = 1, bad = -2, very bad = -3. The choice of these scores has been done empirically having the following in mind: the values must have not much distance between each other, but to be representative of the impact of the given value onto the productivity (and the final score). Therefore, the tiredness rule will be calculated as follows:

1. The sleep score is calculated based on the normalized sleep duration (cutoff at 4 hours) as difference between the normal sleep duration and the entered sleep duration; for normal sleep duration it is considered to be 1.0;
2. Based on the sleep score the influence of the sleep quality is calculated, whereas the good and normal sleep quality have positive effect, and other two — negative;
3. The score from the previous two steps is multiplied by the task difficulty score, which is 1 for easy, 2 for regular, and 3 for challenging task.

On the chart 4.3 the distribution of the values (scores) for the different sleep duration and sleep quality values is shown.

Task Importance Rule

Task importance rule is showing the preferences among tasks regarding the task importance. The rule is to distribute the score from -100 to 100 among three different task importance levels: low, medium, high. Thus, the scores are given as follows: low importance -100, medium importance 0, and high importance 100. This score distribution is showing that caeteris paribus the tasks with high priority are strongly preferred to the tasks with lower priority, meaning that with all other parameters equal the schedule with more tasks with higher priority gets a higher score and thus is considered more optimal.

Task Urgency Rule

Task urgency rule calculates the score to represent the urgency of the task compared to other tasks. The rule is calculated based on the task due date, which is an optional attribute of a task. The score given to a task lies within interval $[-100, 100]$. Tasks without due date are given the minimal score, other tasks' scores depend on how close the due date is to the present moment. The maximal score is reserved for tasks due on the same day when the schedule is made, or are already overdue. The formula describing the distribution of the values for the other tasks was found experimentally based on the following considerations: tasks are getting much higher score if they are closer to the current date; having the minimal score equal to -100 and maximal score equal to 100. The cutoff (or threshold) of 4 weeks was introduced to define which tasks are having the smallest score, event if they have a due date, since to our understanding the due date which is more than 4 weeks away is equivalent to having no due date at all, if looking at this aspect from the perspective of the schedule generation. The formula describing the score distribution can be defined as follows:

$$-sin(x/6.5 + 99) * 100$$

On the next figure (4.4) the distribution of the urgency score is shown (x — the working days to deadline; y — urgency score). Only the workdays are considered. Every value of x which is smaller than 0, or larger than 20 gets the maximal and minimal score respectively.

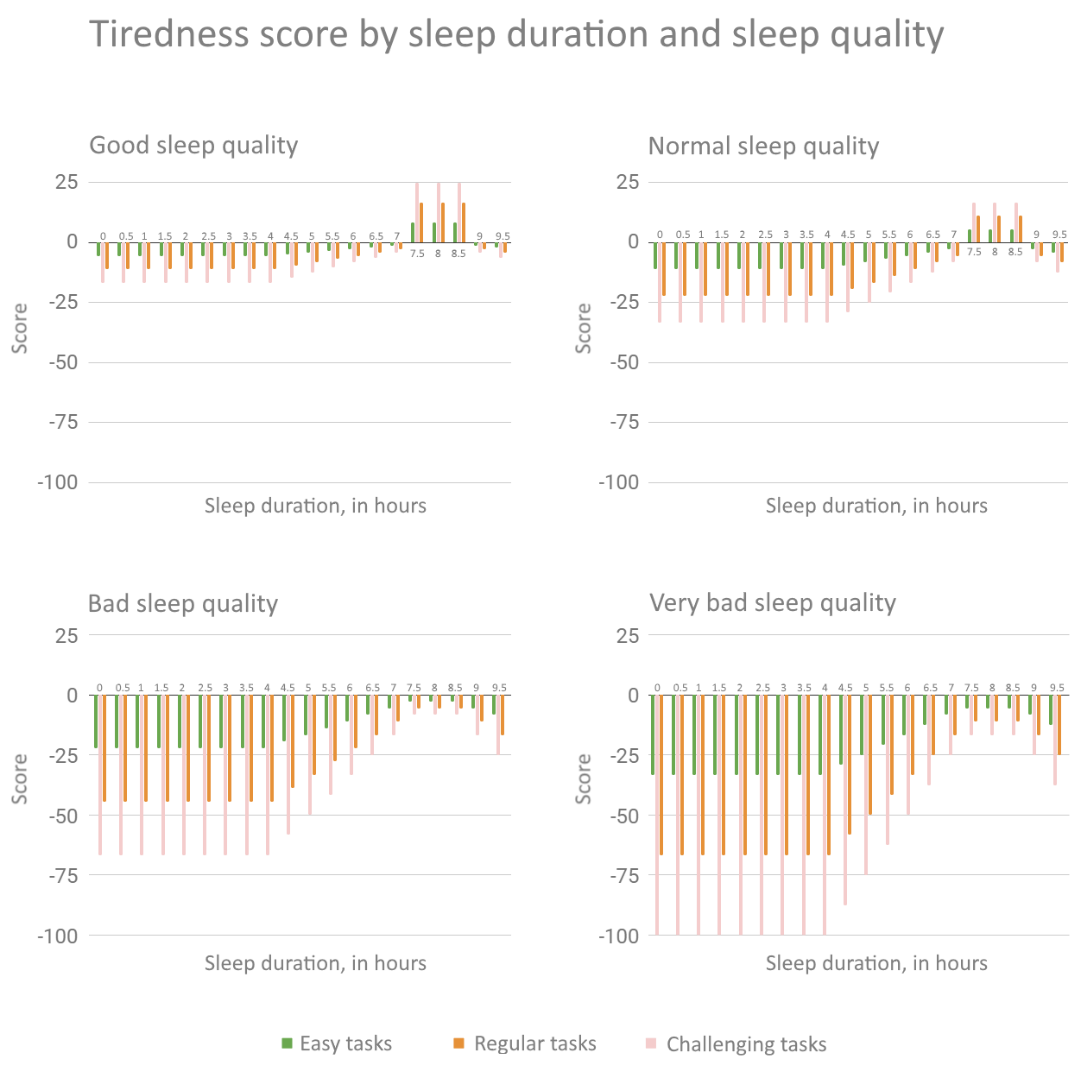


Figure 4.3: Score distribution for sleep quality from good to very bad and sleep duration from 0 to 9.5 hours

Morningness-Eveningness Rule

This rule gives the score for the task depending on the time of day when the task is assigned and the chronotype setting of the user. Further, the 'more productive' periods are defined for each of chronotypes except the neither type. The productive periods are defined as follows: morning type's productivity period starts at 8:30 and ends at 11:30, the evening type's productive period starts at 14:00 and ends at 17:00, thus each of the productive periods being three hours long. Generally, it is strongly preferred to assign the most difficult tasks during productive periods. Therefore, the score of this rule will be conforming to the following statement: $score(CHALLENGING) > score(REGULAR) > score(EASY)$ given tasks with similar duration assigned to the same time. If a task fits to the productive period only partially, the fraction of the score will be returned which corresponds to the fraction of the task duration which fits into productive period. As for all other rules, score belongs to the interval $[-100, +100]$.

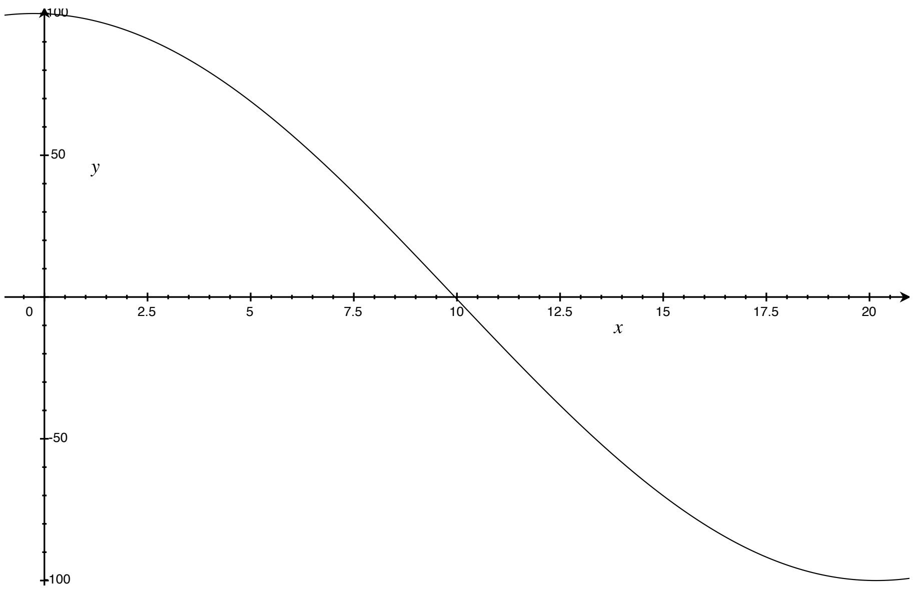


Figure 4.4: Urgency score distribution by the workdays until deadline

When the task is scheduled to some time beyond productivity period, the rule returns 0 because there are no preferences between tasks outside of productivity period from this rule's point of view. 0 is also returned for users who defined their chronotype as "neither", because this type means that there is no period during the day when user feels more productive than during other periods. Thus, the main goal of this rule is to differentiate tasks based on difficulty, if the task is assigned during productive period, making the assignment of more difficult tasks during productive period more preferable to easier tasks. For example, given user with morning type and having two schedules, otherwise identical except for assignment of the task from 9 to 11, where in the first schedule the easy task is assigned, and in the second schedule the challenging task is assigned — the second schedule will receive higher soft score and thus be preferred to the first.

Stress Rule

Stress rule is penalizing more difficult tasks with the negative score, if the stress level is higher than usual. With five different stress levels (insignificant, usual, higher, very high, and too high) with a score defined for each ($[0, 0, -1, -2, -3]$) the final score is calculated as follows:

let stress level be denoted by

$$L = \{l_1 = \text{insignificant}, l_2 = \text{usual}, l_3 = \text{higher}, l_4 = \text{very high}, l_5 = \text{too high}\}$$

and the task difficulty defined as

$$D = \{d_1 = \text{easy}, d_2 = \text{regular}, d_3 = \text{challenging}\}.$$

Then the base score is calculated as follows:

$$l_i * d_j.$$

The base score is then multiplied by a constant k (defined as the length of the negative part of the score range (from -100 to 0) divided by the absolute value of the minimal base score $base_{min} = score(l_5) * score(d_3)$) to normalize the base score to the scale equivalent with other rules. Thus, the score returned by the stress rule is calculated using the following formula:

$$l_i * d_j * k,$$

where $k = 100/base_{min}$.

The rule can be explained as follows: the higher the stress level and the higher the complexity of the task — the lower the score returned by the stress rule. However, the first two levels of stress have no negative impact on the productivity, therefore, the score returned for these stress levels by the rule is equal to 0. Stress level used in this application is a perceived value and represents more the perceived level of the general well-being of the user on each individual day than exact stress level the person may experience acutely or chronically (which could be measured more precisely using methods from medical or psychological research). This, however, does not reduce the claim that the stress as we use it in this application has influence on the worker's productivity, since general well-being was shown to correlate with knowledge worker's productivity [24,37].

Chapter 5

DayO Prototype

In this chapter the design and implementation of the DayO prototype are presented, giving an overview of architecture, description of the system's components, the brief account of incorporated APIs, and review of design decisions made during implementation.

5.1 Overview

DayO prototype is a web application with a client-server architecture and the database. Prototype's architecture is described in the next section. The server is implemented in Kotlin[1] and Java using Spring Boot[2] framework; client is implemented in TypeScript[3], HTML, and CSS using Angular 4[4] framework. Dependency management is done using Maven[5] on server and npm[6] on client.

The core feature of this application is to provide the automatically generated schedule by choosing the tasks from the user-defined task list and assigning them to specific time based on time and productivity constraints. There are several other features which are necessary to be able to provide the core feature:

- providing an interface for user to enter, alter, complete, delete, and view tasks;
- providing possibility for choosing and viewing the schedule for a day;
- interface for adjusting the user settings and filling in the daily form containing questions about the productivity factors;
- saving relevant data to the database;
- synchronizing with Google Calendar API to get the events for a day.

Thus, user interacts with the application to enter, alter, and delete tasks, save relevant information (which is required for schedule generation), and view the tasks and generated schedule. In normal use case when user logs in to the application, he is asked to fill in the daily form, and as soon as this form is submitted to the server, the process of generation of three possible schedules

[1]https://kotlinlang.org/
[2]https://projects.spring.io/spring-boot/
[3]https://www.typescriptlang.org/
[4]https://angular.io/
[5]https://maven.apache.org/
[6]https://www.npmjs.com/

is started. After the scheduling algorithm has finished execution, schedules are available for user to choose. Once the choice of the schedule for a day is made, user will see the dashboard with this schedule and the task list. Afterwards user can update the task list by completing, deleting the tasks etc.

For solving the CSP this application uses the OptaPlanner [2] software which is available under the Apache Software Licence [7] and Google Calendar API [8] is used for getting the events for a day, more information about used APIs is provided in the section 5.6. Other used APIs are Spring Security [9], Java Persistence API [10] on server side; Bootstrap (v4.0.0-alpha.6) [11] and some modules of d3 [12] on client side; the explanation for choice of these APIs given in the 5.6.

In the following sections the architecture of the application including the data model is described, the explanation of the API choice and usage is given, and user interface design is discussed, including the interaction and it's different cases.

5.2 System Architecture

Figure 5.1 provides an overview of the application architecture, presenting the main components of both client and server. Due to number of classes, only few of them could be presented in this first overview, but there is a detailed class diagram for the domain classes (Figure 9.3), the full description of the other classes of the backend module can be found in section 5.4. Further more representations of the components and packages are provided. Therefore, the following architecture representation shows the structure of the application from high level.

The application consists of the following parts: frontend, backend and database. In order to reduce effort spent on database set up for local development environment, embedded database was used. When deployed, PostgreSQL is used instead of embedded database. Backend module is responsible for security, creation, saving, and accessing of objects, as well as for logic of schedule generation and optimization, providing REST interface for communication with the client. Frontend module defines the representation of UI elements with HTML and CSS, functions required for rendering these elements, navigation through application, and synchronization with the server using the REST API provided by the backend module. Two modules are technically separate applications and could be run on different servers independently from one another, thus, it was possible to test and develop two modules independently. In the future this decision allows for flexibility to exchange frontend or backend implementation, provided that the the REST interface remains stable.

To wire the two parts of the application together and be able to deploy them as one .jar file, Maven was chosen. From Maven point of view, the application consists of 3 modules: parent module, frontend module and backend module with a dependency on the frontend module. Both frontend and backend modules defined in respective pom.xml files include a reference to the pom.xml of the parent module. The link to the database and the basic settings for it are included in the application.yml file in the backend module.

[7] `http://www.apache.org/licenses/LICENSE-2.0.html`
[8] `https://developers.google.com/google-apps/calendar/`
[9] `https://projects.spring.io/spring-security/`
[10] `http://www.oracle.com/technetwork/java/javaee/tech/persistence-jsp-140049.html`
[11] `https://v4-alpha.getbootstrap.com/`
[12] `https://d3js.org/`

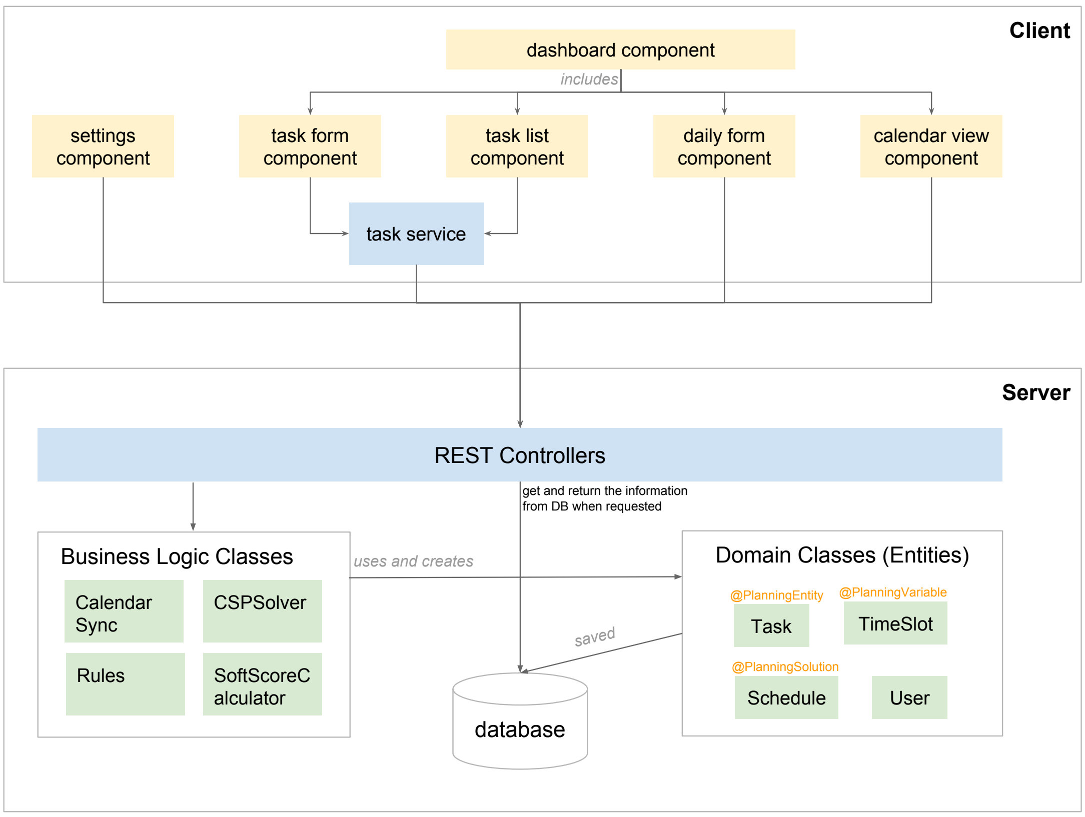


Figure 5.1: Architecture overview

To structure and organize the project, each module was divided into packages. On the diagram 5.2 the main components of the frontend and backend modules are visualized. Backend module was split into packages based on responsibilities of different classes; the alternative would be to split them by functional area, but the application does not have many independent functions and division into modules by functional principle would make little sense. Thus, all controller classes are grouped into `controllers` package, and domain classes into `domain_classes`. Application logic package contains all classes related to schedule generation, synchronization with the calendar API and have some non-trivial methods, in contrast with domain classes, which are just POJOs with annotations required for persistence or used by OptaPlanner.

In the frontend module the grouping was done based on the visual structure of the user interface and it's different screens. Therefore, our application has two main components (or screens): dashboard and settings, which are using the common components and services from `common` package. In the following subsections frontend and backend modules, as well as data model used for storing data in the database are described.

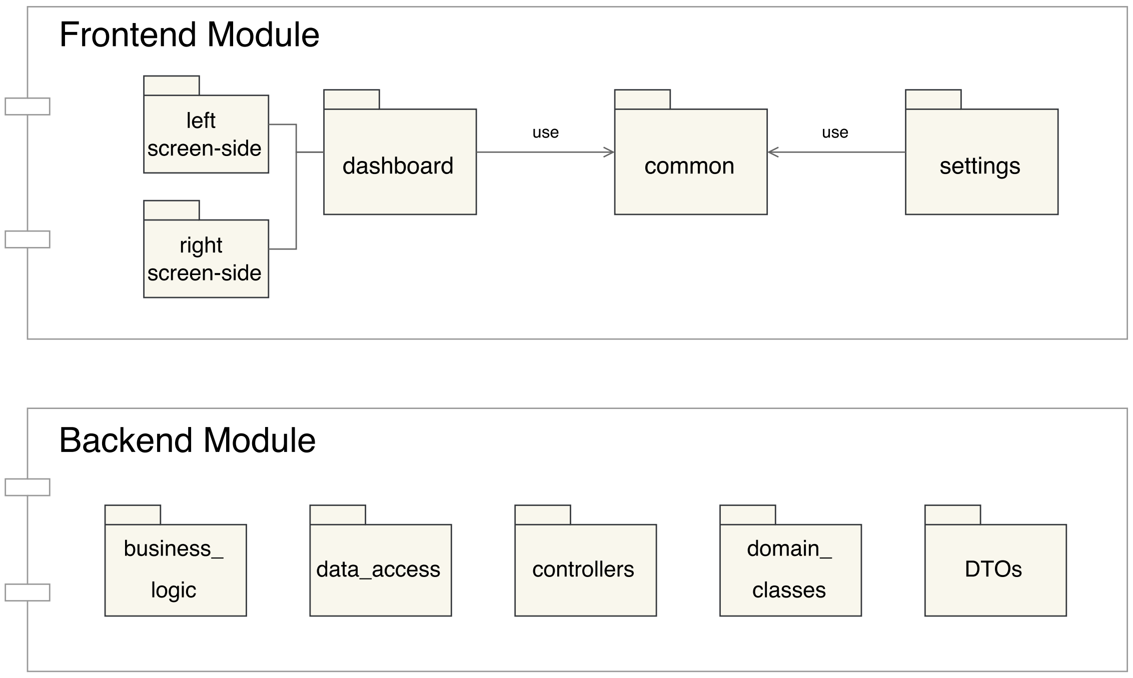


Figure 5.2: Packages of frontend and backend module

5.3 Client

Frontend module consists of Angular components (`@Component(...)`) in which display logic, styles and templates are defined; interfaces for representation of data structures which are coming from server; and services (`@Injectable()`) which can be injected into components and provide different functions (here services are used for communication with server and between components).

5.3.1 Components

Common. In this package of the frontend module functionality and elements which can be used from more than one component are included. For example, the constants are defined in one file `constants.ts`, therefore keeping all configurable aspects of the client-side application in one place. `TaskComponent` represents the task and is used both from schedule and from the task list. The functionality for accessing the server using correct address, as well as error handling is encapsulated in the `http-wrapper.service.ts` which defines all required http methods and as service can be injected into any component.

In this application guiding the user through different steps is very important, thus the `authentication-guard.service.ts` was created to implement handling of different cases which need to be supported for navigating user to the correct page, depending on the use case (the use cases and navigation will be described in 5.3.3).

Dashboard. The main view of this application is dashboard, but depending on the stage of interaction with the user (if user has settings, or tasks, filled in the daily form already, or not yet) the view can be very different. The screen is defined in file `dashboard.component.ts`. In this package components for event, schedule, and daily form representation are contained. On the diagram 5.3 the components and services of this package are shown with their dependencies on inner and outer components.

The types in quotation marks («») above the name of the component denote its type, thus, «Component» means that item is Angular `@Component(...)`, «Service» stands for services which are annotated with `@Injectable()` in Angular, and «Interface» — for TypeScript's interface [13]. We

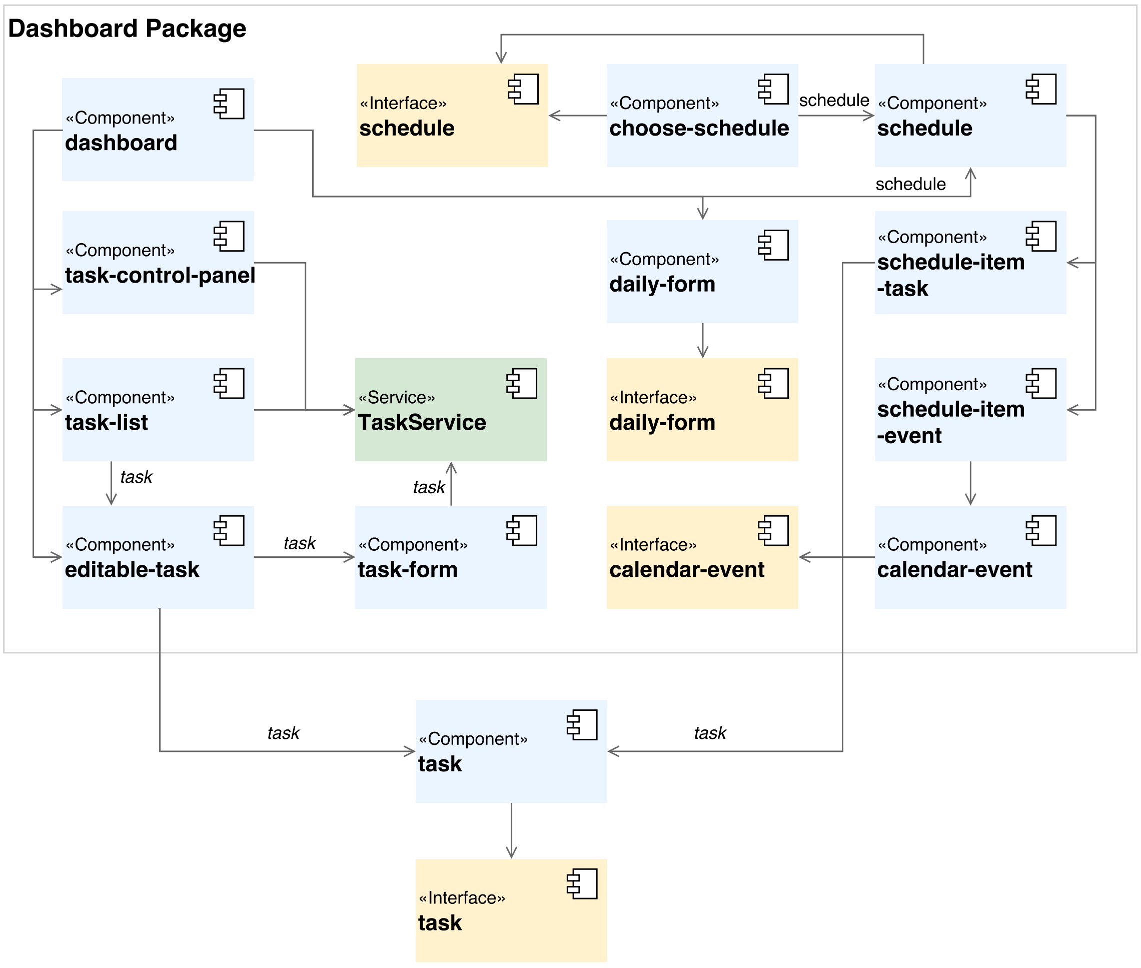


Figure 5.3: Overview of the structure of dashboard package on the frontend

can see from the representation that there are many components in this package, therefore, for the ease of use they were divided into two packages, as shown on figure 5.2. From the components visualized on the drawing 5.3 dashboard, choose-schedule, daily-form, and `TaskService` use the `HttpWrapperService` to connect to server and get or post relevant information.

[13] `https://www.typescriptlang.org/docs/handbook/interfaces.html`

The details of graphical representation are discussed in the section about user interface 5.3.2.

Settings. This package of the frontend module includes settings component and settings interface, but this component is presented on different screen to afford easy switching to it, since settings could be edited by user whenever there is a need for it.

Other components of the frontend module are the usual parts of Angular project, such as `app.-component.ts` with respective `.html` and `.css` files, `app.module.ts` (where all services, components and external modules used in the application need to be listed), `app-routing.-module.ts` (providing routing to components which can be used for navigation or testing purposes), `package.json` for dependency management, and `index.html` — the standard file for the default page which contains links to style sheets and scripts used in the application and the entry point for of the application. `Pom.xml` file helps to wire the whole application together and defines frontend as module managed by Maven, and references parent module. Other files in the project are automatically generated and were not changed during implementation.

5.3.2 User Interface Design

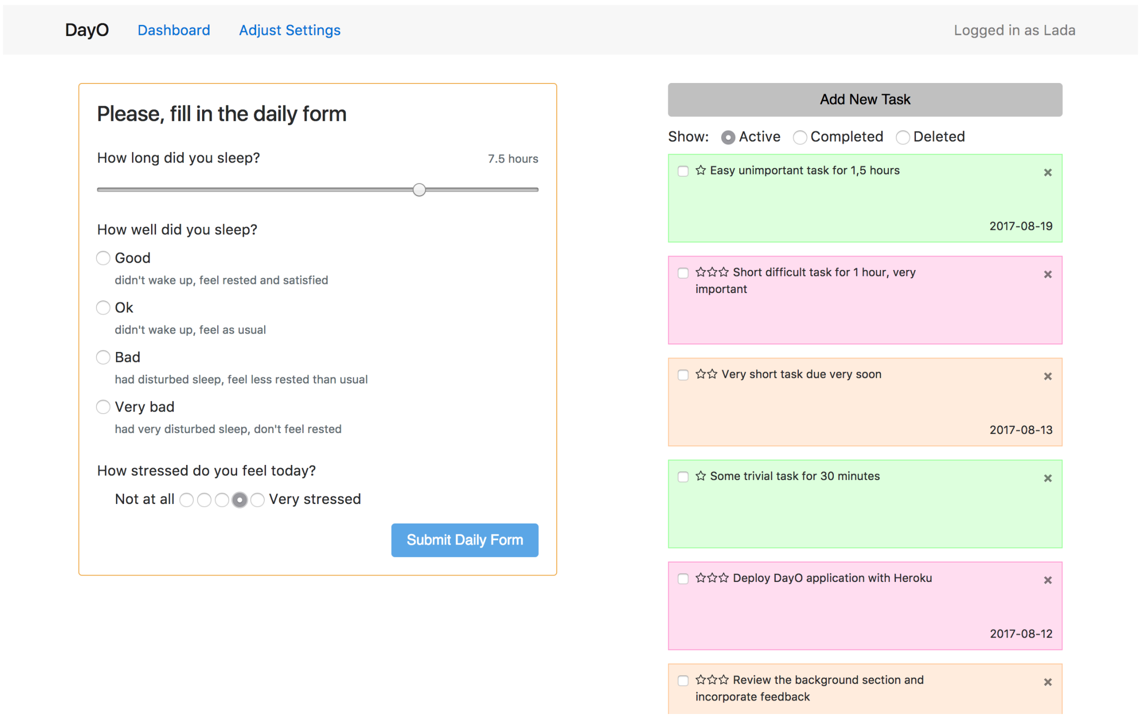
User interface was designed to be simple and understandable, providing all necessary information and guidance for the user. Individual elements were styled using CSS and classes from external style sheets (Bootstrap v4.0.0-alpha.6[14], Font Awesome[15])

User interface consists of several parts: settings page, daily form, task list with control panel and tasks, form for adding new task, waiting screen, schedule view, and screen for choosing the schedule for a day. The visual components are discussed individually in the following paragraphs.

Settings Page. Upon the first time login to the application, the user is asked to fill in the settings form, since having settings is a prerequisite for schedule generation. The form is implemented using standard Bootstrap form[16] controls and has the questions described in the prerequisites of CSP (section 4.1.1). The form is presented in the figure 9.1. The "Save Settings" button remains disabled until all fields are filled in.

Daily Form View. Every day before the schedule generation can be started, user is asked to fill in the daily form with questions relevant for understanding user's productivity. In the figure 5.4 the screen with daily form on the left and the user's tasks on the right is presented. Since not all fields are filled in yet and all of them are required, the button for submitting the form is disabled to prevent user from accidentally submitting the form with empty fields.

[14]`https://v4-alpha.getbootstrap.com/`
[15]`http://fontawesome.io/`
[16]`https://v4-alpha.getbootstrap.com/components/forms/`



DayO Dashboard Adjust Settings Logged in as Lada

Please, fill in the daily form

How long did you sleep? 7.5 hours

How well did you sleep?

Good
didn't wake up, feel rested and satisfied

Ok
didn't wake up, feel as usual

Bad
had disturbed sleep, feel less rested than usual

Very bad
had very disturbed sleep, don't feel rested

How stressed do you feel today?

Not at all ○ ○ ○ ● ○ Very stressed

Submit Daily Form

Add New Task

Show: [✓ Active] Completed Deleted

☆ Easy unimportant task for 1,5 hours — 2017-08-19

☆☆☆ Short difficult task for 1 hour, very important

☆☆ Very short task due very soon — 2017-08-13

☆ Some trivial task for 30 minutes

☆☆☆ Deploy DayO application with Heroku — 2017-08-12

☆☆☆ Review the background section and incorporate feedback

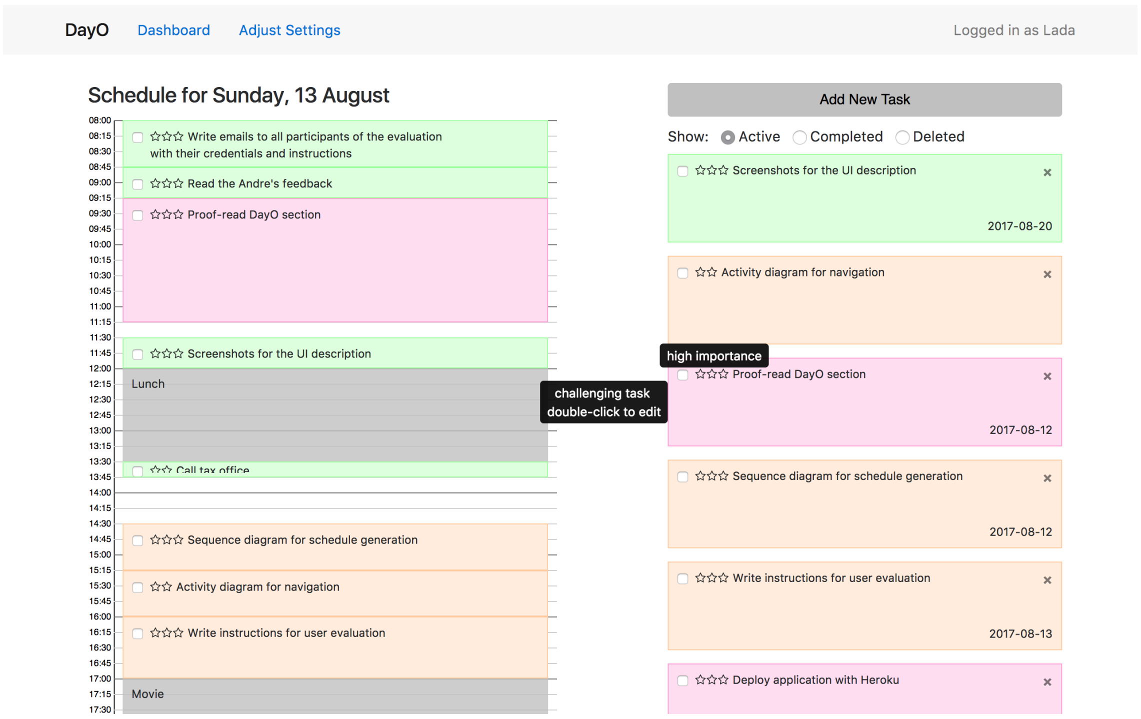
Figure 5.4: Screen with daily form

Task List. The task list consists of two parts: the control panel and the individual tasks. On top of the task list on the figure 5.4 there is a button "Add New Task" and below it there are filtering possibilities ('active' — shows the tasks which are not completed or deleted; 'completed' and 'deleted' options show the tasks which were completed or deleted, respectively). After clicking the "Add New Task" button the form for adding new task is displayed, in which the individual parameters of the task can be filled in. The new task form can be either closed, then no task is added, or submitted, if all required fields (description, duration, importance, and difficulty) are filled in.

Individual task is represented by the card with task description, importance, and due date, if task has a deadline. As it is shown in the figure 5.4 tasks have different colours depending on the difficulty: easy tasks are green, normally difficult tasks are orange, and challenging tasks are pink-coloured. Colour palette was chosen using online software for choosing colour combinations[17]. Importance is represented by star icons preceding the task description; low importance is represented by one star, high importance — by three stars. Task can be edited by double-clicking the task card; this action opens the form for adding the task with the fields pre-filled with the values of the currently chosen task. Double-click action was chosen over the button option, since there are already many repeating elements (button for deletion and the check-box). The tool-tip was added to provide user assistance with the colour meaning, importance, and possible interactions ("double-click to edit the task"). The task list with control panel and tool-tips is presented in the figure 5.5.

The tasks can be deleted by clicking the small "x" button in the task-card's right top corner, and the task will be marked as deleted in the database by setting "deleted" attribute of the task to

[17] `http://paletton.com/`



Schedule for Sunday, 13 August

08:00 08:15 08:30 08:45 09:00 09:15 09:30 09:45 10:00 10:15 10:30 10:45 11:00 11:15 11:30 11:45 12:00 12:15 12:30 12:45 13:00 13:15 13:30 13:45 14:00 14:15 14:30 14:45 15:00 15:15 15:30 15:45 16:00 16:15 16:30 16:45 17:00 17:15 17:30

☆☆☆☆ Write emails to all participants of the evaluation with their credentials and instructions
☆☆☆☆ Read the Andre's feedback
☆☆☆☆ Proof-read DayO section
☆☆☆☆ Screenshots for the UI description
Lunch
☆☆☆ Call tax office
☆☆☆☆ Sequence diagram for schedule generation
☆☆☆ Activity diagram for navigation
☆☆☆☆ Write instructions for user evaluation
Movie

☆☆☆☆ Screenshots for the UI description × 2017-08-20
☆☆☆ Activity diagram for navigation ×
☆☆☆☆ Proof-read DayO section × 2017-08-12
☆☆☆☆ Sequence diagram for schedule generation × 2017-08-12
☆☆☆☆ Write instructions for user evaluation × 2017-08-13
☆☆☆☆ Deploy application with Heroku ×

Figure 5.5: Schedule and task list view with tool-tips

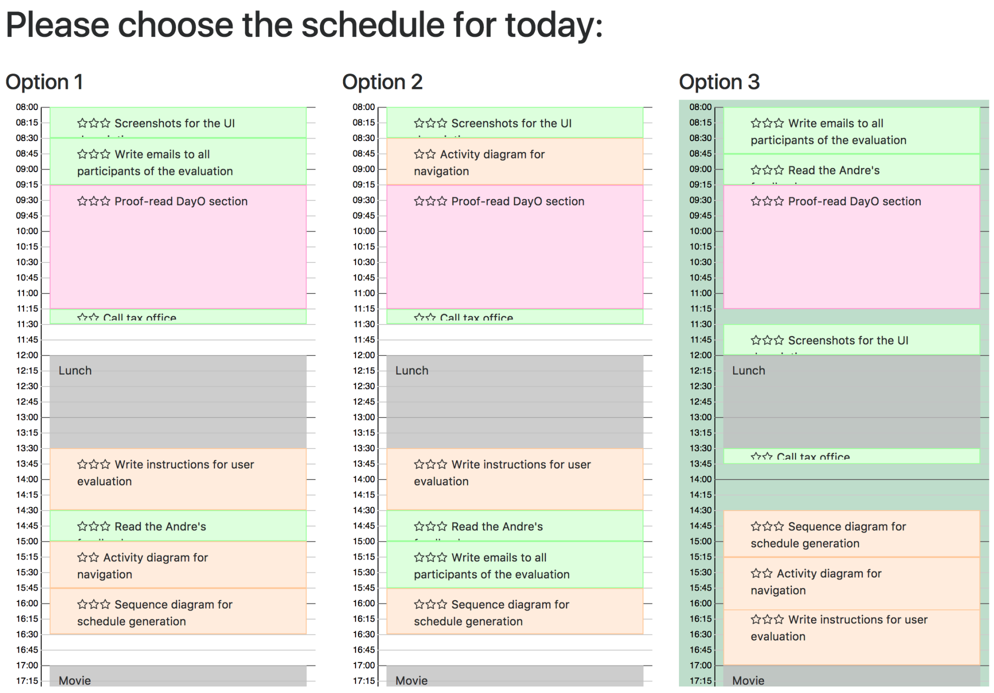
the current date. The completion of task is done by checking the check-box in the top left corner of the task, after the task has been marked completed on client, the "completed" attribute of the task will be set to the current date and completed task will disappear from "Active" list and appear on "Completed". Completing and uncompleting the task is also possible from the schedule view, as it is shown in the figure 9.2. To undo the completion of the task the check-box needs to be unchecked, after this action the state of the task is synchronized with the server and the "completed" attribute is set back to null.

Schedule view. To generate the schedule view, the height of each element and the offset is calculated and placed along the scale which is created using the d3-scale[18]. Events and tasks are placed in the order of their starting time along the scale with tick labels showing the time. In order to make the schedule more readable the tick marks were added (darker for the whole hour and lighter for a quarter of an hour). The events are represented with the grey box to be contrasting with the tasks, which are colourful, and the description of the event. If the tasks are too small to fit the description in (for example 15-minute task), the height will increase on hover to let user read the content of the task description.

When user is choosing the schedule, the suggested schedule alternatives, which were generated by the algorithm, are presented to the user on the separate screen. Options are given in no particular order and the score is not presented to the user to avoid leading user to prefer one schedule over another for some other reason than schedule content. When user hovers over the schedule, the background is changed and the pointer is shown to communicate that the element can be

[18]`https://www.dashingd3js.com/d3js-scales`

clicked. The screen with three schedule options is presented on figure 5.6. In the normal dashboard view (after schedule was chosen), it is presented on the left and the task list on the right. Current date is displayed above the schedule view.



Please choose the schedule for today:

Option 1
- ☆☆☆ Screenshots for the UI
- ☆☆☆ Write emails to all participants of the evaluation
- ☆☆☆ Proof-read DayO section
- ☆☆ Call tax office
- Lunch
- ☆☆☆ Write instructions for user evaluation
- ☆☆☆ Read the Andre's
- ☆☆ Activity diagram for navigation
- ☆☆☆ Sequence diagram for schedule generation
- Movie

Option 2
- ☆☆☆ Screenshots for the UI
- ☆☆ Activity diagram for navigation
- ☆☆☆ Proof-read DayO section
- ☆☆ Call tax office
- Lunch
- ☆☆☆ Write instructions for user evaluation
- ☆☆☆ Read the Andre's
- ☆☆☆ Write emails to all participants of the evaluation
- ☆☆☆ Sequence diagram for schedule generation
- Movie

Option 3
- ☆☆☆ Write emails to all participants of the evaluation
- ☆☆☆ Read the Andre's
- ☆☆☆ Proof-read DayO section
- ☆☆☆ Screenshots for the UI
- Lunch
- ☆☆ Call tax office
- ☆☆☆ Sequence diagram for schedule generation
- ☆☆ Activity diagram for navigation
- ☆☆☆ Write instructions for user evaluation
- Movie

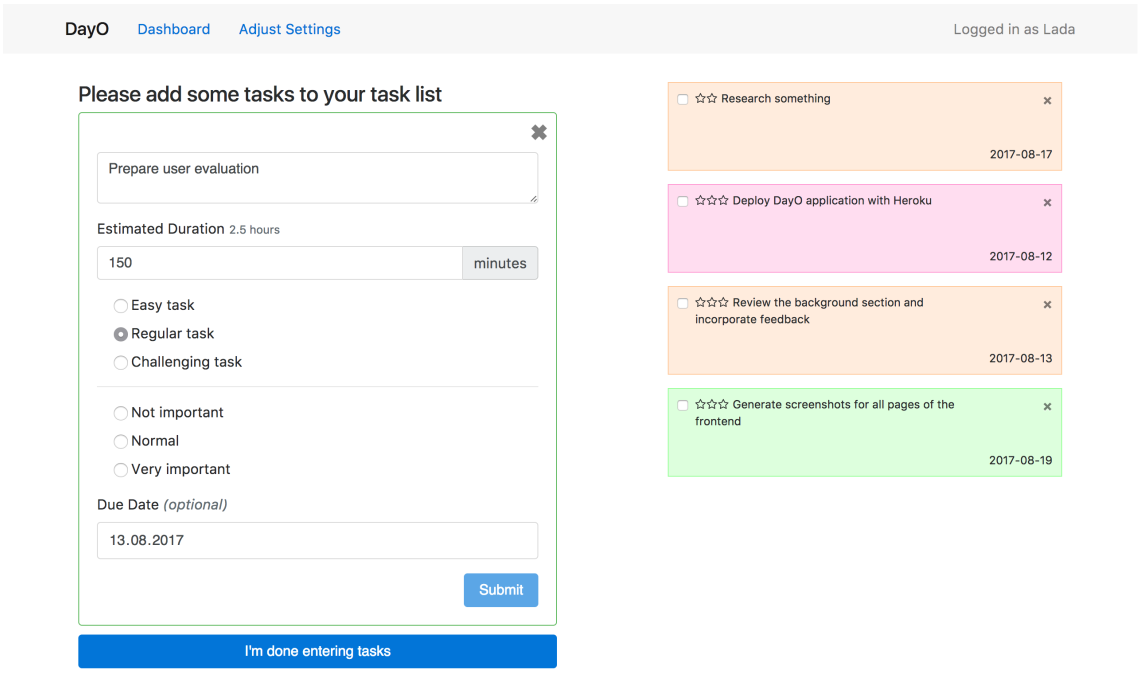
Figure 5.6: Screen with generated schedule options

Waiting Screen. Since the schedule generation can take longer time, communicating the current state of the application to the user is necessary to prevent them from navigating away from the page and show that no error has occured, and the waiting is part of normal program execution. Thus, after the submitting the daily form, user is navigated to the waiting screen which tells user about the state of the application asking him to wait while schedule generation is running. The countdown timer with estimated remaining waiting time is shown. The polling logic in this component periodically sends requests to the server; when response is not empty, user is navigated to screen with schedule options.

5.3.3 Interaction and Navigation

There are several situations important for the interaction and navigation which can be described as follows: user is new to the application, user logs in first time during the day, not first login during the day.

Depending on the situation the interaction and navigation will be different. When user is new to the application, he will be first navigated to fill in the settings, and then having no tasks stored in the database yet, he will be asked to enter some tasks, as it is shown in figure 5.7. Afterwards, interaction and navigation will be the same as for the second situation.



Please add some tasks to your task list

Prepare user evaluation

Estimated Duration 2.5 hours

150 minutes

Easy task [✓ Regular task] Challenging task

Not important Normal Very important

Due Date *(optional)*

13.08.2017

☆☆ Research something × 2017-08-17

☆☆☆ Deploy DayO application with Heroku × 2017-08-12

☆☆☆ Review the background section and incorporate feedback × 2017-08-13

☆☆☆ Generate screenshots for all pages of the frontend × 2017-08-19

Figure 5.7: View of the dashboard when user has no tasks

When user logs in first time a day, there is no filled in daily form yet, so the user will be asked to fill in the daily form. After the daily form is submitted, the schedule generation starts. Immediately after submitting the daily form, user is navigated to the waiting page, where he waits until the schedule generation algorithm is finished. Upon completion of the algorithm execution, the user is navigated to the page with the suggested schedule options. After user has chosen the schedule for a day, he is navigated back to dashboard, where chosen schedule is presented on the left and task list on the right. This screen is then used for all subsequent logins during the day.

Having chosen the schedule, user can view the dashboard, where he can change the task list by completing, deleting, adding, or editing the tasks. It is also possible to navigate to settings page and edit settings, which are submitted to the server when "Save Settings" button is clicked.

The interaction with the user and navigation, which is based on the information coming from the server and decisions made by client, are presented in figure 5.8.

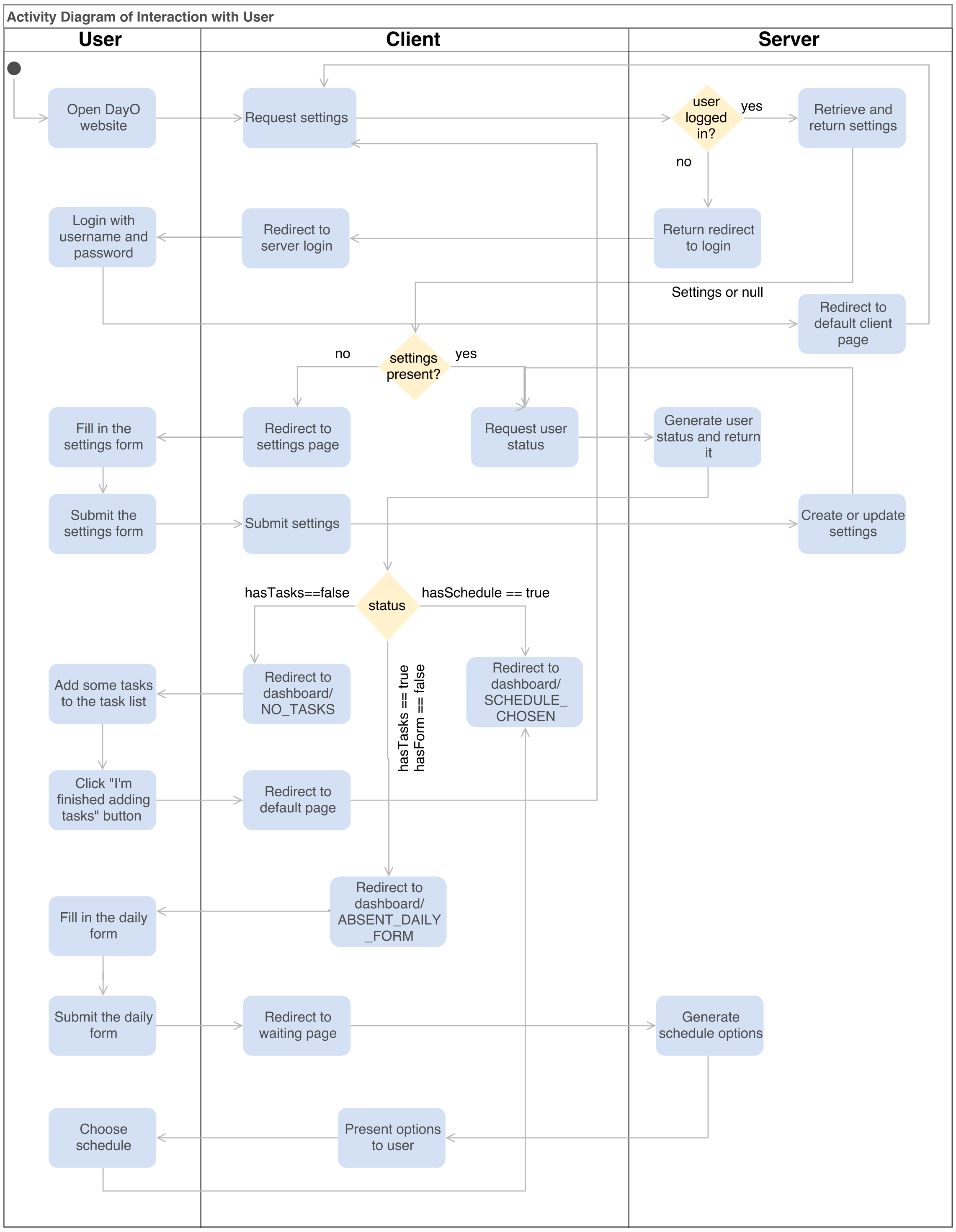

Activity Diagram of Interaction with User
User
Client
Server
Open DayO website
Request settings
user logged in?
yes
Retrieve and return settings
no
Login with username and password
Redirect to server login
Return redirect to login
Settings or null
Redirect to default client page
no
settings present?
yes
Fill in the settings form
Redirect to settings page
Request user status
Generate user status and return it
Submit the settings form
Submit settings
Create or update settings
hasTasks==false
status
hasSchedule == true
Add some tasks to the task list
Redirect to dashboard/ NO_TASKS
hasTasks == true
hasForm == false
Redirect to dashboard/ SCHEDULE_ CHOSEN
Click "I'm finished adding tasks" button
Redirect to default page
Fill in the daily form
Redirect to dashboard/ ABSENT_DAILY _FORM
Submit the daily form
Redirect to waiting page
Generate schedule options
Choose schedule
Present options to user

Figure 5.8: Activity diagram of user interaction and navigation

5.4 Server

The backend module is implemented in both Java and Kotlin. The domain classes are written in Java and everything else — in Kotlin. This decision was made because it is possible to combine two languages in one application, since both of them run on JVM, so calling Java code from Kotlin is possible as well as vice versa. Moreover, Kotlin is more concise than Java, enabling to write less code, and provides useful features like companion objects, file level functions and constants, extension functions and properties as well as very concise data classes. An example of a data class with two extension functions is presented in the listing below (5.1).

```kotlin
package ch.uzh.ifi.seal.DTOs

import ch.uzh.ifi.seal.domain_classes.AssignedTask
import ch.uzh.ifi.seal.domain_classes.Task
import ch.uzh.ifi.seal.domain_classes.TaskDifficulty
import ch.uzh.ifi.seal.domain_classes.TaskImportance
import java.time.LocalDate
import java.time.LocalTime

/**
* Class for passing the information about the task and task assignment to client.
* Has id which references the Task.id to allow manipulation on tasks
*/
data class TaskDTO(val id: Int, // task id
                   val description: String,
                   val duration: Int,
                   val difficulty: TaskDifficulty,
                   val importance: TaskImportance,
                   val startingTime: LocalTime?,
                   val dueDate: LocalDate?,
                   val completed: LocalDate?,
                   val deleted: LocalDate?)

fun AssignedTask.toDTO(): TaskDTO {
   val task = this.task
   return TaskDTO(task.id, task.description, task.duration, task.difficulty,
                  task.importance, startingTime = this.startingTime,
                  dueDate = task.dueDate, completed = task.completed,
                  deleted = task.deleted)
}

fun Task.toDTO(duration: Int? = null, startingTime: LocalTime? = null): TaskDTO {
   return TaskDTO(id, description, duration ?: this.duration, difficulty,
                  importance, startingTime = startingTime, dueDate = dueDate,
                  completed = completed, deleted = deleted)
}
```

Listing 5.1: An example data class with extension functions

5.4.1 Domain Classes

The domain of the application which includes tasks, events, schedules, and information about the user and his or her productivity factors, is modelled with domain classes. Many (but not all) of these classes are persisted in the database. This decision was made due to special requirements of CSP solving software [19] and for convenience purposes. The portion of the classes and attributes which are persisted is presented in subsection 5.5. The class diagram of domain classes is available in Appendix (fig. 9.3).

According to their purpose, domain classes can be divided into classes used for optimization, other domain classes, and enumerations. Further we will discuss the classes used for optimization in detail, and briefly describe other classes and enumerations.

Classes Used for Optimization with OptaPlanner. Three of domain classes are used for optimization: `Task`, `Schedule` and `TimeSlot`.

`Task` represents a task, which user defines, as well as a planning entity; it is annotated with `@PlanningEntity` annotation of OptaPlanner and, being a persisted class, it also has `@Entity` annotation. Each task has a reference to a `TimeSlot` (planning variable in our CSP formulation), the assignment of which is altered during planning between score calculations. This attribute (`startingTimeSlot`) is used in the process of solving of the scheduling problem to keep track of task assignment in the schedule, but is not persisted to the database.

Since `Task` can be assigned to multiple schedules, the information about the time and schedule is not kept in `Task`. For each generated schedule `AssignedTask` is created to hold information about the starting time, the reference to the original task, which holds all relevant attributes, and the schedule, to which it was assigned. `AssignedTask` is persisted in the database to save the information about the suggested schedules which were created for a day. Later this information may be used to see how often particular task was assigned and after how many assignments it was completed. Task's attributes saved to the database are shown in the figure 5.11.

`TimeSlot.java` represents the time slot, to which the `Task` is assigned. It does not have class-level annotation, but the respective fields in `Task` and `Schedule` are annotated with OptaPlanner annotations to show that `TimeSlot` is a variable in the planning problem. The two fields of this class are: id and startTime. Id is used for identifying the time slots, `startTime` denotes the starting time of the time slot. The duration of all time slots is equal and can be set in the file `ConfigurationConstants.kt` by changing the value of the constant `TIME_SLOT_DURATION`. In current implementation the duration was set to 15 minutes.

`Schedule.java` represents the schedule of a day as well as form for definition of the planning problem and planning solution; it is annotated with OptaPlanner's `@PlanningSolution`. This class implements `Solution<HardSoftScore>` interface provided by OptaPlanner. Schedule has information about the date, for which it was generated (`forDate`); start and end of the day (`startOfDay, endOfDay`); list of calendar events imported from user's calendar (`calendar-Events`); reference to the user, who owns it (`owner`); and the boolean field which indicates if the schedule was chosen for a day (`chosen`).

As a definition of the planning problem, schedule stores the list representing all active tasks available for planning `allTasks:List<Task>` annotated with `@PlanningEntityCollection-`

[19] OptaPlanner requires specific annotations for classes which are part of the planning problem `https://docs.optaplanner.org/7.0.0.Final/optaplanner-docs/html_single/index.html#modelAPlanningProblem`

`Property` and list of free time slots (`timeSlots:List<TimeSlot>`) annotated with `@ValueRangeProvider(id = "allSlots")`. Both of these attributes are used for solving the planning problem, but not persisted.

As planning solution, schedule has list of assigned tasks (`assignedTasks`), hard and soft scores which are stored in database as integers and transformed to `HardSoftScore` after `Schedule` was retrieved from the database and split into two fields before persisting it. The `score` attribute of the schedule serves as a measure of solution 'fitness' while the planning engine is running, so that possible schedules are compared using their `HardSoftScore` attribute. The schedule with higher hard and/or soft score is considered more optimal.

The classes described above are presented in the figure 5.9 with their respective fields, as well as both class- and method-level annotations. The diagram shows the attributes of the classes and non-trivial methods which are defined for these classes. The methods for getting and setting the attributes, which are defined for every field of the class, or standard methods of Java Object such as `hashCode()`,`equals(Object obj)`,and `toString()` are omitted in this diagram.

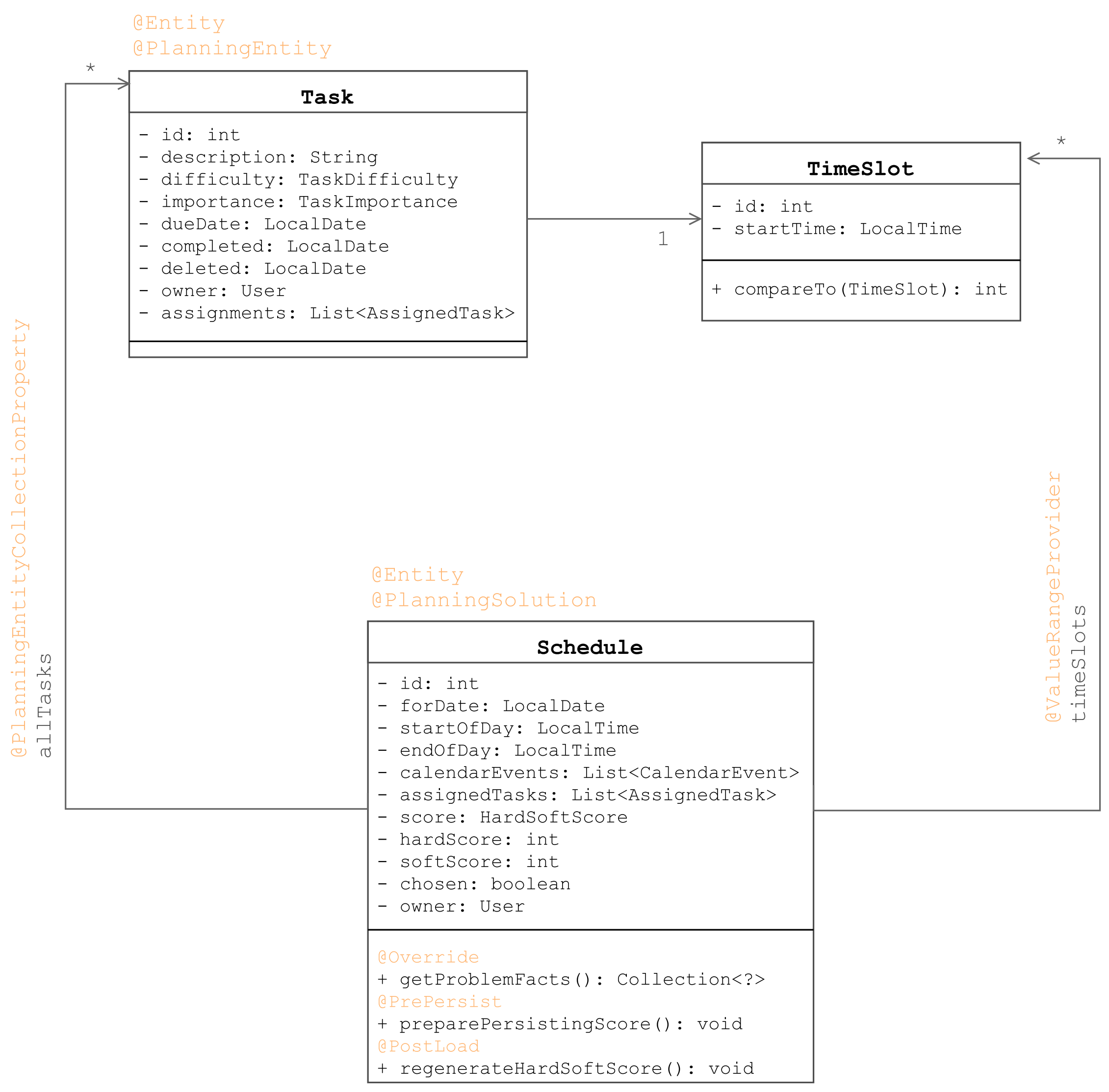


Figure 5.9: Domain classes which are used for optimization with OptaPlanner

Other Domain Classes. The rest of the domain classes model the entities necessary for saving and utilizing the information about the user and his schedules. These classes are:

- `User.java` — represents information about the user, such as name and email; many other classes are connected to the user as it can be seen on entity-relationship diagram 5.5;
- `Settings` — represents user's settings with the start and end of the workday and user's chronotype (`enum MorningnessEveningnessType`);
- `DailyFormInfo.java` — represents the daily state of the user's productivity using enumerations (`SleepQuality` and `StressLevel`) and `sleepDuration`; references the User who owns it;

- `CalendarEvent.java` — represents calendar event received from Google Calendar in the form compatible with the `Task`'s (using the Java 8 time API [20] for date and time);
- `AssignedTask.java` — represents assigned task with information about task assignment to a particular schedule and referencing `Schedule` and `Task`;

Enumerations. Some aspects of the domain are represented as enumerations. In the following list their meaning and values are described:

1. `MorningnessEveningnessType` — represents the chronotype of the user, more precisely relating to the time of the day when user might feel more productive — `MORNING_TYPE` (highest productivity in the morning), `NEITHER_TYPE` (not feeling more or less productive depending on the time of the day), `EVENING_TYPE` (highest productivity in the afternoon);
2. `SleepQuality` — describes the quality of sleep — `GOOD, NORMAL, BAD, VERY_BAD`;
3. `StressLevel` — denotes the level of stress which person currently experiences and which is representative of the level of general well-being — `INSIGNIFICANT, USUAL, HIGHER, VERY_HIGH, TOO_HIGH`;
4. `TaskDifficulty` — describes the complexity of the task. The more complex task is — the better concentration and the more mental capacity is required for completing it — `EASY, REGULAR, CHALLENGING`;
5. `TaskImportance` — signifies the relative importance of the task compared to other tasks. The priority of the task reflects task importance and it's urgency, based on the due date attribute — `LOW, MEDIUM, HIGH`;

The first enumeration is used in user's settings, second and third — in daily form, and the last two — in task. All enumerations except the first have the extension property[21] `score` which gives the integer score to each of the enumeration's values to provide basis for the score calculation. All extensions are defined in the file `ConfigurationConstants.kt` to be easily adjusted, if necessary. The extension property can be called on any value of the enumeration for which it is defined as any other property of the class might be called. In the listing 5.2 the example of definition of an extension property is shown and in the listings 5.3 the call to the extension property is shown.

```
val TaskImportance.score
   get() = when (this) {
      LOW -> 1
      MEDIUM -> 2
      HIGH -> 3
   }
```

Listing 5.2: Extension property for the TaskImportance enum

```
val importance = TaskImportance.LOW
val importanceScore = importance.score // gives 1
```

Listing 5.3: Example of calling of score extension property on enum

[20] `https://docs.oracle.com/javase/8/docs/api/java/time/package-summary.html`
[21] `https://kotlinlang.org/docs/reference/extensions.html`

5.4.2 Application Logic Classes

The classes of this package are responsible for logic of schedule generation and synchronization with Google Calendar to retrieve events for a day. In this section the classes with their functionality are reviewed; the sequence diagram representing the process of schedule generation is provided in Appendix chapter (fig. 9.4).

`CalendarSynchronization` class is responsible for connecting to the Google Calendar service, retrieving next n events, for each event creating a `CalendarEvent`, and putting them into the list which is returned by the public method `fun getCalendarEvents():List<CalendarEvent>`.

All other classes in "application_logic" package are responsible for the functionality of schedule generation. According to their main responsibility, these classes can be divided into two groups: score calculation and schedule preparation. In following paragraphs we will describe the two groups and their classes in detail, providing examples of code.

Schedule Preparation. For running the CSP problem solver, the schedule for a day needs to be constructed with required parameters, such as list of all active tasks for a day, list of time slots and events. The class `ScheduleGenerator` is responsible for the construction of the schedule and calling the solving algorithm after schedule was prepared.

`ScheduleGenerator` has a `SolverFactory` dependency of the type `Schedule`, which is constructed from the xml resource `solver-config.xml`, describing the classes which are used for optimization (`Schedule` and `Task`), the score calculator class (`DayAssignmentScoreCalculator`), and the methods for solving the CSP as well as maximum running time of the optimization algorithm. The configuration of the solver is provided in the listing 9.6.

`FreeTimeSlotGenerator.kt` file includes function for generating free time slots during a day. The user settings and the list of calendar event for a day are provided as parameters. From settings the start and end of the day are retrieved, the events represent the blocks of time, around which free time slots are generated. The public method `generateFreeTimeSlots(settings:Settings, fixedAppointments:List<CalendarEvent>):MutableList<TimeSlot>` creates time slots of the length set in the `ConfigurationConstants.kt` file which are not overlapping with events, thus, fulfilling the first constraint of the hard constraints described in 4.2.

The schedule preparation consists of the following steps:

1. Find the events for this day (today is passed as a parameter) from the list of events which were retrieved from user's calendar;
2. Find active tasks (not completed or deleted) from all tasks user has;
3. Based on the events for this day and user settings, generate free time slots for the day;
4. Construct a problem (`Schedule`), setting start and end of the day, events for the day, active tasks and free time slots;
5. With the solver, built from the resource (`solver-config.xml`), find three best solutions (schedules) to the problem defined in the previous step;
6. For each of the found schedules: transform tasks which were assigned to `AssignedTask`, and save them to the schedule's `assignedTasks` list;
7. Return the possible schedules for a day.

Score Calculation. `DayAssignmentScoreCalculator` is used by OptaPlanner Solver in the process of solving the CSP; it implements `EasyScoreCalculator<Schedule>` and is configured as class to be used for score calculation in `solver-config.xml`. The `hardScore` and `softScore` variables are defined in the beginning of `calculateScore(schedule:Schedule):Hard-SoftScore` method to keep track of hard score which represent the fulfilment of hard constraints for one possible solution, as well as soft score, based on which the optimization of the schedule is done. After the score calculation has finished, the `HardsSoftScore` object is constructed, based on which possible schedules are compared by the planning engine. The checks for hard constraint fulfilment and the process of score calculation in this class are presented in figure 5.10.

There are several private methods in `DayAssignmentScoreCalculator` which are used to check if the task assignment is feasible. The methods from the following list represent a cascade of checks executed for each assigned task of the schedule, passed as parameter; later methods are called if the previous method returned true (the cascade of checks is visualized 5.10):

1. `fitsIntoDaySchedule(task:Task, schedule:Schedule)` checks whether the task is inside the schedule (not before start and not after the end of it);
2. `overlapsWithEvent(task:Task, schedule:Schedule)` checks whether the current task assignment overlaps with some event from the schedule; if this method returns true, the next method in this list is called;
3. `canTaskBePartitioned(task:Task, events:List<CalendarEvent>)` method is checking whether the task overlapping with event can be split into portions large enough; returns true, if the splitting is possible and portions are large enough;
4. `overlap(curTask:Task, nextTask:Task):Boolean` method checks whether the two tasks overlap with one another; it is called for two tasks which are assigned one after the other, since the list of the tasks is sorted by the ids of the time slots, to which they are assigned;
5. `softScoreCalculator.calculateScore(task:Task):Int` is called for tasks which have passed all previous checks (thus the assignment is feasible) to calculate the soft score which represents the 'optimality' of such assignment.

The full code of `DayAssignmentScoreCalculator` class is provided in the listing 9.3 and the implementation of the function `findOverlappingEvents(...)` which is defined in the same file, as it is used both in this class and for generation of the task DTOs, is provided in the listing 9.4.

Further, there are rules for calculating the soft score which are described in 4.2.1. All rules implement the `Rule` interface and need to provide implementation for `init(user:User,today:-LocalDate)` and `getScore(task:Task):Int` methods as well as have weight attribute which signifies how much weight some rule has compared to other rules. All rules accept weight parameter in the constructor with the default value — a constant defined in the `ConfigurationCon-stants.kt` file. Since the scope of our study didn't include establishing relative importance of the factors influencing productivity, all rules have the same weight (0.2). Having interface and separate class for calculating the soft score (`SoftScoreCalculator`), it is easy to add new rules by implementing the interface.

In the listing 9.5 an example of the rule implementation is given. `TirednessRule` was one of the most challenging to implement, since it relies on three different parameters and has complex logic of assigning the score. As it is shown in the listing, there is detailed comment describing the logic of the rule; for each rule there is a comment describing the purpose of the rule and its score calculation strategy.

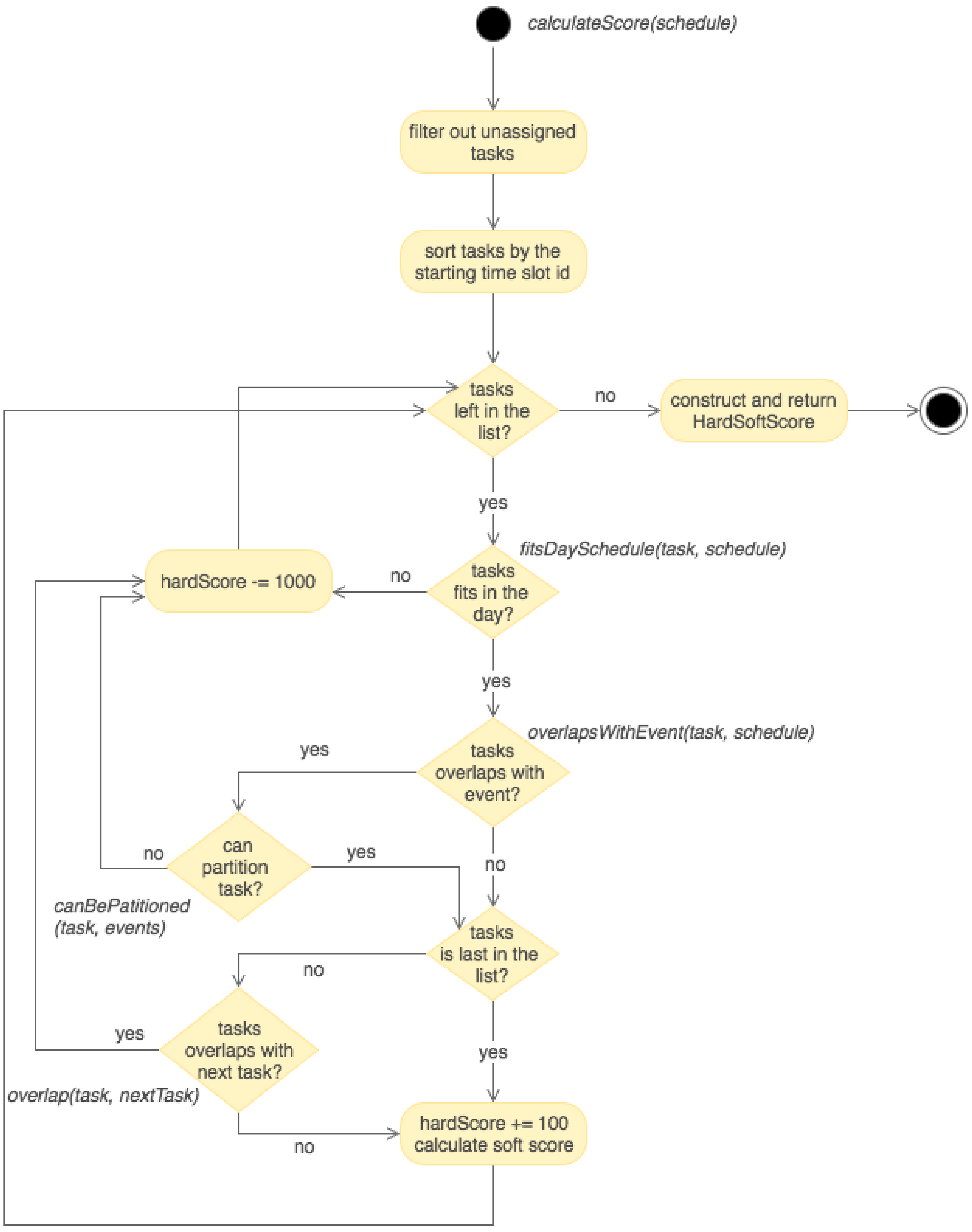

calculateScore(schedule)
filter out unassigned tasks
sort tasks by the starting time slot id
tasks left in the list?
no
construct and return HardSoftScore
yes
fitsDaySchedule(task, schedule)
tasks fits in the day?
no
hardScore -= 1000
yes
overlapsWithEvent(task, schedule)
tasks overlaps with event?
yes
can partition task?
no
yes
canBePatitioned (task, events)
no
tasks is last in the list?
no
tasks overlaps with next task?
yes
overlap(task, nextTask)
no
yes
hardScore += 100 calculate soft score

Figure 5.10: Hard constraints implemented in DayAssignmentScoreCalculator class

The SoftScoreCalculator class is responsible for calculating and returning the weighted sum of scores calculated by the currently defined rules for the task which is passed into the method calculateScore(task:Task):Int from DayAssignmentScoreCalculator. The calculateScore(...) function executes getScore(task:Task):Int method on each of the currently implemented rules and returns the weighted soft score which represents the 'optimality' of the assignment. Currently rules are defined statically in the SoftScoreCalculator.kt file, since DayAssignmentScoreCalculator is a class managed by OptaPlanner and it cannot be loaded to the Spring Context, thus the injection of the rules is not possible.

In the sequence diagram 9.4 the process of preparation and generation of schedule suggestion(s) for a day is shown, starting from the moment when user has submitted daily form to the point when the generated schedules are saved in the database to be retrieved by the client.

5.4.3 Controllers

The frontend module (client) communicates with the backend module (server) using the REST interface, implemented by the classes annotated with @RestController Spring Framework annotation and their methods. All rest controllers used in the application are defined in the package controllers in the backend module and marked with the Spring framework @Transactional annotation, because all of them save, retrieve or update the data in the database.

UserController provides mappings for getting, saving, and updating the information about the user of the application, such as user's settings, or getting the schedule suggestions generated for a day as well as saving and retrieving the chosen schedule.

DailyFormController provides mappings for getting and saving daily form information, it also serves as the entry point for the schedule generation, since submission of the daily form triggers the schedule generation process. Thus, the method for submitting the daily form includes the asynchronous function which calls the schedule generating functions and finally persists the suggested schedules when the execution of the solving algorithm has finished.

The java.util.concurrent.Executors is used to create a single-threaded executor (Executors.newSingleThreadExecutor()) to execute the requests for schedule generation in turn, even if the they arrived at the same time, preventing errors in the execution and unforeseen behaviour of the scheduling algorithm. The asynchronous method is shown in the listing 9.1 and the helper method for persisting the found schedules in the database with all required dependencies is provided in the listing 9.2. The sequence diagram presenting the process of schedule generation which starts in this controller is available in Appendix (fig. 9.4).

TaskController defines mappings for getting, updating, and adding tasks to the user's task list. The method for getting the tasks returns also the tasks which have been recently deleted or completed, methods for reverting the deletion or completion of the task are also provided to improve usability. The TaskRepository is used for managing the task persistence and provides methods for saving, updating, and retrieving the tasks.

SampleDataController was created for testing. It saves sample data to the database, so that further operation with the data can be conducted to test the behaviour of particular functions or the application in general.

5.4.4 Data Access Classes and DTOs

Data Access

For easier data access the interfaces extending the Spring Data[22] `CrudRepository<T, ID extends Serializable>` were created. The `CrudRepository` interface already provides the functionality for saving and deleting entities as well as finding entities by their id. The following `CrudRepository` extensions were created in the `data_access` package: `DailyFormInfoRepository`, `TaskRepository`, and `UserRepository`. Some methods for convenience of data retrieving were defined in these repositories. For example, `DailyFormInfoRepository` defines method which finds the daily form for the user and specific date: `findByOwnerAndDate(user: User, date: LocalDate): DailyFormInfo?`, which is quite useful, since we are mostly interested in the daily forms for today.

DTOs

The data transfer objects (DTOs) are used for transferring the information between client and server. In this application DTOs were defined when the information on server had different form than on client. Three DTO classes were defined in the package "DTOs" of the backend module:

- `CalendarEventDTO` to provide the fields corresponding to those of tasks in order to be able to represent both tasks and events in one schedule;
- `TaskDTO` for representing two different classes defined on server: one for a task generally, and another — for a task assigned to a schedule. There is only one type of task interface on client, so the `TaskDTO` was introduced to correspond to the task interface expected by client. Both `Task` and `AssignedTask` can be easily transformed to the `TaskDTO` by calling the `toDTO()` extension function on any instance of both classes;
- `ScheduleDTO` was introduced to reduce the number of attributes which the `Schedule` class has, thus sending less unnecessary information.

5.5 Data Model

For the persistence of the application data Hibernate as an implementation of Java Persistence API (JPA) was used. Therefore, the creation of tables with necessary columns, management of persisted objects (known as managed objects), and retrieving of objects from the database is done by Hibernate. Annotations for classes and sometimes their fields were used to communicate which classes need to be persisted (with @Entity annotation), which fields of the class are not persisted (`@Transient` annotation), and how the classes are connected with one another (using @OneToMany, @ManyToOne, and @OneToOne annotations).

Entities persisted in the database are: the users of the application, which represent a hub for all other entities, the schedules which were generated for every day (the selected option marked with flag "chosen"), all daily forms which user has filled in to have a possibility to create visualizations of productivity factors' variation over time, tasks which user has added with information about deletion and completion (tasks are never deleted from the database, since they might be useful),

[22] `http://projects.spring.io/spring-data/`

assigned tasks, which are created for each generated schedule, and events for the current day retrieved from the calendar. Thus, there is more information stored than is currently used by the features of DayO, but the data model was devised keeping in mind the possibility to extend the functionality of the application.

The entity-relationship diagram 5.11 represents the data model of the entities and their attributes stored in the database. The transient attributes are not shown on this diagram, since they are not persisted. IDs are almost always integers and automatically generated, except for CalendarEvent, which has a string id is taken from the event retrieved from Google Calendar.

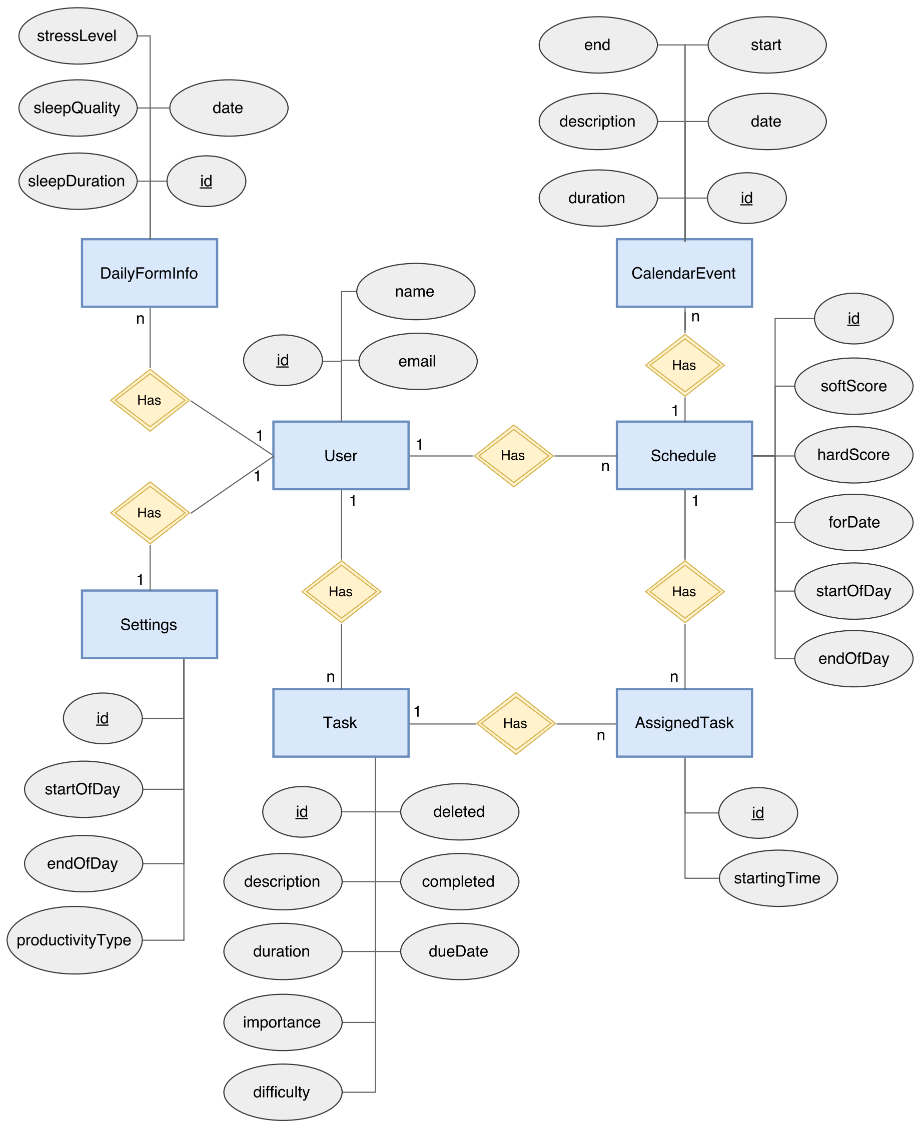


Figure 5.11: Entity-relationship model of the entities stored in the database

5.6 Incorporated APIs

The APIs used for implementation of DayO are described in this section, the reasons for choosing to use them are provided.

Google Calendar API

Google Calendar API[23] is used to retrieve the events for a day from user's calendar. The API was chosen because Google Calendar is widely used and all participants of the user evaluation had Google accounts.

Google OAuth 2.0 Library

In order to access information from Google Calendar API, OAuth 2.0 authentication had to be implemented. As the result of authentication, the application receives user-specific token, which is then used to access the Google Calendar API.

Spring Security

For controlling the access to the application and allowing multiple users to use it independently, it was considered to use Google as the authentication authority, but we decided against it, since it required more effort and was not necessary. Instead, Spring Security[24] was used for simple form-based user authentication. The valid user credentials have been included in the source code and kept in memory during runtime.

OptaPlanner

Different APIs for solving CSP problem were considered; OptaPlanner [2] was chosen for the compatibility with Java and easy configuration of the solver with the .xml file. The possibility to use specifically annotated Java classes for problem definition and detailed documentation of the API with many executable examples made working with this API much easier than with other CSP solver libraries.

In the `solver-config.xml` file the configuration of the planning engine provided by OptaPlanner as well as the classes defining the planning problem and solution are described. The detailed description of the classes defining the planning problem is provided in section 5.4.1 with visualization in figure 5.9. The configuration of the solver is provided in the listing 9.6.

Hibernate

To avoid manual set up of the database, afford the flexibility of choice of the database, and in order to work with database tables in object-oriented way Hibernate ORM[25] has been choosen as

[23]`https://developers.google.com/google-apps/calendar/`
[24]`https://projects.spring.io/spring-security/`
[25]`http://hibernate.org/orm/`

an implementation of the Java Persistence API (JPA) specification. The data model visualization and description of used annotations is available in section 5.5. Hibernate DDL generation has been used to automatically create database schema for use with persistence classes. As the result of this flexibility, we were able to use H2 in-memory database for local testing, and switch to PostgreSQL when the application was deployed.

Spring and Spring Boot

Spring Boot framework was used to create application that exposes REST endpoints for frontend use, contains embedded Application Server (Tomcat) and can be deployed as simple jar file. Spring (Dependency Enjection Framework) was used to wire application classes together and to enable persistence and transaction aspects of the application.

Other APIs

Additionally, Java 8 Date and Time API was used to represent date and time values; Kotlin Standard Library was used for collection manipulation.

Chapter 6

Evaluation

In this chapter the method of the user evaluation, the results of the conducted study, and suggestions for further improvements are described. The user evaluation of DayO was conducted with three participants full-time and one participant part-time during one week. The goal of the user evaluation was to estimate the accuracy of generated schedules as well as to gather feedback about the application and the approach of automatic scheduling of working tasks. In order to conduct the study, the application was deployed on Heroku[1] using the Postgres database[2].

6.1 Evaluation Method

The user evaluation was conducted as one-week long diary study. During one week participants of the study were asked to use DayO for generating the workday schedule from tasks users defined for themselves. Every participant received an email with credentials for logging into the application, the links to the questionnaires, and detailed instructions for the study, in which the necessary initial set up and the daily routine was described. All questionnaires were created and hosted using Google Forms[3].

Every participant was asked to use DayO for five consecutive workdays, filling in the daily questionnaires once after the schedule was chosen and once after at the end of the workday. The study was concluded by one final questionnaire. The questions used for each of the questionnaires are provided in the table 6.1. Data used for evaluation is taken from the production database and answers for the questionnaires. Personal information about the tasks or participants is not used for analysis.

Participants. Four people have agreed to participate in the user evaluation, 3 male. Participants for the study were recruited fusing personal connections. At the time of the evaluation all participants of the study were 100% employed. There was no payment promised for the participation in the user evaluation. The age of participants ranges between 25 and 35 years, all participants have at least one Bachelor's degree.

[1] `https://www.heroku.com/`
[2] `https://www.heroku.com/postgres`
[3] `https://www.google.com/forms/about/`

Table 6.1: Questionnaires used during user evaluation

Questionnaire	**Question**	**Answer Type**
Morning Questionnaire	1. Why did you select the option over the other ones?	text
	2. How good is the schedule you have chosen today?	scale: 1 — 10
	3. Is there something what can be improved or changed in the schedule, if so, what?	text
	4. Do you want to make any other comment? (optional)	text
End of Workday Questionnaire	1. How productive did you feel during the day on average?	scale: 1 — 10
	2. What aspects influenced your productivity?	text
	3. If there were tasks which you did not complete, what was the reason for not completing them?	text
	4. How much did you follow the schedule? (*options: completely, mostly, partially, not at all*)	multiple choice
	5. Do you want to make any other comment? (optional)	text
End of Study Questionnaire	1. How accurate were the suggested schedules for the day?	scale: 1 — 10
	2. Please explain what made schedule accurate and/or inaccurate?	text
	3. Is there something you liked or disliked about DayO? If so, what?	text
	4. Is there anything [feature, information] you would like to be included to make DayO more valuable and useful to you? If so, what?	text
	5. Do you think that DayO (or an improved version of it) could be used for everyday planning? If not, why?	text
	If you have any comments, ideas or suggestions about the project, the tool, or the study, please share them here:	text

6.2 Results

The conclusions about results of the evaluation were made based on analysis of the data stored in the database during evaluation and qualitative analysis of the responses to the questionnaires which were filled in during the study. First the quantitative results of the evaluation are described, followed by the qualitative results of the questionnaires analysis. Specific suggestions of study participants are discussed in section 6.2.3.

6.2.1 Quantitative Results

From analysis of the data stored in the database during the user evaluation we have made conclusions about the quantity of information added during the study, the accuracy of the suggested schedules, and the task completion rate.

Tasks. During the evaluation which took one working week our study participants have entered 101 task to the database, 11 of which were afterwards deleted. From these tasks 84 (83%) were assigned to the chosen schedule at least once. From all tasks about 70 were completed. From all not deleted tasks only 20 (about 20%) were not completed during evaluation. There were no tasks which were both deleted and completed at the same time. Thus, the task completion rate was about 70%. From 84 tasks which were assigned to the chosen schedule at least once 66 tasks (79.5%) were completed, from which 53 (63%) were marked as completed on the same day as they were assigned.

Table 6.2: Tasks statistics

Tasks	Total	min	max	Average
added	101	15	32	25.25
deleted	11	0	6	2.75
not completed	31	4	14	7.75
assigned at least once	84	13	25	21
assigned and completed the same day	53	6	20	13.25

The average estimated task duration was 75.6 minutes (about 1 hour and 15 minutes). Most of the tasks (about 70%) had estimated duration of less then 1 hour, and only 17.8% had estimated duration of more than 2 hours.

Events. During the evaluation week 30 events planned during the workday were synchronized from the Google Calendar. This makes an average of 1.5 events per user during one day. Some users did not have any events during the day on some of days, which seems unlikely, since participants were asked to enter their lunch and other meetings to the calendar prior to the schedule generation. We assume that someone might have forgotten to follow instructions regarding this point, therefore the number of events might have been higher.

Table 6.3: Events statistics

Events	Total	min	max	Average
during one week	30	6	10	7.5
per participant and day	—	0	3	1.5

Schedules Accuracy Analysis. The total of 52 schedules were generated for all participants during the evaluation week, with the minimum of 10 and maximum of 15 schedules per participant during the evaluation week . The number of schedules per user and day varies between 1 option in three cases (15%) and 3 options in most of the cases (75%). There were 20 schedules chosen — five for each participant meaning that all participants have generated schedules on all days of the study.

Every schedule has a hard score and soft score based on which schedules can be compared. Hard score represents the number of the tasks which were assigned and soft score represents how well the current schedule fulfils the productivity and task-related constraints. Hard score almost always remained stable for all options generated for a particular day. Since soft score is there only to compare different options when choosing the best solution to the problem, the absolute value of the soft score is not important. Generally, the option with larger soft score would be considered as more desirable (event if both scores are negative).

In the figure 6.1 the distribution of the scores for all participants and all generated options is shown. The data is not grouped in any way and presents solely the distribution of the score. We can see that there is a negative correlation between higher hard score and soft score. The reason being that the scores given by the rules are distributed between -100 and 100 and bad sleep quality or high stress level are pulling the score down.

For example, when user has entered sleep duration of less than 7 hours and/or sleep quality of less then or equal to 'BAD', then all scores given by the tiredness rule are negative (much smaller for complex tasks than for easy ones). Thus, more tasks assigned on such day would lead to lower soft score. To counterbalance this tendency the 100 points of the hard score are given for each assigned task, making the reason for negative correlation obvious. Since all schedules are generated using the same rules and possible assignments are compared based on both hard and soft score, the suggestions are at most three best solutions found to the planning problem, even though the soft score might be negative for all of them.

On the chart 6.1 scores of chosen solutions are represented with orange point and all other scores — with gray. Mostly, when generating options for one day and user, solver would find schedules with similar number of tasks, and thus similar hard score. On the chart 6.1 we can see that orange dots in most of the cases are higher then gray ones with the same hard score. This means that in most cases the schedules with the best scores were chosen.

The figure 9.5 provides an overview of all schedules and their respective scores which were generated during the user evaluation for the study participants. From that table it is clear that from 20 sets of options generated for the study participants during the evaluation week only in 4 cases (20%) not the best option was chosen (once the best option differed from the chosen by only one soft score point, thus two options were almost equivalent), 3 of the choices were trivial — choosing from one option, the rest of the choices (13) were non-trivial and corresponded to the best option. Thus, the accuracy of the best suggestion is 65%, meaning that the option with highest score, which would be preferred by the algorithm, was chosen by the study participants in 13 non-trivial cases out of 20.

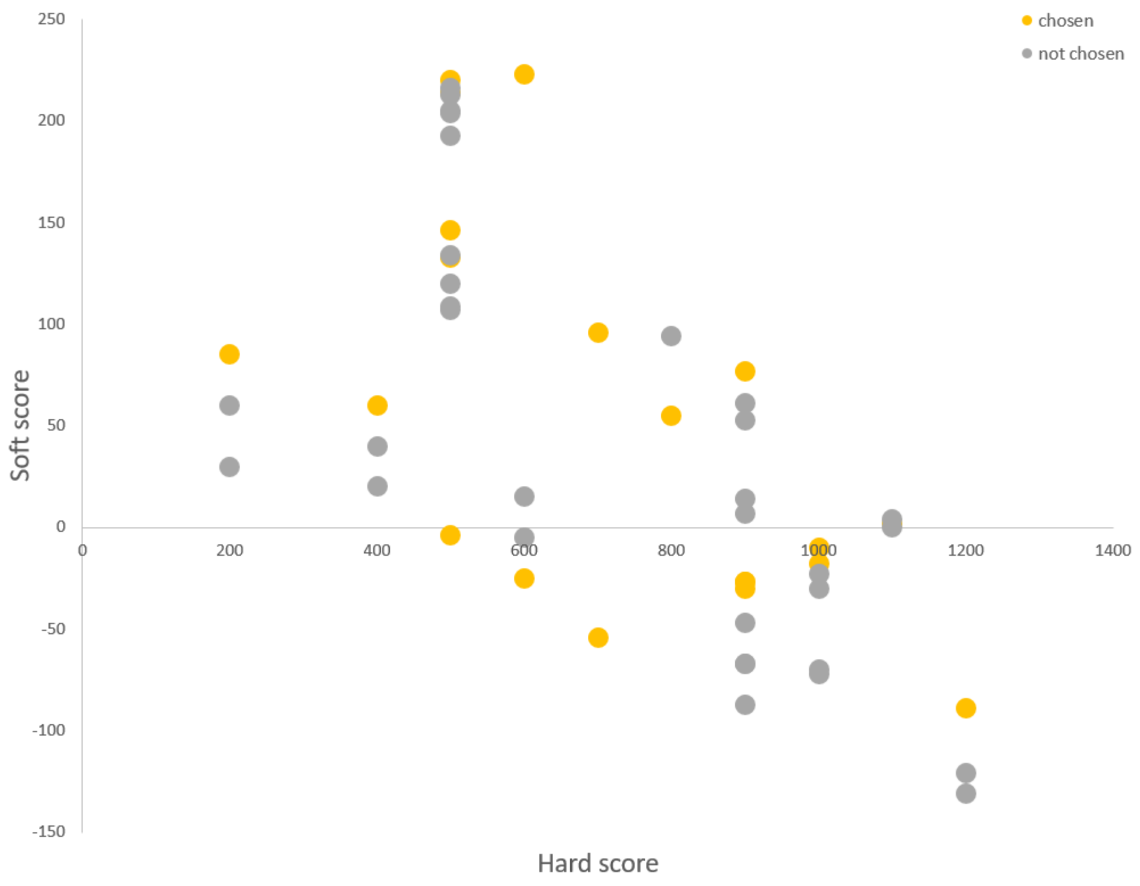


Figure 6.1: Hard and soft scores of all generated schedules for study participants

The results of the quantitative analysis of the data from the production database show that most of the tasks (70%) were completed, 63% of assigned tasks were marked as completed on the same day as they were assigned, and that in 65% of the non-trivial cases (two or three options to choose) users have preferred the schedule with the best score.

6.2.2 Qualitative Results

This results are based on the analysis of the questionnaires which participants have filled in during the evaluation week. There were three different questionnaires: daily morning (21 answers) and end of workday (19 answers) questionnaire were supposed to be filled in every day, and final questionnaire (4 answers) which was filled in at the end of the study. The questions asked in each of these questionnaires are presented in the table 6.1.

Satisfaction with the Schedule Choice

Right after the schedule was selected participants were asked to assess the quality of the schedule they have chosen, the chart with responses is presented in figure 6.2. Most of the users (85.5%) were satisfied with their choice and selected the score of 6 or more points (of maximum 10). Almost 20% of participants were very satisfied with the selected schedule giving 9 (4.8%) and 10 (14.3%) points.

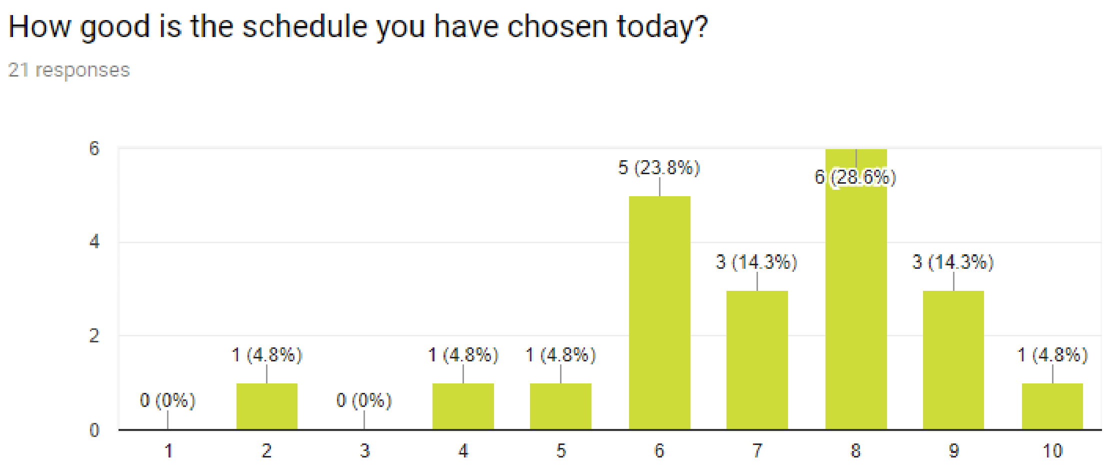


Figure 6.2: The perceived quality of the chosen schedule assessed after the schedule was chosen

The reasons for choosing the option were:

- the tasks with high priority and/or difficulty were more present in the chosen schedule and/or planned before other ones
  - S1: "It had most important tasks planned first"
  - S2: "Because i saw a challenging task more present on the option"
  - S4: "...difficult + important stuff first"
  - S4: "it suggested to do the difficult and most important tasks in the morning"
- the schedule suited user's expectations regarding choice and ordering of specific tasks
  - S1: "It had the task which I wanted to do today."
  - S3: "It presented the tasks of interest in the most suitable order"
  - S4: "it was absolutely perfect, exactly what I planned to do (in the same order, etc.)"
- the schedule was without blank spaces
  - S2: "The time is used completely , without big holes in between"
- the easy tasks were more evenly spread during the day

  - S1: “It had easy tasks nicely spread over the day”
- some schedules were chosen because there was no alternative provided

Following Chosen Schedule. On the chart 6.3 the answers of participants regarding following the chosen schedule are shown. On most of the days people followed the schedule at least partially, on very few occasions people did not follow the schedule at all, and even more seldom followed the schedule entirely.

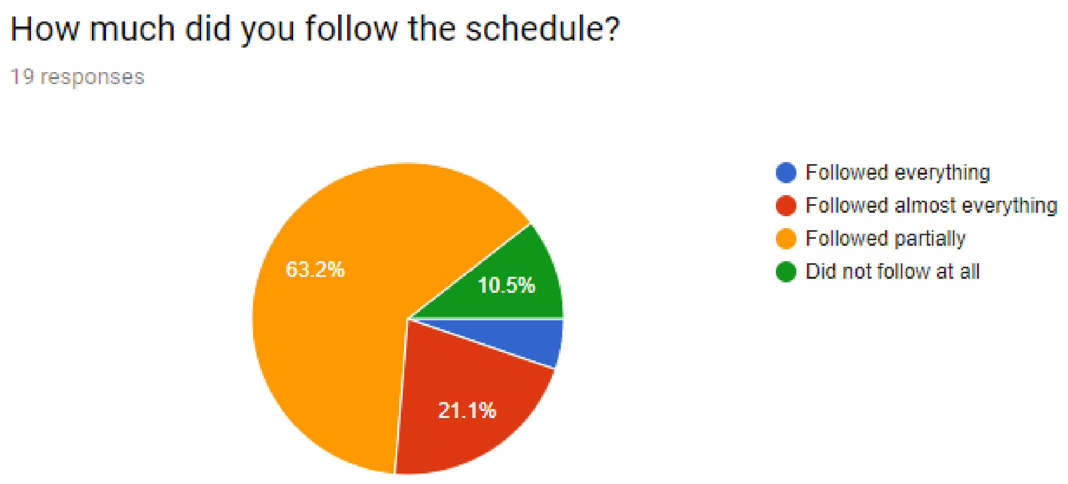


Figure 6.3: Statistics of following schedule during the day assessed in the evening of the same day

Perceived Productivity and its Factors

At the end of each workday participants were asked to fill in the ’End of Workday Questionnaire’ in which users were asked about their perceived productivity during that day, the possible reasons for it, and how closely they were following the chosen schedule.

Most of the users reported to feel more productive than average during the workday, choosing between 6 and 8 point out of 10 in 79% of cases. Very few times (2) people reported to be very unproductive, and never considered themselves to be very productive (giving 9 or 10 points). The distribution of the perceived productivity is shown on chart 6.4.

The following factors which might have influenced participants productivity were named:

- feeling tired or sleepy due to lack of sleep or bad quality of sleep
  - S1: “I didn’t sleep enough and was stressed by one urgent task that required my attention.”
  - S2: “I slept bad, that’s why i was several times pretty sleepy”

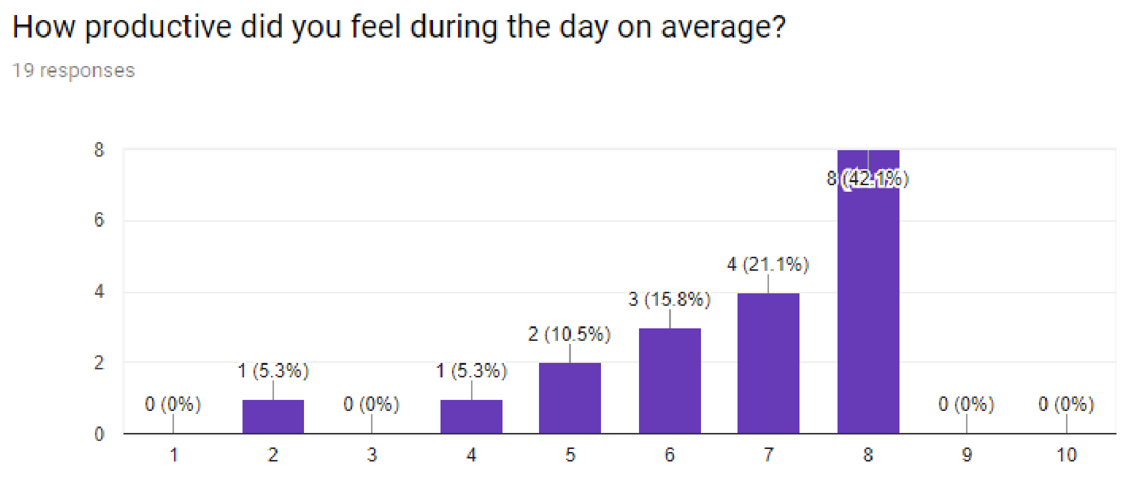


Figure 6.4: The perceived productivity assessed at the end of workday

  - S3: "General feeling of tiredness and frustration "
  - S4: "tiredness - the morning was super good, the afternoon not good at all..."
- stressful work situation (environment at work or related to tasks)
  - S1: "... lack of fresh air in the office."
  - S2: "The pressure to be ready with the tasks by the end of the day "
  - S3: "Too many people in the room"
  - S4: "didn't finish anything I wanted, interruptions, unexpected stuff"
- task-specific qualities
  - S3: "Challenging tasks in programming"
  - S3: "Interesting tasks"

Interestingly, on the days when participants were naming some negative factors influencing their productivity the productivity was was estimated score lower, and when some positive factors were named participants answered with higher score to the question about their productivity.

Reasons for not Completing Planned Tasks. Different reasons for not completing task(s) on the chosen schedule were given:

1. The most frequently named reason — unplanned urgent tasks (which were not included in the schedule or arose after the schedule was generated)
   (a) S1: "Unplanned stuff came along and I had to switch to it."
   (b) S2: "Because there were many tasks which came in-between. So i needed to postpone the tasks i planed originally."

(c) S3: "The task that was not completed was tricky, moreover there was an importnat task which was not in the schedule, that was more urgent to start with"

(d) S4: "lack of focus, context swicthes, unexpected other tasks"

2. task execution took more time than was expected
   (a) S3: "Other tasks took longer than expected"
   (b) S4: "another task took longer than planned"
3. Lack of focus and problems with accomplishing the tasks
   (a) S1: "I was lazy and didn't feel like doing them."
   (b) S2: "I just didn't want to do it, because it was hard, i do better easier tasks instead:)"
   (c) S4: "lack of focus, unexpected problems"
4. Too many tasks planned in the schedule.

Summary of Qualitative Results According to the analysis of the questionnaires we can make following conclusions about the choice of the schedules, perceived productivity, and reasons for not completing planned tasks:

- Schedules were chosen based on how much schedule corresponded to the expectation of the user regarding the tasks which should be included and the ordering of the tasks.
- Importance and difficulty of the planned tasks were often deciding factors in choosing some schedule. For example, schedule with more difficult and/or important tasks would be preferred to the one with fewer such tasks. Some participants voiced clear preference to have important and/or difficult tasks planned in the morning.
- Among factors which influenced the productivity during the day, tiredness or lack of sleep were named most often, followed by the external pressure and other factors of the working environment. Interruptions and unplanned tasks were also named as aspects influencing productivity.
- If some of the planned tasks were not completed, it was attributed to unplanned tasks arising, important tasks not appearing in the suggestions, tasks taking more time than was originally estimated, and lack of focus or motivation to complete these tasks.

6.2.3 User Feedback

We have received quite a few comments about the features which users have enjoyed, like synchronization of events from Google Calendar

- S4:"it's nice that it always fetches the newest calendar entries from google cal",

the design of the user interface and guidance provided by the application

- S1: "I liked the colors, the representation of the tasks on the day view together with fixed appointments - it was very helpful. It was clear what the application expects from me at any given moment, so I didn't feel lost.",
- S2: "I like the simplicity - at the left size the plan, at the right side the tasks.",

- S3: "... I liked that it has minimalistic design and clear indicators of task importance and difficulty (via colors and stars)..."),
- S4:"it's super nice that I get guided a bit...",

the general idea of automatic schedule generation was mostly perceived as useful, especially having multiple suggestions of day schedule

- S3: "It is important that the application offers different schedule options, so that one can select the most suitable one."

On the other hand some things were perceived as annoying or unpleasant. The time it tool to generate the suggestions was perceived as too long

- S1: "I didn't like long waiting time (1 minute) to generate the schedule, I was getting distracted and reading something on the internet while waiting...",

the assignment of tasks to particular time was perceived as rigid and stressful

- S1: "I feel like it's a bit too rigid. If I would be really super concentrated to do all my tasks - I would probably do it all and appreciate that the tasks were planned for particular times during the day. But I cannot imagine myself working in such mode for a long time and without a very, very good reason to do so."

For some participants it was challenging to estimate duration of tasks

- S1: "It's very difficult, if not impossible, to be realistic about expected task duration...",
- S3: "sometimes it is difficult to say, how long a task lasts."

Some comments were about the application not suggesting important tasks or tasks with close deadline in any of the provided options, which certainly will need to be improved.

Following suggestions were made regarding DayO and the concept of automated day planning:

- It was suggested that the task urgency would have a higher weight and tasks with close deadline would be planned before other tasks as well as very important tasks.
- Frequently the suggestion about having breaks between or during the longer task has come up in the comments and often in connection with remarks that there were too many tasks planned for the day.
- Make it possible to manipulate the schedule after it was selected to add some urgent tasks or the important tasks which were not included.
- Grouping the tasks by some criteria and including the dependencies between the tasks (some tasks need to be done before in order to begin with execution of the task).
- Using machine-learning to improve the suggestions and 'tailor' schedules depending on previously made choices and learned preferences of the user.

6.3 Further Improvements

According to the comments of study participants, the concept of the automatic scheduling could be improved by giving more weight to the tasks with the due date, since sometimes the tasks with close deadline were not appearing in the suggestions. Some participants of the study expected to see the important and difficult tasks planned before other tasks during the day and such behavior was not programmed in our rules based on which the optimization was made. The dependencies between tasks were also not part of the current implementation which made it impossible for the application to know which tasks should be completed before others could be worked on. It was an assumption we made that all the tasks in the list are independent and theoretically can be completed at any time, which certainly falls short of reality.

The rigidity of the schedule poses a usability problem, since many times it was reported by the study participants that they were not able to complete the tasks planned for a day due to unplanned tasks arising after the schedule was chosen or because suggestions did not include some tasks users considered the most important. To improve usability, functionality for schedule manipulation after the schedule for a day was chosen will need to be included, so that some tasks can be removed or added to the schedule or moved to the other time, if necessary.

The feature for including the breaks between or during longer tasks was also requested by the study participants, otherwise there might be too many things planned in the schedule and having no time for breaks may cause stress, leading to the dissatisfaction with the tool and lower productivity. This feature was considered during planning of the project, but was not implemented due to lack of time. It was thought, that in the settings users could specify the frequency and duration of breaks, and whether they want to make breaks during the task execution. Then breaks would be planned around and during tasks according to the settings, and then presented in the schedule.

It was suggested by some study participants that the ability of the algorithm to 'learn' about the connections and dependencies between tasks could be used to improve the planning by putting the tasks which belong together closer in the schedule or even guessing which task(s) should be executed before the other task can be planned. One participant has also suggested to use machine learning to improve planning. Together with the flexible schedule machine learning approach might improve the schedule suggestion over time adapting it to the user's preferences by learning the habits and creating new rules for schedule optimization dynamically.

6.4 Limitations and Conclusions

The user evaluation study shown that DayO with some improvements described in 6.3 (or other tool for automatic task scheduling) could be used for workday planning and that combined view of the fixed appointments and planned tasks was valuable as representation of a workday. The schedules generated by the algorithm were of high accuracy, since most of the time (65% of cases) users have chosen the option with the highest score, which is considered the best option found by the algorithm. Designing the user evaluation as a diary study allowed us to understand how satisfied participants were with schedule suggestions and how productive they felt every day. On average users were quite satisfied with the chosen schedule on daily basis, in 85.5% of the cases giving 6 point or more out of 10, at the end of the study the average of the reported satisfaction with the suggested schedule options was equal to 5.75 (one participant giving only 3 points out of 10, and all other — 5 or more).

Limitations of this study are the duration of the study and the number of participants. The fact that all participants were employed in the field of computer science could also be seen as a limitation of this study, since DayO is designed to be used for planning workday of knowledge workers. Longer evaluation with more participants from different fields could provide more insights into constraint-based workday planning and help determine the usefulness of automated task scheduling.

Chapter 7

Discussion

In this chapter the we discuss the importance of the DayO concept and application in relation to available solutions, reveal some limitations of the solution described in this thesis, and provide suggestions for improving the concept and DayO application as well as topics for the future research.

7.1 Discussion

For all knowledge workers a question arises many times: what should I work on right now? Being able to effectively work during the day while maintaining reasonable (not too high) level of stress and completing the tasks at hand depends on finding a good answer to this question every time. Knowledge workers often work on multiple projects at the same time and have multiple responsibilities. To plan a workday schedule, many different task parameters (such as deadlines, priority, and difficulty) as well as amount of available time need to be considered. The more tasks on the task list — the more complicated planning gets. Given that it is most probably not possible to complete all tasks during one day, the tasks to be completed today need to be first chosen, and then ordered based on some criteria. Meetings and other appointments during the day make planning effort even bigger.

To answer the research question we have reviewed existing literature on personal productivity (detailed description is provided in section 3.3). The research revealed many factors which might influence knowledge worker's productivity, ranging from extraneous factors (such as information overload and work environment) to the factors which could be changed willingly (such as sleep duration and quality). From all identified factors we have chosen those with clearest definition and impact on effectiveness of performing cognitive tasks, such as sleep, stress, and productivity type (chronotype).

The DayO concept and application (detailed description provided in chapter 4 and 5 respectively) allows users to automatically generate schedule for a day from the tasks they enter in the application. The schedule is generated taking into account the time constraints (such as start and end of the workday, calendar events), task characteristics (priority, difficulty and due date) and the productivity factors which user enters daily (sleep quality, sleep duration and stress level) and in settings (productivity type: morning, neither, or evening). The application was tested locally and during one-week user evaluation which has shown that the schedule suggestions were of high accuracy and that users perceived automatically generated schedules as useful.

Compared to other similar applications and concepts, which are described in detail in section 3.2, task is optimizing the schedule taking into account user's productivity constraints additionally to task parameters and timing constraints. Before generating the suggestions for workday plan, DayO retrieves the events planned for a day from user's calendar to assign the tasks around existing events. After entering the information about daily productivity constraints (sleep and stress) and choosing one of the suggested schedule options, user sees the unified representation of tasks and events planned for a day.

However, there are some limitations of the approach suggested in this thesis. For example, the assumption that all tasks on the task list are independent from one another and can be accomplished any time does not correspond to most real-world situations because tasks may have dependencies, making it impossible to plan some tasks before other ones were completed. Some tasks also may logically belong together and it might be better to plan them closer to one another to avoid switching of the context. Implementation of this functionality would require extension of the current domain model and more input from the user; usability of the tool might suffer from requiring more input and introducing additional complexity for the user.

In the current implementation the schedule cannot be changed after it was chosen and this limitation was considered as drawback by some users, since sometimes, due to unplanned tasks and interruptions or absence of some important tasks in the suggested schedules, the chosen schedule would not anymore correspond to the reality of the day. This makes schedule almost useless on days when such events occur and causes additional stress for the user to consolidate reality with the existing schedule.

Currently the optimization of the schedule is done using the rules concerning different factors for choosing and positioning the tasks. Even though all rules have weights and it would be possible to make some rules more 'important' than others by manipulating them, in the current implementation all rules have the same weight. Assigning of different weights to the rules could be used to manipulate the relative influence of each rule on the scheduling decision.

The choice of the library for the solving of the CSP imposed certain restrictions on the problem design and solution strategies. The solution of the CSP can take quite long time when there are many different options to consider. Due to the nature of CSP problems (being NP-hard such a problem might have no optimal solution and exploration of the entire solution space takes too long) and the implementation of OptaPlanner [2], the solving must be restricted to some time and there is no guarantee that the solution found during this time is optimal.

To improve the quality of the suggestions made by the algorithm and shorten the time of execution, more computational power is required. For example, when tested locally, the application provided reasonable suggestions with few unoccupied time slots, producing schedules corresponding to expectations of what needed to be planned. However, when the application was deployed on Heroku[1] generated schedules would not always correspond to expectations and would often have blank spaces, even though there was a evident possibility to fill them. Ideally, the algorithm would be expected to fill all available free space with tasks, since each task assignment was additionally rewarded.

Waiting time for suggestions to be provided (a bit more than one minute) was experienced by users as too long, and led in few occasions to inconsistent state of the system, which later needed to be fixed, costing the study participants additional time. The long waiting time is clearly a usability issue, which would need to be solved in the future.

[1]https://www.heroku.com/

7.2 Future Work

To improve the quality of schedule suggestions, testing of different weights for existing rules would need to be performed, having large enough sample of tasks and clearly distinguishable use cases with documented expectations and outcomes. Even better, if different rule weights could be tested with users to understand which set of weights produces the most accurate suggestions. The question of defining and choosing the factors which influence knowledge worker productivity could be researched further by conducting a diary study, during which knowledge workers would be asked to reflect on their productivity and factors which influenced it. The results of such research could help improving the rules for schedule optimization.

Usability of DayO could be improved by providing the possibility to change the chosen schedule for a day, so that the schedule could be adapted to changes in tasks, priorities and environment. It should be possible to add previously unplanned tasks to existing schedule, remove tasks, and manually reschedule them. The question remains open whether these changes are just one-time adjustments or should be incorporated into behavior of the algorithm. If the later is desirable, then some type of machine learning would need to be added to the existing algorithm. With machine learning it could be also possible to learn the habits of the user and evolve the rules based on which the optimization is made.

Adding breaks between or during tasks was one of the features requested by the participants of the user evaluation. This feature was considered during the planning of the project, but was later left out in order to concentrate on improving the algorithm for day planning. The concept of the break feature goes as follows: on the settings screen users will be asked how often they want to make breaks and how long the breaks should be, optionally breaks can also be configured to be inserted during tasks which exceed some length (for example, if the task is longer than 2 hours, then the breaks should be inserted); the breaks feature would be optional, so that user can turn it on or off.

Using pomodoro technique[2] to interchange the focused periods with breaks was considered as another option for implementing the break feature. User would be asked in the settings if he wants to use the pomodoro technique and once it's on, the values for desired focus time and break lengths would be requested, otherwise the default suggestion of 25/5 with longer break of 15 minutes would be saved. User could then be reminded whenever the break or next focus session starts. The question of how to notify user about making break or starting to work on the next focused session from the web application without overwhelming the user by continuous notifications remains to be answered.

According to feedback of some participants, they expected to see the tasks with closest deadlines and/or highest priority and/or complexity to be planned before all other tasks. Such rule is not implemented in the current version and could be added to the future version of DayO.

Since the application gathers information about user's sleep and stress level on daily basis, visualization of this data could be presented to the user. Such visualizations could provide insights and raise awareness of sleep behavior and stress. This feature was considered during the planning of DayO, but not implemented due to lack of time. If users would also be asked to assess their productivity during the day (as participants of the study were asked during evaluation), this information could be also visualized. Moreover, the relation between the productivity factors (sleep and stress) and the perceived level of productivity could be then assessed over time, providing insights into user's personal productivity factors.

[2]`https://en.wikipedia.org/wiki/Pomodoro_Technique`

Chapter 8

Conclusions

8.1 Summary

To alleviate the effort of workday planning and provide additional value by choosing the tasks for a day which best correspond to the current productivity level and position them according to productivity type, we have created a concept of automatic task scheduling based on set of constraints. To test this concept we have implemented DayO application.

The evaluation of DayO with users has shown that the algorithm for task scheduling provides quite accurate suggestions: in 65% of the cases the choice made by users corresponded to the option with the highest score, as rated by the algorithm. Study participants were on average quite satisfied with the schedule they have chosen for a day, giving 6 or more points out of 10 in 85% of the cases. User study also revealed the possibilities for further improvement of the DayO application and the concept of automatic task scheduling.

In future work the constraints of knowledge workers' productivity could be researched further by conducting a study in which knowledge workers would be asked to assess the level of their productivity on daily basis and reflect on the factors which might have influenced it. It would be also useful to ask the study participants about the factors influencing the choice of tasks to work on during the day. The results of such study could be used to improve the rules currently used for optimizing the schedule. To better understand and analyse the DayO application, longer user evaluation could be helpful.

Usefulness of the DayO could be improved by allowing users to manipulate the chosen schedule by adding, removing and moving the tasks. Breaks feature can be added to improve the schedule by adding breaks between and possibly during longer tasks. The information gathered about the productivity can be visualized to increase awareness of sleep behavior and stress level. Additionally, completed tasks' metrics and visualizations could be provided to motivate the user to complete the tasks scheduled for the day.

8.2 Contributions of This Thesis

1. The available literature on factors influencing the productivity of knowledge workers is reviewed in this thesis, providing an overview of aspects bringing positive or negative impact on effectiveness of solving work-related tasks.

2. Based on the productivity constraints, chosen from the variety of factors influencing knowledge worker's productivity, the concept of constraint-based workday planning was designed. The constraints include timing constraints (start and end of the workday, fixed appointments planned for a day), task-related constraints (tasks are prioritized based on their importance and urgency), and productivity constraints (sleep quality, sleep duration and stress level, filled in daily; productivity type, specified once in the user settings).

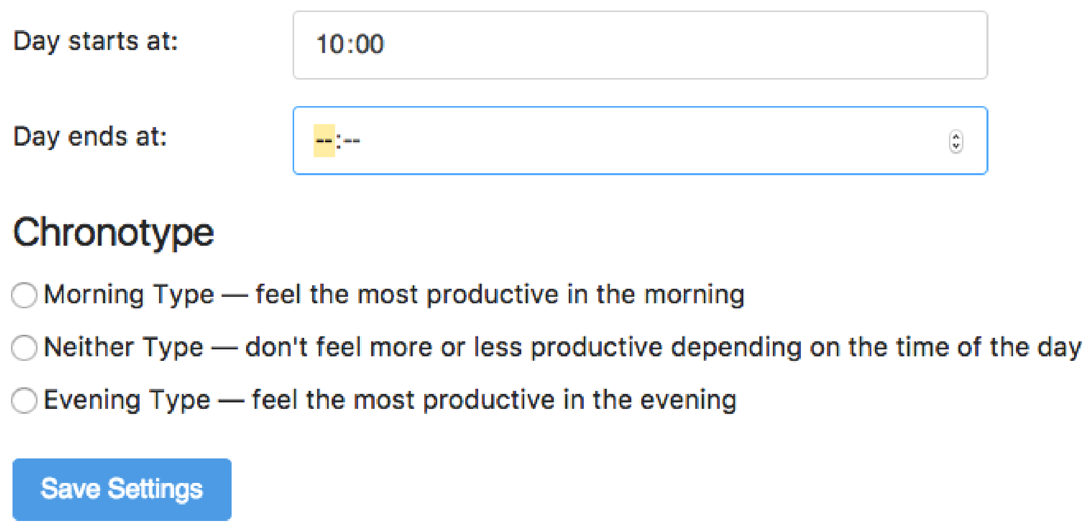
3. This thesis presents the DayO application developed to implement the concept of constraint-based planning described above. DayO provides simple interface for managing the task list and entering information about the productivity factors, and clear navigation for guiding the user through the steps required to generate schedule suggestions and choose the most appropriate one.

4. The concept and the DayO application were tested with knowledge workers during one-week diary study, which provided insights into the possibilities to improve the existing approach and usability of the application. The analysis of the data from the database and questionnaires filled in during the study has shown that automatically generated schedules were of high accuracy, and the possibility to have tasks automatically scheduled was perceived as useful.

Chapter 9

Appendix

9.1 Additional UI Screens

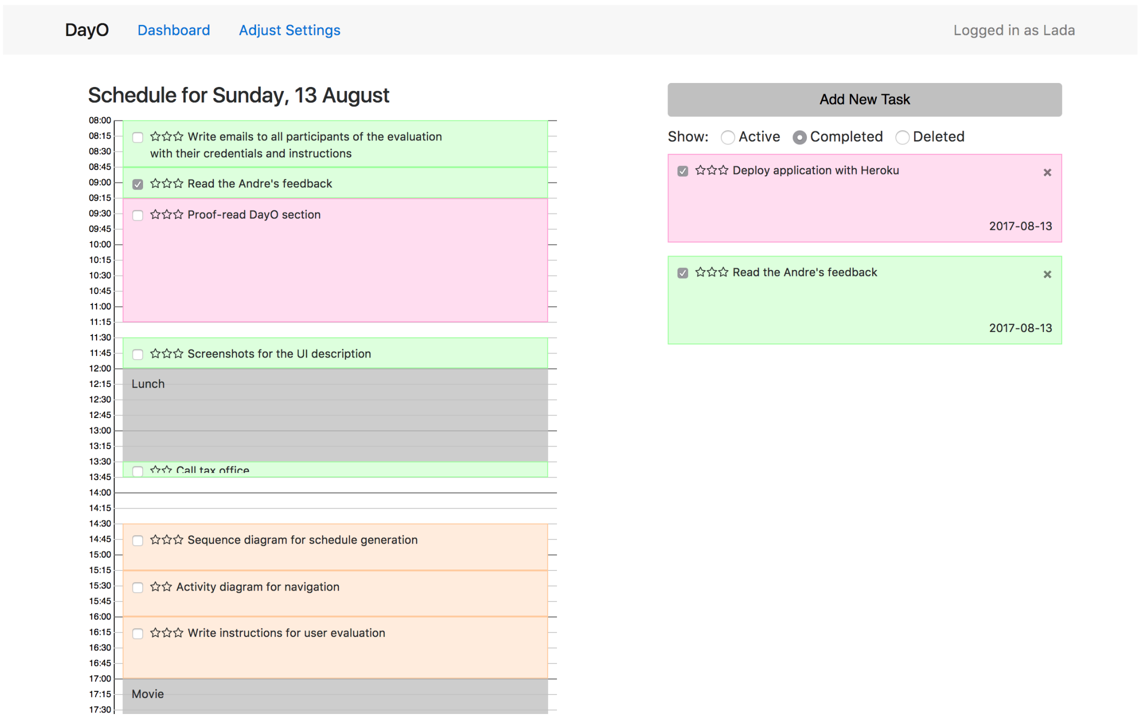


Day starts at:	10:00
Day ends at:	--:--

Chronotype

- ◯ Morning Type — feel the most productive in the morning
- ◯ Neither Type — don't feel more or less productive depending on the time of the day
- ◯ Evening Type — feel the most productive in the evening

Figure 9.1: Settings form



Schedule for Sunday, 13 August

- ☐ ☆☆☆ Write emails to all participants of the evaluation with their credentials and instructions
- ✓ ☆☆☆ Read the Andre's feedback
- ☐ ☆☆☆ Proof-read DayO section
- ☐ ☆☆☆ Screenshots for the UI description
- Lunch
- ☐ ☆☆ Call tax office
- ☐ ☆☆☆ Sequence diagram for schedule generation
- ☐ ☆☆ Activity diagram for navigation
- ☐ ☆☆☆ Write instructions for user evaluation
- Movie

- ✓ ☆☆☆ Deploy application with Heroku × 2017-08-13
- ✓ ☆☆☆ Read the Andre's feedback × 2017-08-13

Figure 9.2: Completing and viewing completed tasks

9.2 Code Listings

9.2.1 CSP Solving Execution

```
CompletableFuture.runAsync(Runnable {
   txOps.execute {
      val userInner = userRepo.findById(userId)!!
      urgencyRule.recalculateWorkingDays(today)

      // when daily form submitted, algorithm for schedule generation is started
      // events are synchronized daily and passed to the schedule generator
      val events = calSync.getCalendarEvents(userInner)
      // rules need to be initialized!
      rules.forEach { it.init(userInner, today) }
      // generate and save the schedules for the further use
      val schedules = scheduleGenerator
                          .generateScheduleOptions(userInner, today, events)

      persistSchedules(schedules, userInner)
   }
}, executor).exceptionally {
      logger.error(it, {"Exception while asynchronously generating schedules"})
      null
   }
}
```

Listing 9.1: Asynchronous call to schedule generating method

```
private fun persistSchedules(schedules: List<Schedule>, user: User) {

   user.schedules.addAll(schedules)

   schedules.forEach { schedule ->
         schedule.calendarEvents.forEach {
         it.schedules.add(schedule)
         em.persist(it)
      }
      schedule.assignedTasks.forEach {
         it.schedule = schedule
         em.persist(it)
      }
   }
   schedules.forEach(em::persist)
}
```

Listing 9.2: Method for persisting generated schedules

9.2.2 Score Calculation

`DayAssignmentScoreCalculator` incorporates the logic of hard constraints and calls the `calculateScore()` method on `SoftScoreCalculator` returning the soft score based on which the assignment is optimized. Private methods of this class are responsible for checking the feasibility of task assignment. In the following listing '`||`' is replaced with OR, since pipe signs are not represented properly.

```
/**
* The score calculator provides the way to calculate, if the hard constraints
* are fulfilled and assigns softScore to find more optimal schedule.
*/
@Component
class DayAssignmentScoreCalculator : EasyScoreCalculator<Schedule> {

   private val softScoreCalculator = SoftScoreCalculator()

   override fun calculateScore(schedule: Schedule): HardSoftScore {
      var hardScore = 0
      var softScore = 0
      //time slot with the id of -1 is considered unassigned
      val assignedTasks = schedule.allTasks
               .filter({ t -> t.startingTimeSlot != null
                           && t.startingTimeSlot.id != -1 })
               .sortedBy { it.startingTimeSlot.id }
```

```
    for ((i, task) in assignedTasks.withIndex()) {
       if (!fitsIntoDaySchedule(task, schedule)) {
          hardScore -= 1000
       } else if (overlapsWithEvent(task, schedule)
                && !canTaskBePartitioned(task, schedule.calendarEvents)) {
          hardScore -= 1000
       } else if (i != assignedTasks.size - 1
                && overlap(task, assignedTasks[i + 1])) {
          hardScore -= 1000
       } else {
          hardScore += 100 // for each assigned task
          softScore += softScoreCalculator.calculateScore(task)
       }
    }
    return HardSoftScore.valueOf(hardScore, softScore)
 }

 private fun overlap(curTask: Task, nextTask: Task): Boolean {
    val nextTimeSlotId = getNextTimeSlotId(curTask)
    return nextTimeSlotId > nextTask.startingTimeSlot.id
 }

 private fun fitsIntoDaySchedule(task: Task, schedule: Schedule): Boolean {
    val nextTimeSlotId = getNextTimeSlotId(task)
    val taskFinishedTime = task.startingTimeSlot.startTime
                           .plusMinutes(task.duration.toLong())
    return nextTimeSlotId < schedule.timeSlots.size
             && !taskFinishedTime.isAfter(schedule.endOfDay)
 }

 private fun getNextTimeSlotId(curTask: Task): Int {
    val timeSlotId = curTask.startingTimeSlot.id
    val taskDurationInNumberOfSlots = curTask.duration / TIME_SLOT_DURATION
    return timeSlotId + taskDurationInNumberOfSlots
 }

 /**
 * Check, if the task assignment is overlapping with the event scheduled in the day
 * @return false, if there is no overlap
 */
 fun overlapsWithEvent(task: Task, schedule: Schedule): Boolean {
    if (schedule.calendarEvents.isEmpty()) return false

    val sortedEvents = schedule.calendarEvents
    val taskStarts = task.startingTimeSlot.startTime
    val taskEnds = task.startingTimeSlot.startTime.plusMinutes(task.duration.toLong())
```

```
      if (taskEnds.isBefore(sortedEvents.first().start)
         OR taskStarts.isAfter(sortedEvents.last().end))
         return false

      sortedEvents.forEach { event ->
         if (taskStarts.isBefore(event.start) &&
            !(taskEnds.isBefore(event.start) OR taskEnds == event.start))
         return true
         }
      return false
   }

   /**
   * Checks, if the task can be split into the subtasks. Task can be split, if
   * it is not to short: duration >= 60 minutes
   * if the portion is large enough (> MIN_TASK_PARTITION_DURATION)
   * if the breaking event is not too long
   * @return false, if the partitioning of the task does not satisfy conditions above
   */
   fun canTaskBePartitioned(task: Task, events: List<CalendarEvent>): Boolean {
      if (task.duration < 60) return false
      else {
         val overlappingEvents = findOverlappingEvents(task = task, events = events)
         var remainingTaskDuration = task.duration
         var taskStart = task.startingTimeSlot.startTime

         for (event in overlappingEvents) {
            val eventDuration = Duration.between(event.start, event.end).toMinutes()
            val partition = Duration.between(taskStart, event.start).toMinutes()
            remainingTaskDuration -= partition.toInt()

            if (partition < MIN_TASK_PARTITION_DURATION eventDuration > 2 * partition)
               return false
            else
               taskStart = event.end
         }
         return remainingTaskDuration >= MIN_TASK_PARTITION_DURATION
      }
   }
}
```

Listing 9.3: DayAssignmentScoreCalculator class

```
fun findOverlappingEvents(task: Task? = null, assignedTask: AssignedTask? = null,
                        events: List<CalendarEvent>): List<CalendarEvent> {

   require(task != null assignedTask != null)

   val taskStarts: LocalTime
   var taskEnds: LocalTime
   val overlaps = mutableListOf<CalendarEvent>()

   if (task != null) {
      taskStarts = task.startingTimeSlot.startTime
      taskEnds = task.startingTimeSlot.startTime.plusMinutes(task.duration.toLong())
   } else {
      taskStarts = assignedTask!!.startingTime
      taskEnds = assignedTask.startingTime
                        .plusMinutes(assignedTask.task.duration.toLong())
   }
   val possiblyOverlappingEvents = events.filter { it.start > taskStarts }
   possiblyOverlappingEvents.forEach {event ->
      if (event.start in taskStarts..taskEnds && taskEnds != event.start) {
         overlaps.add(event)
         taskEnds = taskEnds.plusMinutes(Duration.between(event.start, event.end)
                                             .toMinutes())
      } else {
         return overlaps
      }
   }
   return overlaps
}
```

Listing 9.4: Function for finding overlapping events

9.2.3 Rules

Rules are calculating soft score based on the productivity constraints and task parameters. To exemplify how the rules work and how they were implemented one implementation of the Rule interface is provided below.

```
/**
 * Rule incorporates the logic about the impact of sleep duration and quality on
 * the productivity level.
 * Factors:
 * - sleep quality
 * - sleep duration
 * - task difficulty
 *
 * Generally, the larger sleep debt (hours slept - suggested sleep duration as
 * defined in the ConfigurationConstants)
 * => the lower the productivity
 * the worse quality of sleep => the lower is the productivity
 * the harder is the task => the more impact the previous two factors
 * will have (positive as well as negative)
 *
 * Mostly this rule will give a large penalty for the difficult task, if the
 * sleep duration or quality were insufficient
 * If sleep quality and duration were good, then score will be positive,
 * showing the boost to productivity.
 *
 * The range of the score returned will be in the interval
 * [-SCORE_SPAN, +SCORE_SPAN] whereas it will never be reaching
 * the upper bound, since the boost to the productivity is not
 * as large as penalty.
 */
class TirednessRule(
      override var weight: Double = TIREDNESS_RULE_DEFAULT_WEIGHT) : Rule {
   override fun init(user: User, today: LocalDate) {
      this.user = user
   }

   /* the cutoff for the sleep duration everything below this sleep duration
    * will be treated as having equally negative impact on the productivity */
   internal val sleepDurationCutoff = 4.0

   lateinit var user: User

   override fun getScore(task: Task): Int {
      // information about the user related to user's tiredness
      // sorting the daily forms by date
      // to have the most recent one as last entry
      user.dailyForms.sortBy { it.date }
      val dailyFormInfo = user.dailyForms.last()
      val sleepDuration = dailyFormInfo.sleepDuration
      val sleepQuality = dailyFormInfo.sleepQuality

      val sleepScore = calculateSleepScore(sleepDuration, sleepQuality)
      val sleepAndTaskDifficultyScore = task.difficulty.score * sleepScore

      return sleepAndTaskDifficultyScore.round()
   }

   @VisibleForTesting
   internal fun calculateSleepScore(sleepDuration: Double,
                                    sleepQuality: SleepQuality): Double {

      val normalizedSleepDuration = max(sleepDurationCutoff, sleepDuration)
      val sleepDurationScore = getSleepDurationScore(normalizedSleepDuration)

      val res = if (sleepDurationScore < 0) {
         when (sleepQuality) {
         // good sleep mitigates negative effect of shorter sleep duration
            GOOD, NORMAL -> sleepDurationScore / sleepQuality.score
         // bad sleep magnifies negative effect of lack of sleep
            BAD, VERY_BAD -> sleepDurationScore * abs(sleepQuality.score)
         }
         /* if sleep duration is in sleep tolerance zone,
           score depends largely on quality of sleep */
      } else {
         if (sleepQuality == NORMAL) 0.0
         else sleepDurationScore + sleepQuality.score
      }
      return res * getNormalizationMultiplier()
   }

   /* Absolute value by which the calculated results will be multiplied to
    * bring them to the same scale with other rules */
   private fun getNormalizationMultiplier(): Double {
      val minimalScore = getSleepDurationScore(sleepDurationCutoff) *
            VERY_BAD.score * TaskDifficulty.CHALLENGING.score

      return abs(ConfigurationConstants.SCORE_SPAN / minimalScore)
   }
}
```

Listing 9.5: TirednessRule implementation as example of rule

9.2.4 Solver Configuration

```
<solver>
   <!-- Define model -->
   <solutionClass>ch.uzh.ifi.seal.domain_classes.Schedule</solutionClass>
   <entityClass>ch.uzh.ifi.seal.domain_classes.Task</entityClass>
   <environmentMode>FAST_ASSERT</environmentMode>

   <!-- Define score function -->
   <scoreDirectorFactory>
      <scoreDefinitionType>HARD_SOFT</scoreDefinitionType>
      <easyScoreCalculatorClass>
         ch.uzh.ifi.seal.application_logic.DayAssignmentScoreCalculator
      </easyScoreCalculatorClass>
   </scoreDirectorFactory>

   <!-- Configuration for optimization algorithms (optional) -->
   <termination>
```

```
      <secondsSpentLimit>60</secondsSpentLimit>
   </termination>

   <constructionHeuristic>
      <constructionHeuristicType>FIRST_FIT</constructionHeuristicType>
   </constructionHeuristic>
   <localSearch>
      <acceptor>
         <lateAcceptanceSize>400</lateAcceptanceSize>
      </acceptor>
      <forager>
         <acceptedCountLimit>1</acceptedCountLimit>
      </forager>
   </localSearch>
</solver>
```

Listing 9.6: Configuration for OptaPlanner planning engine

9.3 Diagrams

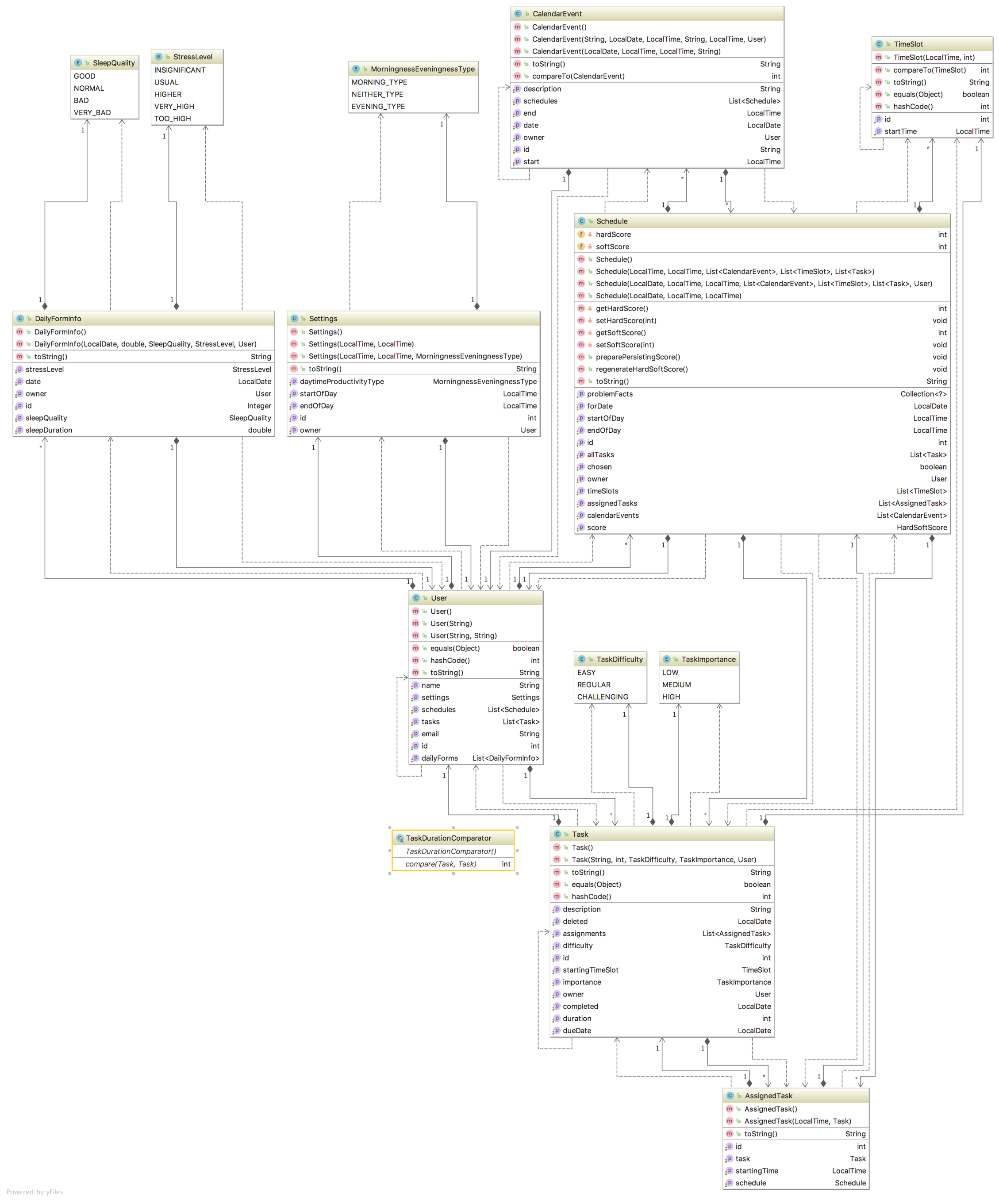


Figure 9.3: UML class diagram of classes in domain_classes package

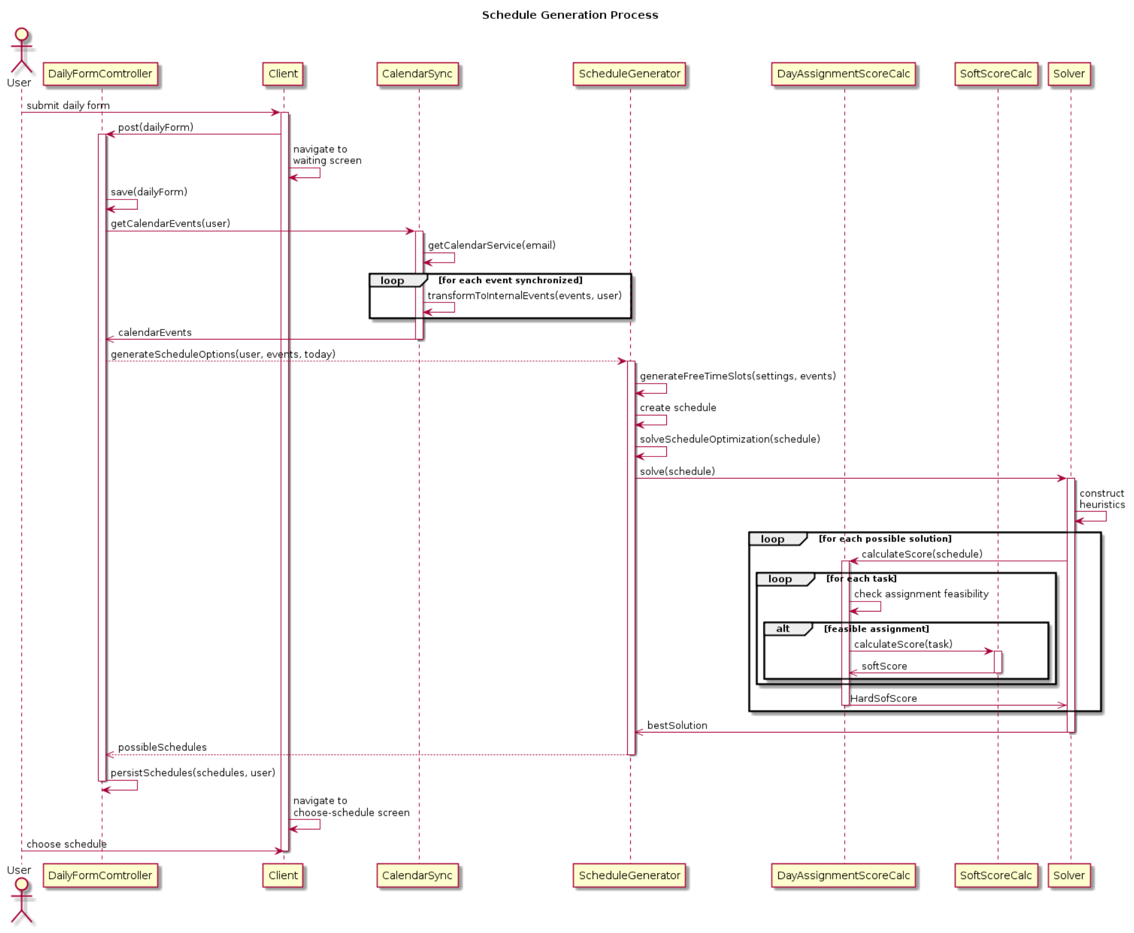
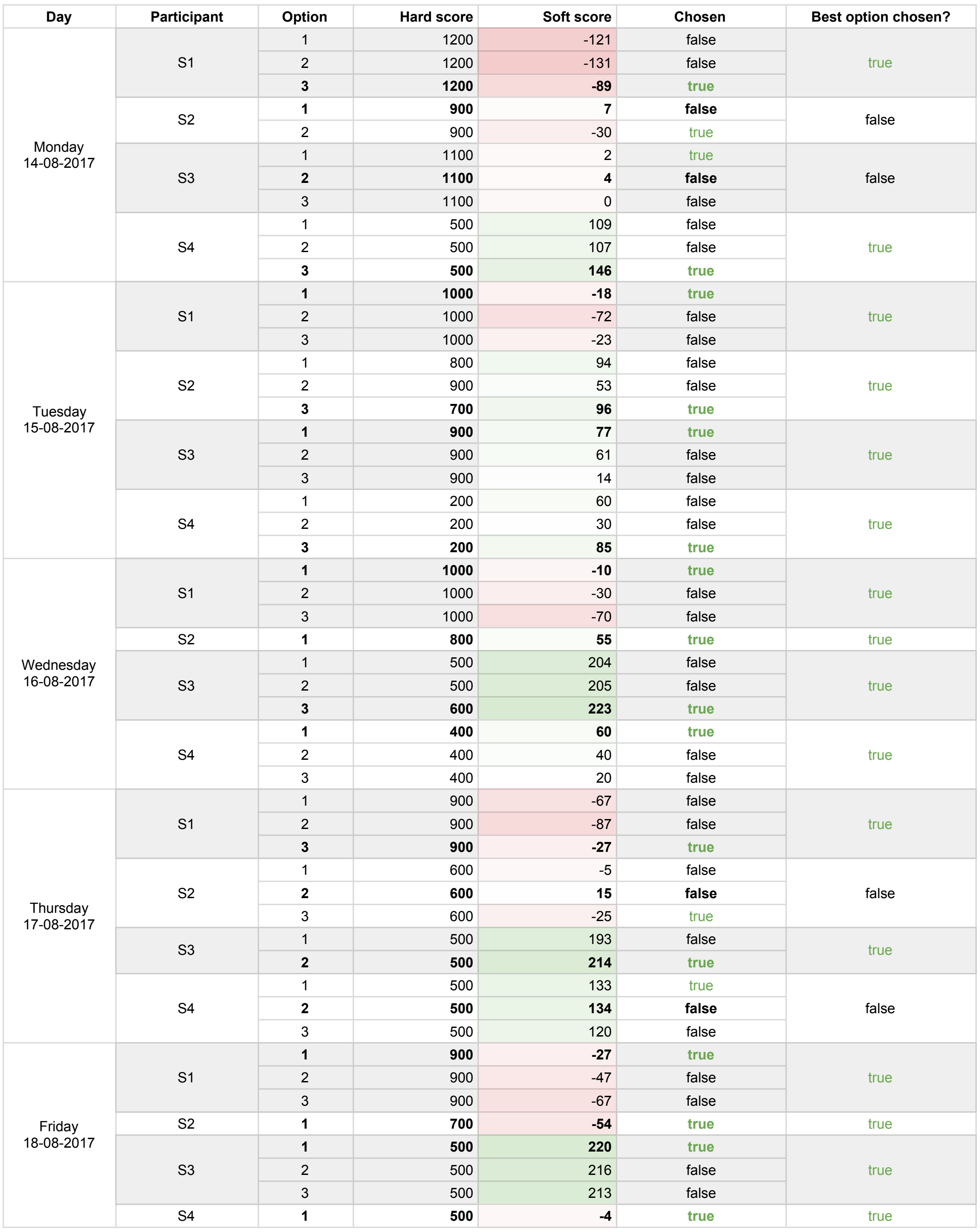

Schedule Generation Process
User
DailyFormComtroller
Client
CalendarSync
ScheduleGenerator
DayAssignmentScoreCalc
SoftScoreCalc
Solver
submit daily form
post(dailyForm)
navigate to waiting screen
save(dailyForm)
getCalendarEvents(user)
getCalendarService(email)
loop [for each event synchronized]
transformToInternalEvents(events, user)
calendarEvents
generateScheduleOptions(user, events, today)
generateFreeTimeSlots(settings, events)
create schedule
solveScheduleOptimization(schedule)
solve(schedule)
construct heuristics
loop [for each possible solution]
calculateScore(schedule)
loop [for each task]
check assignment feasibility
alt [feasible assignment]
calculateScore(task)
softScore
HardSofScore
bestSolution
possibleSchedules
persistSchedules(schedules, user)
navigate to choose-schedule screen
choose schedule

Figure 9.4: Sequence diagram of the schedule generation process

9.4 Evaluation



Day	Participant	Option	Hard score	Soft score	Chosen	Best option chosen?
Monday 14-08-2017	S1	1	1200	-121	false	true
		2	1200	-131	false	
		3	**1200**	**-89**	**true**	
	S2	**1**	**900**	**7**	**false**	false
		2	900	-30	true	
	S3	1	1100	2	true	false
		2	**1100**	**4**	**false**	
		3	1100	0	false	
	S4	1	500	109	false	true
		2	500	107	false	
		3	**500**	**146**	**true**	
Tuesday 15-08-2017	S1	**1**	**1000**	**-18**	**true**	true
		2	1000	-72	false	
		3	1000	-23	false	
	S2	1	800	94	false	true
		2	900	53	false	
		3	**700**	**96**	**true**	
	S3	**1**	**900**	**77**	**true**	true
		2	900	61	false	
		3	900	14	false	
	S4	1	200	60	false	true
		2	200	30	false	
		3	**200**	**85**	**true**	
Wednesday 16-08-2017	S1	**1**	**1000**	**-10**	**true**	true
		2	1000	-30	false	
		3	1000	-70	false	
	S2	**1**	**800**	**55**	**true**	true
	S3	1	500	204	false	true
		2	500	205	false	
		3	**600**	**223**	**true**	
	S4	**1**	**400**	**60**	**true**	true
		2	400	40	false	
		3	400	20	false	
Thursday 17-08-2017	S1	1	900	-67	false	true
		2	900	-87	false	
		3	**900**	**-27**	**true**	
	S2	1	600	-5	false	false
		2	**600**	**15**	**false**	
		3	600	-25	true	
	S3	1	500	193	false	true
		2	**500**	**214**	**true**	
	S4	1	500	133	true	false
		2	**500**	**134**	**false**	
		3	500	120	false	
Friday 18-08-2017	S1	**1**	**900**	**-27**	**true**	true
		2	900	-47	false	
		3	900	-67	false	
	S2	**1**	**700**	**-54**	**true**	true
	S3	**1**	**500**	**220**	**true**	true
		2	500	216	false	
		3	500	213	false	
	S4	**1**	**500**	**-4**	**true**	true

Figure 9.5: Table of schedules generated during user evaluation with scores and user choice

Bibliography

[1] Microsoft to-do application, official website: https://todo.microsoft.com.

[2] Optaplanner — open-source constraint satisfaction solver, official website: https://www.optaplanner.org/, documentation: https://docs.optaplanner.org/7.1.0.Final/optaplanner-docs/html_single/index.html.

[3] Purp — to-do list and goal-tracker application; official website: http://www.purpapp.com/; "brain" feature description: https://www.producthunt.com/posts/purp.

[4] Todoist — collaborative task management; official website: https://en.todoist.com/; description of "smart schedule" feature: https://blog.todoist.com/2016/11/16/todoist-smart-schedule/.

[5] D. Allen. *Getting Things Done: The Art of Stress-Free Productivity*. Penguin Books, 2015.

[6] T. Althoff, E. Horvitz, R. W. White, and J. Zeitzer. Harnessing the web for population-scale physiological sensing: A case study of sleep and performance. In *Proceedings of the 26th International Conference on World Wide Web*, pages 113–122. International World Wide Web Conferences Steering Committee, 2017.

[7] S. Banks and D. F. Dinges. Behavioral and physiological consequences of sleep restriction. *J Clin Sleep Med*, 2007.

[8] J. E. Biddle and D. S. Hamermesh. Sleep and the allocation of time. *Journal of Political Economy 98.5*, 1990.

[9] M. Boles, B. Pelletier, and W. Lynch. The relationship between health risks and work productivity. *Journal of Occupational and Environmental Medicine*, 46(7):737–745, 2004.

[10] S. Boyd and L. Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[11] A. Bradley, D. P. Brumby, A. L. Cox, and J. Bird. How to manage your inbox: is a once a day strategy best? *Proceedings of the 27th International BCS Human Computer Interaction Conference*, page 20, 2013.

[12] S. C. Brailsford, C. N. Potts, and B. M. Smith. Constraint satisfaction problems: Algorithms and applications. *European Journal of Operational Research*, 119(3):557–581, 1999.

[13] N. Brixius. Explaining task scheduling for a project, Jan. 5 2006. US Patent App. 10/881,900.

[14] D. N. Card. The challenge of productivity measurement. *Pacific Northwest Software Quality Conference*, pages 1–10, 2006.

[15] P. Carr and Y. Lu. Information technology and knowledge worker productivity: A taxonomy of technology crowding. *AMCIS 2007 Proceedings*, page 51, 2007.

[16] B. J. Claessens et al. *Perceived control of time: Time management and personal effectiveness at work*. Technische Universiteit Eindhoven Eindhoven, 2004.

[17] D. Clark. Method and apparatus for planning and monitoring multiple tasks based on user defined criteria and predictive ability and for automatically detecting task related work, June 13 2006. US Patent 7,062,449.

[18] D. Clements-Croome and L. Baizhan. Productivity and indoor environment. *Proceedings of Healthy Buildings*, 1:629–634, 2000.

[19] S. Cohen, T. Kamarck, R. Mermelstein, et al. Perceived stress scale. *Measuring stress: A guide for health and social scientists*, 1994.

[20] T. W. Colligan and E. M. Higgins. Workplace stress: Etiology and consequences. *Journal of workplace behavioral health*, 21(2):89–97, 2006.

[21] M. Czerwinski, E. Horvitz, and S. Wilhite. A diary study of task switching and interruptions. *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 175–182, 2004.

[22] L. Dabbish, G. Mark, and V. M. González. Why do i keep interrupting myself?: environment, habit and self-interruption. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 3127–3130, 2011.

[23] T. DeMarco and T. Lister. Programmer performance and the effects of the workplace. *Proceedings of the 8th international conference on Software engineering*, pages 268–272, 1985.

[24] I. Donald, P. Taylor, S. Johnson, C. Cooper, S. Cartwright, and S. Robertson. Work environments, stress, and productivity: An examination using asset. *International Journal of Stress Management*, 12(4):409, 2005.

[25] M. J. Eppler and J. Mengis. The concept of information overload: A review of literature from organization science, accounting, marketing, mis, and related disciplines. *The information society*, 20(5):325–344, 2004.

[26] T. Ertemalp. Method and apparatus for arranging and displaying task schedule information in a calendar view format, Apr. 28 1998. US Patent 5,745,110.

[27] J. F. Gaultney. The prevalence of sleep disorders in college students: Impact on academic performance. *Journal of American College Health*, 2010.

[28] T. Gillie and D. Broadbent. What makes interruptions disruptive? a study of length, similarity, and complexity. *Psychological research*, 50(4):243–250, 1989.

[29] V. M. González and G. Mark. Constant, constant, multi-tasking craziness: managing multiple working spheres. *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 113–120, 2004.

[30] C. Gross and K. Seebaß. The standard stress scale (sss): Measuring stress in the life course. *NEPS Working Paper*, 45, 2014.

[31] P. A. Hancock. A dynamic model of stress and sustained attention. *Human factors*, 1989.

[32] I. Harjunkoski, C. T. Maravelias, P. Bongers, P. M. Castro, S. Engell, I. E. Grossmann, J. Hooker, C. Méndez, G. Sand, and J. Wassick. Scope for industrial applications of production scheduling models and solution methods. *Computers & Chemical Engineering*, 2014.

[33] F. Heylighen and C. Vidal. Getting things done: the science behind stress-free productivity. *Long Range Planning*, 41(6):585–605, 2008.

[34] J. A. Horne and O. Östberg. Individual differences in human circadian rhythms. *Biological Psychology*, 5(3):179–190, 1977.

[35] J. A. Horne and O. Östberg. A self-assessment questionnaire to determine morningness-eveningness in human circadian rhythms. *International Journal of Chronobiology*, 4(2), 1976.

[36] J. M. Hudson, J. Christensen, W. A. Kellogg, and T. Erickson. I'd be overwhelmed, but it's just one more thing to do: Availability and interruption in research management. *Proceedings of the SIGCHI Conference on Human factors in computing systems*, pages 97–104, 2002.

[37] S. Johnson, C. Cooper, S. Cartwright, I. Donald, P. Taylor, and C. Millet. The experience of work-related stress across occupations. *Journal of managerial psychology*, 20(2):178–187, 2005.

[38] P. Karr-Wisniewski and Y. Lu. When more is too much: Operationalizing technology overload and exploring its impact on knowledge worker productivity. *Computers in Human Behavior*, 26(5):1061–1072, 2010.

[39] K. Kushlev and E. W. Dunn. Checking email less frequently reduces stress. *Computers in Human Behavior*, 43:220–228, 2015.

[40] S. M. LaValle. *Planning Algorithms*. Cambridge university press, 2006.

[41] R. Levinson. Intelligent planning and calendaring system with cueing feature and floating tasks, Apr. 4 2000. US Patent 6,047,260.

[42] G. Mark, S. T. Iqbal, M. Czerwinski, and P. Johns. Bored mondays and focused afternoons: the rhythm of attention and online activity in the workplace. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 2014.

[43] G. Mark, S. T. Iqbal, M. Czerwinski, P. Johns, A. Sano, and Y. Lutchyn. Email duration, batching and self-interruption: Patterns of email use on productivity and stress. *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, pages 1717–1728, 2016.

[44] K. N. McKay and V. C. Wiers. Integrated decision support for planning, scheduling, and dispatching tasks in a focused factory. *Computers in Industry*, 50(1):5–14, 2003.

[45] A. N. Meyer, L. E. Barton, G. C. Murphy, T. Zimmermann, and T. Fritz. The work life of developers: Activities, switches and perceived productivity. *IEEE Transactions on Software Engineering*, 2017.

[46] A. N. Meyer, L. E. Barton, G. C. Murphy, T. Zimmermann, and T. Fritz. The work life of developers: Activities, switches and perceived productivity. *IEEE Transactions on Software Engineering*, 2017.

[47] N. Miller. Task management system, Aug. 8 2000. US Patent 6,101,481.

[48] C. A. Monk. The effect of frequent versus infrequent interruptions on primary task resumption. *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, 48(3):295–299, 2004.

[49] S.-J. Paine, P. H. Gander, and N. Travier. The epidemiology of morningness/eveningness: Influence of age, gender, ethnicity, and socioeconomic factors in adults (30-49 years). *Journal of Biological Rhythms*, 2006.

[50] L. A. Perlow. The time famine: Toward a sociology of work time. *Administrative science quarterly*, 44(1):57–81, 1999.

[51] P. Philip, J. Taillard, P. Sagaspe, C. Valtat, M. Sanchez-Ortuno, N. Moore, A. Charles, and B. Bioulac. Age, performance and sleep deprivation. *Journal of sleep research*, 13(2):105–110, 2004.

[52] R. Rhett. Method for automatically developing suggested optimal work schedules from unsorted group and individual task lists, Apr. 30 2009. US Patent App. 12/255,432.

[53] P. Roelofsen. The impact of office environments on employee performance: The design of the workplace as a strategy for productivity enhancement. *Journal of facilities Management*, 1(3):247–264, 2002.

[54] M. R. Rosekind, K. B. Gregory, M. M. Mallis, S. L. Brandt, B. Seal, and D. Lerner. The cost of poor sleep: Workplace productivity loss and associated costs. *JOEM*, 2010.

[55] J. Siegrist and R. Peter. Measuring effort-reward imbalance at work: guidelines. *Dusseldorf: Heinrich Heine University*, 1996.

[56] J. Siegrist, D. Starke, T. Chandola, I. Godin, M. Marmot, I. Niedhammer, and R. Peter. The measurement of effort–reward imbalance at work: European comparisons. *Social science & medicine*, 58(8):1483–1499, 2004.

[57] C. Speier, J. S. Valacich, and I. Vessey. The influence of task interruption on individual decision making: An information overload perspective. *Decision Sciences*, 30(2):337–360, 1999.

[58] J. B. Spira and J. B. Feintuch. The cost of not paying attention: How interruptions impact knowledge worker productivity. *Report from Basex*, 2005.

[59] T. W. Strine and D. P. Chapman. Associations of frequent sleep insufficiency with health-related quality of life and health behaviors. *Sleep medicine 6.1*, 2005.

[60] J. Taillard, P. Philip, and B. Bioulac. Morningness/eveningness and the need for sleep. *Journal of Sleep Research*, 1999.

[61] J. Taillard, P. Philip, J.-F. Chastang, and B. Bioulac. Validation of horne and ostberg morningness-eveningness questionnaire in a middle-aged population of french workers. *Journal of biological rhythms*, 19(1):76–86, 2004.

[62] S. Thomée, A. Härenstam, and M. Hagberg. Computer use and stress, sleep disturbances, and symptoms of depression among young adults–a prospective cohort study. *BMC psychiatry*, 12(1):176, 2012.

[63] M. D. B. Trockel, Mickey T. and D. L. Egget. Health-related variables and academic performance among first-year college students: implications for sleep and other behaviors. *Journal of American college health 49.3*, 2000.

[64] R. van Solingen, E. Berghout, and F. van Latum. Interrupts: just a minute never is. *IEEE software*, 15(5):97–103, 1998.

[65] R. Q. Wolever, K. J. Bobinet, K. McCabe, E. R. Mackenzie, E. Fekete, C. A. Kusnick, and M. Baime. Effective and viable mind-body stress reduction in the workplace: a randomized controlled trial. *Journal of occupational health psychology*, 17(2):246, 2012.

[66] J. M. Zelenski, S. A. Murphy, and D. A. Jenkins. The happy-productive worker thesis revisited. *Journal of Happiness Studies*, 9(4):521–537, 2008.