Support for Human Planners

Scheduling and Visualisation

Cas Buijs
Arthur Guijt
Lars Stegman

Delft University of Technology

TU Delft

Support for Human Planners

Scheduling and Visualisation

by

Cas Buijs
Arthur Guijt
Lars Stegman

to obtain the degree of Bachelor of Science
at the Delft University of Technology,
to be defended on Monday July 3rd, 2017 at 12:00h.

Students:	Cas Buijs Arthur Guijt Lars Stegman	
Project duration:	April 24th, 2017 – July 4th, 2017	
Project committee:	Dr. Mathijs de Weerdt, Dr. ir. Léon Planken, Ir. Otto Visser, Dr. Huijuan Wang	TU Delft, supervisor West IT Solutions, client Computer Science Bachelor Project Coordinator Computer Science Bachelor Project Coordinator

This thesis is confidential and cannot be made public until July 4, 2017.

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

TU Delft Delft University of Technology

Summary

West IT Solutions is expanding into a new focus area, ”Smart Planning”. They want to develop new modules for the Enterprise Resource Planning software they resell, Odoo. Odoo allows companies to manage their projects, employees and other resources. The goal of this project was to create a software product that assists users in creating plannings for their projects in Odoo.

The product should offer the following features:

- Visualise a planning for the user’s projects;
- Facilitate manually adapting the planning;
- Propose an automatically generated improvement of the planning.

During the research phase of the project we met with a client of West, TNO. TNO explained they are performing research for the Dutch navy to create software that assists planners in creating schedules for battleships. With this case problem in mind, we created a list of requirements our software product should adhere to. We discovered that the case problem West provided us with is a subset of the case problem TNO provided. This meant that if we created a model for the case problem of TNO, the model would work for the case problem of West as well.

After the research phase was completed, we started development on the software product. The development phase consisted of seven weeks, and in each week new functionalities were added to the product. In addition to adding new functionalities, we also kept refining our models to improve the performance of the automatic planning generation.

The final software product consists of an interface that visualises a planning, a model that can be entered into external algorithms to automatically generate plannings and a module that allows users to import existing projects from Odoo.

The user can view their planning through a visualisation that allows them to modify it using a drag and drop interaction. These modifications are setting preferences, like starting times for activities or the assignment of certain persons to certain activities. Once the user has set their preferences, they can ask the software to find an optimised planning that keeps their preferences in mind.

At the end of the project, we gave a demo to TNO and they were impressed with the results of the project. West IT was also happy with the delivered product and is interested in further developing it

Preface

Bringing this project to a successful end would not have been possible without the help provided by several people. First we would like to thank Léon Planken, our client, who, even though he was not our TU Coach, went above and beyond to support us on almost all aspects of the project. Second, we would like to thank Mathijs de Weerdt for helping us focus more on the academic aspect of the project, providing us with information on what algorithms to use and for always responding within 30 minutes of sending an e-mail. Third, we want to thank the people at West IT for providing us with a work space and making us feel welcome. We would also like to thank Patrick Hanckmann and Jan de Gier from TNO for providing us with their case problem. A final thanks goes to all people not mentioned above, who provided any contribution to our project.

Cas Buijs
Arthur Guijt
Lars Stegman
Delft, June 2017

Contents

List of Figures

List of Tables

List of Listings

List of Algorithms

1

Introduction

When working in a group to accomplish a goal, an efficient distribution of sub-goals between the persons in the group can lead to an earlier achievement of this goal. This distribution of sub-goals between the persons in a group is realised when a (work-)planning or also called, a work schedule, is created. In this planning persons are assigned to sub-goals and are given a timespan in which the sub-goals have to be achieved.

One of the problems with creating an efficient planning is that many factors have to be taken into account. Not every person might have the skills necessary for completing a sub-goal and it might happen that two sub-goals cannot be completed at the same time, because of a dependency relationship between them. When there are many persons in the group and many sub-goals to be achieved, the planning gets big and it will take more time to find the most efficient distribution.

This project is about supporting human planners by creating a system that can visualise a planning using data from Odoo, allows manual changes to be made in this visualisation and that can propose a new planning that takes into account the manual changes made by a planner.

Throughout this report the term *planning* is be used to refer to what was earlier called, a work schedule.

The essential parts of the research report have been integrated in this final report. The research report is not essential for understanding this report but is included as appendix for archival purposes.

This report consists of 10 chapters, each concerned with a different aspect of the project. Chapter 2 describes the project organisation, it explains who the client is and what the assignment for this project consists of. Chapter 3 gives an analysis of the problem described in chapter 2, lists the functional requirements and a discusses the ethical implications of the product. Chapter 4 describes the problem models we defined to generate plannings. Chapter 5 discusses the initial design of the product together with the design choices made, while chapter 6 describes the developed product. In chapter 7 a performance analysis of the product is given as well as an analysis of the code quality and chapter 8 discusses how the product has been tested and validated. Chapter 9 describes the development methodology and development tools used and discusses the feedback from SIG. Finally, chapter 10 suggests possible features to extend the product with, reflects on the project and concludes the report.

2

Project Organisation

This chapter provides general information about the project. In section 2.1 is explained who the client for this project is. In section 2.2 a description of the project is given and in section 2.3 the case problem used during this project is described. In section 2.4 the success criteria to evaluate the product and the project are given.

2.1. Client

The client that proposed this project on BEPSys is West IT Solutions (West). West is a company that, among other business, resells a software package called 'Odoo'. Odoo is an enterprise resource planning (ERP) system that allows for integration of different modules, with each its own functionality, into a single environment and which allows companies to centralise their business data. For example, with Odoo it is possible to centralise employee management, production management and project management, which can then be combined.

A part of West's business focusses on smart planning, planning optimisation using statistical and dynamical factors. By taking into account the dynamical factors and with the use of advanced mathematical models it is possible to generate a more efficient planning. It is this department that proposed the project in order to get more insight in the possibilities of smart planning in combination with Odoo.

2.2. Project Description

The proposed project consisted of three assignments, the original project description can be found in appendix H:

1. Visualising a manual planning made in Odoo
   It is possible to create a planning using the project management module in Odoo. The goal here is to create a visualisation of the planning made in Odoo.
2. Facilitate manually adapting this planning
   When the planning in Odoo has been visualised, it should be possible for the planner to make manual changes using the visualisation interface.
3. Propose an automatically generated improvement of the manual planning
   The goal here is to create a component that can automatically generate an improvement of the manual planning in Odoo, while taking into account any preferences the planner might have.

At the end of the project there should be a product that can be used to demonstrate to customers the possibilities of combining smart planning with Odoo. This kind of product makes it possible for companies to reduce the time necessary for creating a planning.

2.3. TNO Case

Achieving the goals aforementioned is the main task for this project, but for this an algorithm to create a planning is required. Our client wanted us to investigate a case problem of one of their clients, this case problem would provide us with the details an algorithm would have to take into account when creating a planning. Together with the client was decided which details would be taken into account during this project and which are left aside. The choices made can be found in the research report (appendix M). The case problem has been provided by TNO, a Dutch research institute that aids governments and other parties to apply scientific knowledge in real-world applications.

The case problem of TNO is concerned with the planning for a navy ship. The upcoming ship replacement program of the Royal Dutch navy requires a reduction in the number of crew members on a ship. This decrease in the number of crew members cannot affect the core activities that take place on board. The current plannings, which are created manually, are not efficient enough to both reduce the number of crew members and not affect the core activities while doing so.

The complete problem definition can be found in the research report, a short summary of the problem definition is provided:

The case problem is about the planning that has to be made for a navy ship. On board of the ship there are two types of activities, work activities or personal activities. When assigning these activities to crew members certain constraints have to be taken into account, to avoid running into practical problems on board:

- Crew members have to work in shifts.
- Crew members can only perform one activity at a time.
- Crew members have to be carrying out work activities at all times, an exception are the moments planned for personal activities.
- Crew members should have diverse schedules.
- Crew members have 4 hours of personal time a day, at least two of these hours should be right before them going to sleep.
- Crew members have 8 hours of sleep time in a day.
- A dependency relation between activities can exist.
- A subset of qualified crew members is be able to carry out an activity.
- The planning should take into account the planner's wishes.

2.4. Success Criteria

Evaluation criteria for both the final product and the project in general have been defined during the research phase of the project. A short overview of these criteria is provided, explanations for each criterion can be found in the research report.

To evaluate the success of the project, the following success criteria are used:

- The deadlines during the project have been met,
- The client and coach are pleased with the performed work,
- The process is properly documented,

- The students have met the educative goals of the project,
- The solution fulfils its success criteria.

As has been mentioned in the research report, failure to meet a success criterion does not immediately mean the project is unsuccessful. There might be valid reasons for why a certain criterion could not be met, these reasons are taken into account during the evaluation.

To evaluate the success of the final product to deliver, the following success criteria are used:

- The solution meets all 'Must Haves',
- The solution meets most 'Should Haves',
- The solution is properly documented,
- SIG feedback is positive,
- The solution is tested well.

Similar to the project evaluation, not meeting some of the criteria, does not mean the product is a failure. The reasons for not meeting a success criterion are taken into account during the evaluation.

3

Problem Analysis

This chapter covers the analysis of the case problem provided by TNO as described in section 2.3. Section 3.1 defines the mathematical representation of the case problem that is used in any further analysis. Section 3.2 proves the NP-hardness of the case problem. Assuming that $P \neq NP$, this means that there exists no polynomial algorithm to solve the problem. Finally, section 3.3 analyses the functional requirements for the product and section 3.4 describes the ethical implications of our product.

3.1. Mathematical Definition

At the start of the project together with the client was decided to choose a subset of the requirements defined by TNO. Because of this choice we could focus more on the important and unique aspects of the case problem. The elements we did not consider in our models are the assignment of personnel to shifts and the maximisation of utilisation of personnel.

The assignment of personnel to shifts is usually done by hand and should not be too hard to do manually as the assignments are not likely to change within the timespan of a planning. The maximisation of utilisation of personnel can easily be added later by extending the models we defined in chapter 4.

A work planning can be described using two components, activities that have to be performed and persons that can be assigned to perform the activities.

Person:

- μ The set of persons.

Activity:

- r_i Release time for activity i; The time from when the activity may be performed;
- p_i Processing time; The amount of time it takes to perform activity i;
- d_i Deadline; The time by which activity i should be completed;
- ω_i Delay penalty weight; The weight for a penalty in case activity i is scheduled to finish after its deadline;
- m_i^r Required number of persons; The number of persons needed to perform activity i;
- $\mu_i \subseteq \mu$ Qualified persons; The persons qualified to perform activity i;
- $\mu_i^r \subseteq \mu$ Mandatory assigned persons; Persons who must be assigned to activity i;
- $pred(i)$ Activity Dependencies; The activities activity i depends on.

The objective function used for this problem is to minimise the weighted tardiness [1]. This means that if an activity is finished after its deadline, the value of the objective function increases with the penalty weight multiplied by the amount of time it finishes after its deadline.

Objective function:

$$\gamma = \min \sum_i \omega_i T_i \tag{3.1}$$

$$= \min \sum_i (\omega_i \cdot \max\{0, c_i - d_i\}) \tag{3.2}$$

With T_i being the tardiness and c_i the completion time of the activity.

3.2. Problem Complexity

To make sure that no time had to be spend looking for polynomial algorithms to find solutions for our problem instances, unless $P = NP$, we proved our problem to be NP-hard. To prove this we give a reduction from a known NP-hard problem, namely the Open Shop Problem with Multiprocessor Tasks (OSMPT), to our problem. This section describes the OSMPT problem, shows that it is strongly NP-hard, describes the mapping from OSMPT to our problem and gives proof that the mapping is correct. We also show that our problem is in NP, which means that our problem is NP-complete.

3.2.1. Notation

The following notation is used in open shop problems:

- J The set of jobs;
- $n^J = |J|$ The total number of jobs;
- i Job i;
- O_i The set of operations for i;
- $|O_i| = k$ The number of operations for i, equal for all i;
- O_{ij} Operation j for i;
- r_i Release time for i;
- p_{ij} Processing time for O_{ij}.
- d_i Due date for i;
- s_i Start time for i;
- c_i Completion time for i;
- ω_i Penalty weight per unit of time for i;
- M The set of machines;
- $m = |M|$ The total number of machines;
- $\mu_{ij} \subseteq M$ The set of machines capable of performing operation O_{ij};
- $m_{ij} = |\mu_{ij}|$ The number of machines capable of performing O_{ij};
- $V(T)$ The set of nodes in graph T;
- $E(T)$ The set of edges in graph T;
- G^J Directed acyclic precedence graph for jobs;

- $pred(t_i) \subsetneq V(T)$ The items t_i depends on;
- $succ(t_i) \subsetneq V(T)$ The items that depend on t_i;
- γ The objective function

Additionally, we create the following notation to use in the mapping from OSMPT to our problem:

- $\mu_{ij}^r \subseteq \mu_{ij}$ The set of machines that must be assigned O_{ij};
- m_{ij}^r Number of machines required for performing O_{ij}. $1 \leq |\mu_{ij}^r| \leq m_{ij}^r \leq m_{ij}$
- $\mu_{ij}^c \supseteq \mu_{ij}^r$ The set of machines that have be chosen to perform O_{ij};

3.2.2. Open Shop Multiprocessor Task Problem

To explain the mapping in the next section, we first explain exactly what the OSMPT problem is and then show the NP-hardness for OSMPT. This section makes use of the explanations of shop problems in Scheduling Algorithms by Brucker [1].

OSMPT is the problem to find a schedule for a shop. The shop generates n^J goods, we call this jobs and each job consists of k operations. In the open shop problem it does not matter in which order these operations are performed. Each job has a release time r_i, the time work on the job may be started, a deadline d_i, the time by which work on the job should be completed, and a penalty ω_i, that can be used in the objective function. Every operation has a set of machines μ_{ij} required to perform it and a processing time p_{ij}, the time required to perform the operation. Additionally, the problem has a dependency graph G^J that indicates the order in which jobs should be completed. In the open shop problem, every job consists of the same operations that each require the same machines. This means that operation $O_{i'j}$ requires the same machines as O_{ij}. The open shop problem also assumes that machines are not used twice by the same job.

The goal of the problem is to find a schedule that satisfies a certain objective function, γ. A schedule consists of assignments of machines to operations and of operations to starting times. The objective function can consist of any linear combination of properties of the schedule, e.g. finding a schedule where the sum of all completion times is smaller than a number K.

NP-Hardness of Open Shop Multiprocessor Task Problem

We deduce that the OSMPT problem we use for the mapping is strongly NP-hard. To show this we make use of a list of known strongly NP-hard problems created by Sigrid Knust [5]. The list 'Shop Problems with Multiprocessor Tasks' shows that the problem $OMPT2|chains; p_{ij} = 1|\sum T_i$ is known to be strongly NP-hard. By taking $p_{ij} = 1$ for all operations, taking the precedence graph G^J to be a graph with chains[1], taking $\omega_i = 1$ for all jobs and having two machines capable of performing any operation, we have an instance of $OMPT2|chains; p_{ij} = 1|\sum T_i$. This means that the more general OSMPT problem we use in our mapping will most likely be strongly NP-hard as well.

3.2.3. NP-Hardness Proof

This section describes the proof that our problem is strongly NP-hard by describing a reduction from OSMPT to our problem.

Mapping from OSMPT to Our Problem

To map OSMPT to our problem we create the mapping described below. We first describe the general idea behind the mapping and then give a formal definition.

[1] Chains is a special kind of precedence relationship for which every item depends on a maximum of one other item.

As our problem has no concept of operations, every operation in OSMPT is mapped to a separate activity. Since release times, deadlines and penalty weight are defined per job, every activity in the mapped object gets these properties from their ”parent” job. We also map the dependency constraints of jobs to dependency constraints between their operations. These dependencies are represented by the resulting graph G'. Properties like processing time and capable machines/persons are not modified.

The mapping function g from OSMPT to our problem is defined as follows:

$$A' = \bigcup_i O_i \tag{3.3}$$

$$r_{ij} = r_i \qquad \text{for } i = 1, \dots, n^J;\ j = 1, \dots, k \tag{3.4}$$

$$p_{ij} = p_{ij} \qquad \text{for } i = 1, \dots, n^J;\ j = 1, \dots, k \tag{3.5}$$

$$d_{ij} = d_i \qquad \text{for } i = 1, \dots, n^J;\ j = 1, \dots, k \tag{3.6}$$

$$\omega_{ij} = \omega_i \qquad \text{for } i = 1, \dots, n^J;\ j = 1, \dots, k \tag{3.7}$$

$$\mu_{ij} = \mu_{ij} \qquad \text{for } i = 1, \dots, n^J;\ j = 1, \dots, k \tag{3.8}$$

$$\mu^r_{ij} = \mu_{ij} \qquad \text{for } i = 1, \dots, n^J;\ j = 1, \dots, k \tag{3.9}$$

$$m^r_{ij} = |\mu_{ij}| \qquad \text{for } i = 1, \dots, n^J;\ j = 1, \dots, k \tag{3.10}$$

$$V(G') = \bigcup_i O_i \tag{3.11}$$

$$E(G') = \left\{ O_{i'} \times O_i \;\middle|\; i' \in pred(i) \right\} \tag{3.12}$$

$$\gamma = \gamma \tag{3.13}$$

$$K = K \tag{3.14}$$

The mapping of operations to activities is described in equation 3.3.

Release times (eq. 3.4), processing times (eq. 3.5), due dates (eq. 3.6), penalty weights (eq. 3.7) and capable machines (eq. 3.8) can be mapped one to one.

Operations each have a set of machines required to perform it, so we set μ^r_{ij} equal to the set μ_{ij}. Because OSMPT requires every machine in μ_{ij} to perform O_{ij}, we need to set m^r_{ij} (eq. 3.10) equal to the number of required machines. The machines in OSMPT are mapped to persons in our problem.

The precedence graph G' represents the precedence between jobs. Because every operation of the original instance is mapped to a job (eq. 3.11) in the new instance, we need to adjust the precedence graph. We create an edge from each operation in $O_{i'}$ to every operation in O_i if i depends on i' (eq. 3.12).

In figure 3.1 the concept of mapping operations to activities is made more clear. Jobs 1 and 2 each consist of $k = 3$ operations. These operations are mapped to separate activities and the dependency between jobs 1 and 2 is translated to dependencies between the operations of these jobs. This ensures that the input precedence graph G^J and the output precedence graph G' have the same meaning, even though they are different.

Polynomiality of the Mapping

We take the size of the input to be defined as the total number of operations, i.e.

$$n = |input| = n^J \cdot k$$

Transforming each operation to an activity, including properties like release time, due date etc., takes a linear amount of time in the number of operations, thus this step is $O(n)$. The maximum number of edges that can be constructed by the mapping is: $\binom{n^J}{2} \cdot k^2$. This is the maximum number of edges in a DAG [4] times the number of edges created per edge in $E(G^J)$.

$$\begin{aligned}\binom{n^J}{2} \cdot k^2 &= \frac{1}{2} n^J (n^J - 1) \cdot k^2 \\ &= \frac{1}{2} n^J k \cdot (n^J - 1)k \\ &= \frac{1}{2} n^J k \cdot (n^J k - k) \\ &= O\big(n^J k \cdot (n^J k - k)\big) \\ &= O(n \cdot (n - k)) \\ &= O(n^2 - nk) \\ &= O(n^2)\end{aligned}$$

This means that the number of created edges is polynomial in the input size. There is one caveat with this argument: we have ignored the size of the dependency graph. If we would take this into account, n would become bigger and as we have used a smaller n, and the order is still polynomial, using a bigger n cannot influence the polynomiality.

Using the reasoning above and the fact that constructing each edge takes a constant amount of time, creating the mapped instance takes $O(n^2) + O(n) = O(n^2)$ time and is thus polynomial in the input size.

Correctness Proof

To prove that the mapping described is correct, we use the decision versions of the Open Shop Multiprocessor Task problem and our problem. These problems can be defined as follows:

> Does a schedule exist for which $\gamma = \Sigma_i \omega_i T_i \leq K$ holds?

This means that an instance is a yes instance if there exists a planning for which the weighted tardiness is less than or equal to K where K is an arbitrary positive number.

$\text{Yes}_{\text{OSMPT}} \Rightarrow \text{Yes}_{\text{Our Problem}}$ First we prove that if an instance of OSMPT is a yes instance, that the mapped version of this instance is a yes instance of our problem.

If we assume that $X = (M, J, O, G, \mu, r, p, d, \omega)$ is a yes instance of OSMPT, then there must exist a schedule s that satisfies $\gamma \leq K$.

The mapping g keeps the qualified machines, the release and processing times, deadlines and penalty weights exactly the same. Additionally, the mapped dependencies between activities enforce the same order of execution as the dependencies between the jobs did. This is true, because it does not matter in which order the operations within jobs are executed. However, the order in which jobs are executed is important. If job i is dependent on i', the

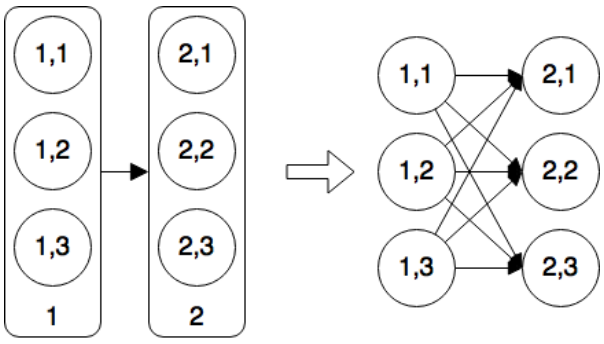


Figure 3.1: Mapping Dependencies

new dependencies enforce that every operation in i' is completed before any operation in i can be started. This is equivalent to enforcing that job i' is completed before job i is started.

Because all parameters are the same, or have an equivalent meaning, we can construct a schedule s' in which starting times for activities and starting times for the operations they represent are the same as in s. It is also possible to use the same person to activity assignment as the machine to operation assignment in s. Because the starting times, deadlines, processing times and penalty weights in s' and $g(X)$ have the same values in s and X, the value of γ does not change. This means that $g(X)$ is a yes instance of our problem

$\textbf{Yes}_{\textbf{Our Problem}} \Rightarrow \textbf{Yes}_{\textbf{OSMPT}}$ We now prove that if $g(X)$ is a yes instance of our problem, then X must be a yes instance of OSMPT.

We assume that $g(X)$ is a yes instance. This means that a schedule s' exists for which $\gamma \leq K$ holds. The dependency constraints in $g(X)$ represent dependencies between jobs, thus the order of activities in s' is a valid order for the operations in jobs. Every activity in $g(X)$ represents an operation in X. These activities kept the same release and processing time as the operations had in X. The same holds for the deadline, penalty weight and the qualified persons/machines. By keeping the starting times and assignments for operations in X the same as the values in $g(X)$, a valid schedule s can be constructed. Because the starting times, deadlines, processing times and penalty weights are kept the same, the value for γ should not change and thus s should be a valid schedule for X. This means that X is a yes instance of OSMPT.

From the proofs above we can conclude that the reduction is correct and thus that our problem must be strongly NP-hard as well.

3.2.4. Polynomial Verifiability

Schedules can also be verified in a polynomial amount of time, thus our problem is a member of the complexity class NP. An algorithm for verifying schedules can be found in appendix B.

3.2.5. Conclusion

It can be concluded that the mapping described is a valid mapping and that the reduction is correct. Since OSMPT is a strongly NP-hard problem, our problem must be strongly NP-hard as well. Because our problem is in NP and in NP-hard, we can also conclude that our problem is NP-complete. Assuming $P \neq NP$, this means that no polynomial algorithm for our problem exists.

3.3. Functional Requirements

At the start of the project, in consultation with the client, a set of functional requirements have been defined. The functional requirements have been prioritised using the MoSCoW method[2], this means that each of the functional requirements have been placed in 1 of 4 categories: Must Have, Should Have, Could Have or Won't Have. The requirements belonging to the category *Must Have* form the 'Minimal Viable Product (MVP)', these requirements have to be implemented. The requirements belonging to the category *Should Have* extend the MVP and should be implemented, but not at all cost. The requirements belonging to the category *Could Have* are desirable for the client, but having a product without these requirements implemented will cause the client no problems. As the category *Won't Have* adds no real value to the list of requirements, this category has been omitted.

For each of the requirements, if applicable, a number in brackets has been added at its end. These numbers refer to the goals in section 2.2 the requirements are related to.

[2] https://www.agilebusiness.org/content/moscow-prioritisation-0

Must Have

1. The system shall be able to create an initial planning [1]
2. The system shall have a visual interface of the planning. [1]
3. The system allows saving the created planning. [2]
4. The system allows manually changing the planning [2]

Should Have

5. The system allows automatic retrieval of persons from the Odoo system. [1]
6. The system allows automatic retrieval of tasks from the Odoo system. [1]
7. The system allows automatic retrieval of roles (qualifications/skills) from the Odoo system. [1]
8. The system shall give multiple suggestions for a planning [1]
9. The system shall inform the user when a created planning violates constraints [2]
10. The system allows computing planning using the manual adjustments of the user [3]

Could Have

11. The system allows manual adding/changing/removal from persons. [2]
12. The system allows manual adding/changing/removal from tasks. [2]
13. The system allows manual adding/changing/removal from roles (qualifications/skills). [2]
14. The system allows assigning roles to persons. [2]
15. The system allows the user to choose different ways to represent the planning visually [1]
16. The system allows multiple users to view/edit/save the planning
17. The system shall give multiple suggestions for a planning that are significantly different from each other
18. The system allows assigning priorities to tasks [2]
19. The system allows people to view their personal schedule
20. The system allows the user to print the planning
21. The system shall give an explanation for how the planning was created

3.4. Ethical Implications

The product delivered for this project can have a large effect on society. In every organisation where a group of persons are working together to achieve a goal, a planning tells who works when and who is responsible for what. There are two ethical implications we discuss, the first is concerned with the increase in unemployment and the second is concerned with an ethical question about using a (semi-)automatically generated planning.

3.4.1. Increase in Unemployment

The product developed in this project is able to (semi-)automatically suggest a planning for a group of persons. This has some advantages as less time is required for creating a planning and the possible increase in efficiency compared to a manually created planning. Besides these advantages, there is also a disadvantage for the persons currently employed to create

the plannings and the persons that are employed because of the inefficiency of the manual plannings.

Having a human planner will still be necessary for organisations, as the product cannot take into account all factors a human planner might have the intuition to take into account. It will no longer be necessary to have multiple planners to speed up the process of creating the planning. A consequence could be that multiple planners are reduced to a single planner who is responsible for creating all plannings in an organisation, which leads to the other planners losing their jobs.

When more efficient plannings can be made it might happen that many employees become redundant in the organisation. An example is the case problem described in section 2.3, which is concerned with reducing the number of crew members on a navy ship. As the employees are redundant, they may be at risk of losing their jobs.

Although, the consequences may seem bad on first sight, they do not necessarily have to be. Instead of firing the redundant employees, an organisation has now the opportunity to grow without increasing the costs for labour. For example, a new division can be created, which opens up new positions that the redundant employees can fill. There will be no increase in the costs of labour for the organisation, while this new division can bring its own benefits. The responsibility for firing employees lies with the organisation, the product only provides additional information and should thus not be held accountable for the consequences.

3.4.2. Using a (Semi-)automatically Generated Work Planning

In the European Union (EU) a new General Data Protection Regulation becomes active in 2018, which grants citizens of the EU the right to meaningful information about the logic involved with profiling [2]. This does not concern our product at all, but it does show an increase in the importance of being able to explain automatic made decisions.

A (semi-)automatic generated planning has advantages, of which a few have been aforementioned, but it also has disadvantages. A disadvantage is that it can become quite difficult for the planner to understand the decisions the algorithm made to come to its final result. Products, like our final product, might be able to explain the decisions to a certain extent, but the more advanced an algorithm becomes and the more factors it takes into account, the more difficult it will become to explain its decisions.

The planner and the planned persons will have to put trust in an algorithm they do not understand. Not being able to understand it implies that it will be difficult to identify the source of any harm caused by the planning generated by this algorithm. This could be harm done to the organisation, as in providing a sub-optimal planning, or harm done to the persons the planning relates to, making some persons do extra work while other persons have only little to do.

The planning generated by the algorithm can be very efficient but might be difficult for a planner to understand. There have been so many factors taken into account by the algorithm, that the planner cannot discover why certain things are the way they are. This relates to the problem of not understanding the algorithm that creates the planning.

As for the creation of an efficient planning a number of factors might have to be taken into account that is too large for a human planner to work with, it will come down to the question of what is more important. Is it really necessary to understand the decisions made by the algorithm or does it affect the planned persons so little that it can be disregarded? This is a question the organisation using this product has to make for itself in consultation with its staff, the product only offers a possibility, it is not a necessity to use it.

4

Problem Models

This chapter discusses the Mixed Integer Programming (MIP) models created for solving the case problem as defined in section 3.1. Section 4.1 describes notable parts of the first MIP model and what led to the creation of further models. Section 4.2 discusses the changes made in the second MIP model and disadvantages that came with said change. Section 4.3 describes the third and last MIP model created to overcome a disadvantage of the first two. This chapter only presents details necessary for understanding the MIP models. Formal descriptions for models 1, 2 and 3 can be found in appendices C, D and E respectively.

4.1. MIP Model 1

Given the mathematical definition in section 3.1, we created a Mixed Integer Programming model to find solutions.

In order to constrain persons to only work on at most a single activity at a given moment, both the time and person assignment for activities need to be taken into account. To make reasoning about this easier, the first and second MIP models make use of discrete time.

There are two binary decision variables: assigning an activity to a specific time slot and assigning a person to an activity. This is represented by two binary matrices.

$$b_{it} = \begin{cases} 1 & \text{if activity } i \text{ is scheduled at time } t \\ 0 & \text{Otherwise} \end{cases} \tag{4.1}$$

$$v_{ij} = \begin{cases} 1 & \text{if person } j \text{ is assigned to activity } i \\ 0 & \text{Otherwise} \end{cases} \tag{4.2}$$

Finally, using a multiplication we can count for every point in time how much work a person has to do, and restrict it so one person is only performing a single activity at most at every point in time.

$$\sum_i b_{it} v_{ij} \leq 1 \text{ for } \begin{matrix} j=1,\ldots,m \\ t=1,\ldots,T_n \end{matrix} \tag{4.3}$$

With m the amount of persons, T_n the duration of the project and with i all activities

There is a problem though, constraint 4.3 contains a multiplication between two decision variables. This model definition is therefore not a linear model, but a quadratic one. This

means a more extensive and usually more expensive MIP solver is required and limits choice in related tools such as the solver abstraction layer as described in 5.2.2. Additionally, solvers usually have an easier time finding solutions for linear models than for quadratic models.

This was the reason for removing the multiplication and replacing it with something linear, to create MIP model 2.

4.2. MIP Model 2

All quadratic constraints in MIP model 1 are multiplications between binary variables. This multiplication can be interpreted as an AND operation between two binary variables. An AND between two binary variables a and b can also be done using two linear equations and a variable c, where c is the result of the AND operation. The multiplication $a \cdot b = c$ can be written as: $0 \leq a + b - 2c \leq 1$ [3].

To replace every multiplication between the decision variables, a new variable and two new constraints need to be added. This causes a large increase in variables and constraints, but there are no longer any quadratic constraints. This means that a wider variety of tools can be used.

Even though the model is now linear, time is still discrete. This means that if the timespan, or time resolution, scales up the model instances will grow very fast. One of the constraints that is sensitive to this is the constraint for dependencies.

$$\sum_{t}^{T_n} 2^{t-1}(b_{it} - b_{i't}) < 0 \qquad \begin{matrix} \text{if } \delta_{ii'} = 1 \\ \text{for} \quad \begin{matrix} i = 1, \dots, n \\ i' = 1, \dots, n;\ i' \neq i \end{matrix} \end{matrix} \tag{4.4}$$

By interpreting the sequence of bits that assign activities to timeslots as a number, it is possible to ensure one activity starts before another by constraining one number to be bigger than the other. This constraint contains a power of two which can cause some very big numbers to occur, and is therefore a potential issue for solvers.

To solve this problem we created MIP model 3.

4.3. MIP Model 3

While having discrete time was beneficial for easy definition of constraints, the size of the model was in many instances getting large. This happens when the time resolution or timespan of the planning becomes larger.

Unlike the previous two models, MIP Model 3 uses continuous variables to represent the starting time of an activity instead. This means that the number of time slots no longer affects the number of constraints and variables.

Just like the first two models, decision variable 4.5 indicates whether person j is assigned to activity i.

$$v_{ij} = \begin{cases} 1 & \text{if person } j \text{ is assigned to activity } i \\ 0 & \text{Otherwise} \end{cases} \tag{4.5}$$

New in this model is the continuous decision variable 4.6, which indicates the starting time for activity i.

$$s_i\ (\mathbb{R}) \qquad \text{Starting time of activity } i \tag{4.6}$$

To ensure that persons are not assigned to two activities at the same time, decision variables 4.7 and 4.8 are created and constraints 4.9 and 4.10 enforce that the constraint is not violated. $c_{ii'j}$ is forced into having the right value through the use of the AND constraint over all activities and persons. $c_{ii'}$ is forced into having the right value through the use of an OR operation over all persons, similar to the method described in [3].

$$c_{ii'j} = \begin{cases} 1 & \text{if person } j \text{ is assigned to activity } i \text{ and } i' \\ 0 & \text{Otherwise} \end{cases} \tag{4.7}$$

$$c_{ii'} = \begin{cases} 1 & \text{if any person is assigned to both activity } i \text{ and } i' \\ 0 & \text{Otherwise} \end{cases} \tag{4.8}$$

$$0 \leq v_{ij} + v_{i'j} - 2c_{ii'j} \leq 1 \qquad \text{for } \begin{matrix} i = 1, \dots, n \\ i' = i+1, \dots, n \\ j = 1, \dots, m \end{matrix} \tag{4.9}$$

$$-m + 1 \leq \sum_{j} (c_{ii'j}) - m\, c_{ii'} \leq 0 \qquad \text{for } \begin{matrix} i = 1, \dots, n \\ i' = i+1, \dots, n \end{matrix} \tag{4.10}$$

With $i, i' (i \neq i')$ both representing an activity, j representing a person and m being the total number of persons.

To make sure that activities that have overlapping persons assigned to them are not scheduled simultaneously, constraints 4.12 and 4.13 are introduced. These constraints become active once $c_{ii'} = 1$, because the L in term 1 becomes inactive. Depending on the value of $o_{ii'}$ either constraint 4.12 or constraint 4.13 becomes active, which results in either $s_i \geq s_{i'} + p_{i'}$ or $s_{i'} \geq s_i + p_i$, respectively, because term 2 makes the constraint either trivial or becomes 0.

$$o_{ii'} = \begin{cases} 1 & \text{if activity } i \text{ goes before activity } i' \\ 0 & \text{Otherwise} \end{cases} \tag{4.11}$$

$$\overbrace{L(1 - c_{ii'})}^{1} + \overbrace{L\, o_{ii'}}^{2} + s_i \geq s_{i'} + p_{i'} \qquad \text{for } \begin{matrix} i = 1, \dots, n \\ i' = i+1, \dots, n \end{matrix} \tag{4.12}$$

$$L(1 - c_{ii'}) + L\,(1 - o_{ii'}) + s_{i'} \geq s_i + p_i \qquad \text{for } \begin{matrix} i = 1, \dots, n \\ i' = i+1, \dots, n \end{matrix} \tag{4.13}$$

With L as a large enough number to make the constraint always true. In this case: the end time of the project.

Dependencies can be implemented similarly to the constraints that remain after elimination of terms 1 and 2 in constraints 4.12 and 4.13: $s_{i'} \geq s_i + p_i$ with i' as the dependent and i as the dependency.

This solution solves the problem of the instance growing quadratically with the number of timeslots, but the influence of the number of activities has increased. MIP Model 3 contains constraints for pairs of activities causing a quadratic increase in constraints and variables whenever the number of activities increases linearly. Models 1 and 2 did not have this problem.

5 System Design

This chapter describes the design of the software system and discusses the design choices made. Section 5.1 describes the overall structure of the software system, which can be separated in two components: *Server* and *Web App*. Section 5.2 describes the Server component and section 5.3 the Web App component.

5.1. Overall Design

A good design for a software system can help to reduce development costs in the future. For this project a modular approach is used to build the software system as it brings certain advantages. Using a modular approach, when a part of the system breaks, the other parts stay unaffected and remain functional. Another advantage is that parts can be replaced by better versions with the same functionality, without having to change all the code.

Taking a modular approach, the system consists of two main components, one component is responsible for generating a planning and the other component is responsible for visualising a planning. The first component we call *Server* and the second component *Web App*.

5.2. Server

The Server component is responsible for retrieving information from Odoo, generating a planning, storing both the retrieved information and the generated planning for later use and it has to communicate with the Web App component to visualise the planning. For this the Server will consist of 4 smaller components: Odoo Connector, Solver Connector, Database Connector and Web API. An overview of the different components and their interactions can be found in figure 5.1. The coloured blocks are the components of the Server and the white blocks are external pieces of software that our system interacts with. The Web App component is also represented by a white block as it is considered external from within the Server component.

5.2.1. Odoo

As the Server component is responsible for retrieving information from Odoo, a question is whether this component should be integrated in Odoo or separated from the Odoo environment. Being a module in Odoo makes it easier for users as they only have to install a single module to use the system. The system can be used without having to install software outside the Odoo environment, which follows the underlying thought of Odoo that all functionalities should be centralised inside a single environment. As this component is responsible for generating a planning, separating this component from Odoo enables its reuse in different

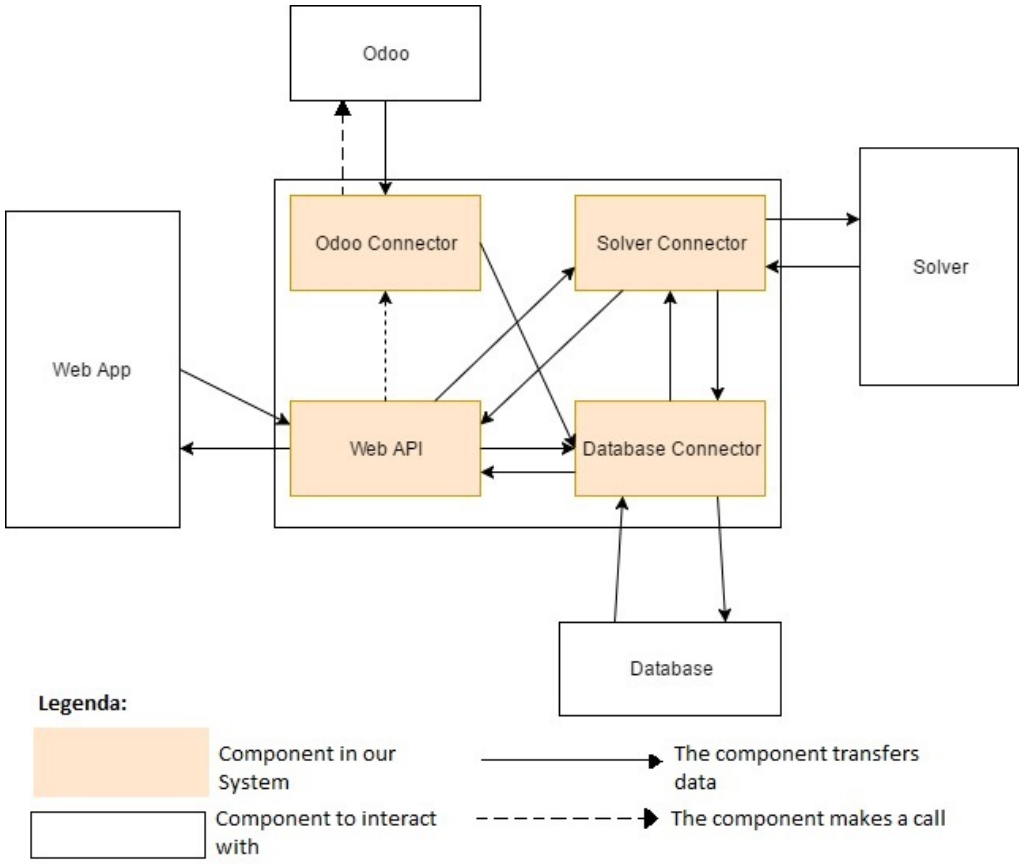


Figure 5.1: System Design Overview

business environments that do not make use of Odoo. An example is TNO, which provided us with the case problem used in this project. The separation of this component from Odoo also makes it possible to generate a planning on a machine different from the one running the Odoo environment. Allowing the use of different resources, so that both do not influence the performance of each other.

Together with the client was chosen to separate this component from Odoo. The advantages of separating this component from Odoo outweigh the advantages of integrating it within the Odoo environment.

5.2.2. Solver Connector

To generate a planning we formulated multiple models that can be solved by an algorithm. To solve the model there are two options: implementing an algorithm ourselves or using an external tool. Implementing an algorithm ourselves requires research into which algorithms can be used best to solve our model and it requires time for the implementation itself. As this project has a limited timespan, we chose to use an external tool, a solver, with the necessary algorithms already implemented.

The Solver Connector component is responsible for the communication with the solver. The solver will get the problem information from the Solver Connector, solve the problem model (MIP Model) we provided using the problem information and returns the results to the Solver Connector. There are many different solvers available for solving MIP problems of which some have been mentioned in the research report (appendix M). As there can be a big difference between the time each solver needs for solving the problem and the price of the solver, we decided that the software system should support multiple solvers.

Supporting multiple solvers gives the planner more control over the software system, they can decide to choose the cheap solver to reduce the usage costs or to choose the best solver to reduce the time required for solving. To provide the support of multiple solvers we decided to use an abstraction layer as a part of the Solver Connector. Using an abstraction layer it is no longer possible to use solver specific functionalities, like configuring the solver for a specific problem to reduce the time required for solving. The advantage of being able to choose which solver to use outweighs the loss of solver specific functionalities as the cost of the software system is an important factor to consider when choosing to make use of it.

Instead of creating our own abstraction layer, we made a comparison (appendix F) between different existing abstraction layers, which can be found in table 5.1. For testing the software system we use the solver Gurobi, a state-of-the-art MIP solver, which is why the support of Gurobi is taken into account for the comparison.

Name	License	Windows	Linux	macOS	Supported Solvers	Gurobi Support	Python Support	Quadratic Constraints
AMPL	Proprietary	Yes	Yes	Yes	10	Yes	Yes	Yes
CMPL	GPLv3	Yes	Yes	Yes	5	Yes	Yes	No
OR-Tools	Apache 2.0	Yes	Yes	Yes	7	Yes	Yes	?
JuMP	Mozilla 2.0	Yes	Yes	Yes	16	Yes	Not native	Yes
PuLP	Free	Yes	Yes	Yes	6	Yes	Yes	No
Pyomo	BSD	Yes	Yes	Yes	6	Yes	Yes	Yes

Table 5.1: Solver Abstraction Layers Comparison

The abstraction layer we chose is called *PuLP*. JuMP supports the most solvers but does not have native python support and requires programming in an additional programming language called *Julia*. AMPL supports, after JuMP, the most solvers but it is proprietary software and attempts to install OR-Tools resulted in problems that could not be solved without aid from its developers. The comparison table shows that Pyomo is a better candidate

than PuLP as it also supports quadratic constraints, but Pyomo had only been discovered near the end of the project. There was not enough time left to change from PuLP to Pyomo.

5.2.3. Database Connector

To store data a choice has to be made for how it should be stored. We considered two possible options: storing the data using a relational database or storing the data using a list of files.

Choosing for a relational database means that a database has to be chosen, created, configured and maintained. From experience we know that this is costly in terms of time, also as none of us has lots of experience with doing this. Choosing for a list of files only requires a structure for storing the data to be defined.

The intended usage of the software system is for demonstration purposes to show the capabilities of smart planning in cooperation with the Odoo environment. The costs, in terms of time, for using a relational database outweigh the benefit in this situation that can be gained from it, making us choose for a list of files.

Before storing the data it is useful to understand the relationship between the different kinds of data. The data necessary for the case problem can be divided into 5 types: Activity, Person, Planning, Project and Qualification.

Activity
- Release Time
- Processing Time
- Due date
- Delay penalty weight
- Required amount of persons
- Required qualification
- Mandatory assigned persons
- Activity Dependencies
- Mandatory starting time

Project
- Start Time
- End Time
- Persons
- Activities

Qualification

Person
- Qualifications

Planning
- Assignments
- Starting Times

Figure 5.2 gives an overview of the relationships between these 5 types of data in the form of an Entity-Relationship (ER) Model.

To store the data in files, we considered the options of storing the data using an XML structure or to store the data as JSON objects. The Web App component makes use of JSON objects to keep the data internally, storing the data as JSON objects makes the translation of the data between the Server and Web App components easier. Because of this reason we chose to use the JSON structure.

In the future the type of data storage might be changed. To make it easier to switch to a different type of data storage without having to rewrite lots of code, we use an interface between the Database Connector and the data storage. To make it easier to change to a relational database, a possible database model is included in appendix I.

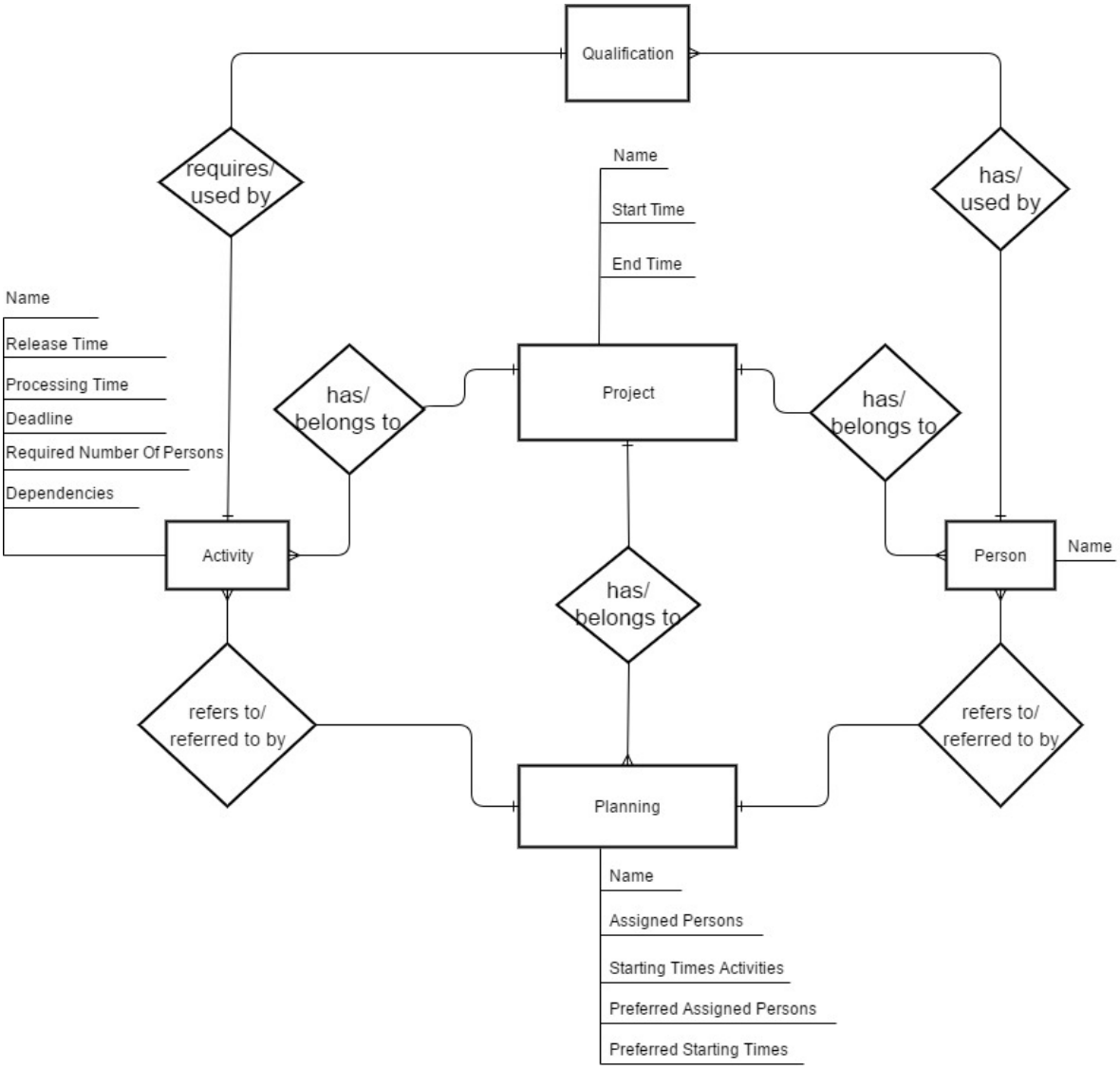


Figure 5.2: Entity Relationship - Model

5.3. Web App

The Web App component is responsible for visualising a planning. For this component the same two options apply as for the Server component: integrating it as a module in Odoo or separating it from Odoo.

The advantage of integrating the visualisation in the Odoo environment is that users of Odoo can keep the visualisation of their planning inside the Odoo environment.

The advantage of separating the component from Odoo is that the system can still be used in business environments, that do not make use of Odoo. Separating the component from Odoo does not mean that if in the future a decision is made to integrate the system in Odoo, that everything has to be developed again. The visualisation interface can be integrated using an iframe, adding minimal development costs.

Together with the client was chosen to separate the Web App component from Odoo. The separation makes it possible to use the system in business environments that do not make use of Odoo. Also, if chosen to integrate the system with Odoo in the future the extra development costs are minimal. The advantages of separating this component from Odoo outweigh

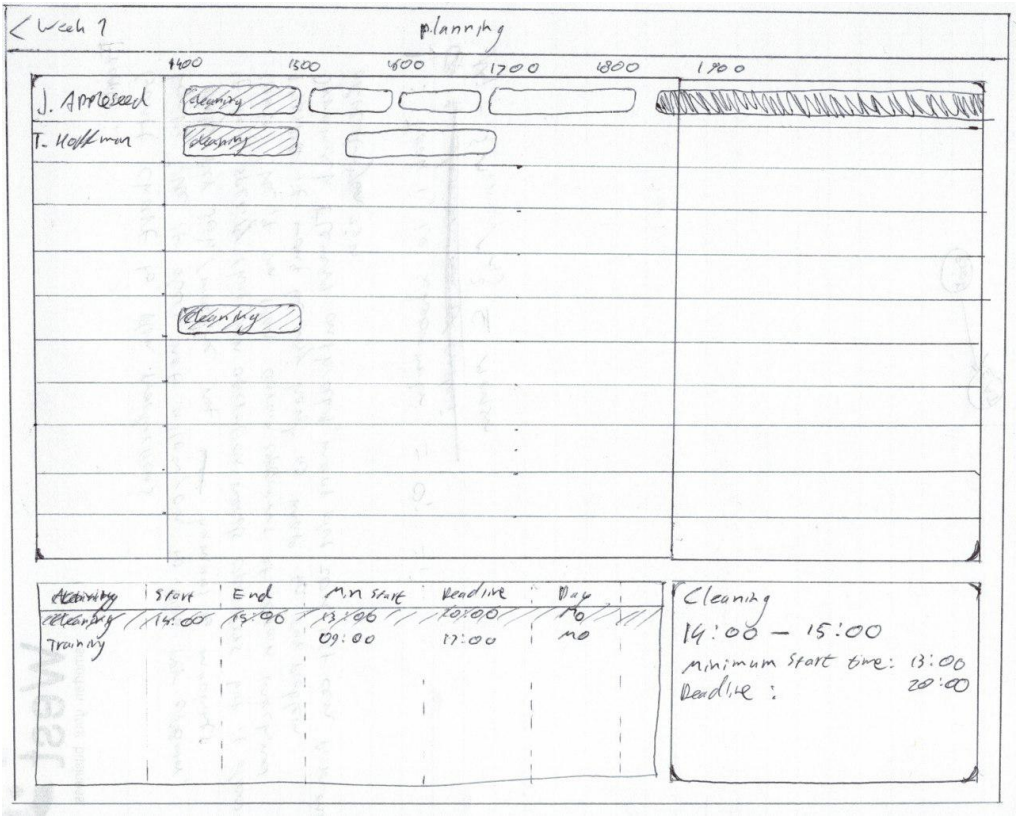


Figure 5.3: Planning - Drawing

the advantages of integrating it as a module.

5.3.1. Visualisation

To get an idea of what the visualisation could look like, we made drawings of a possible interface. In the drawing of the planning visualisation, a Gantt Chart is used. We chose to use a Gantt chart for the visualisation as it shows the most useful information. A problem with this is that activities that are not assigned to anyone, or not scheduled yet, cannot be shown in the Gantt chart. For this reason we also added a table that shows an overview of all activities. Figure 5.3 shows the visualisation of the planning.

A planning belongs to a bigger concept, called a Project, to which also the activities and persons used in a planning belong. Just the visualisation does not tell much about the persons themselves, to solve this we designed a page for a project overview. This page should provide information about the activities and the persons belonging to the project. This will give the planner a better understanding of the project the planner is creating a planning for. Figure 5.4 displays the design for this page.

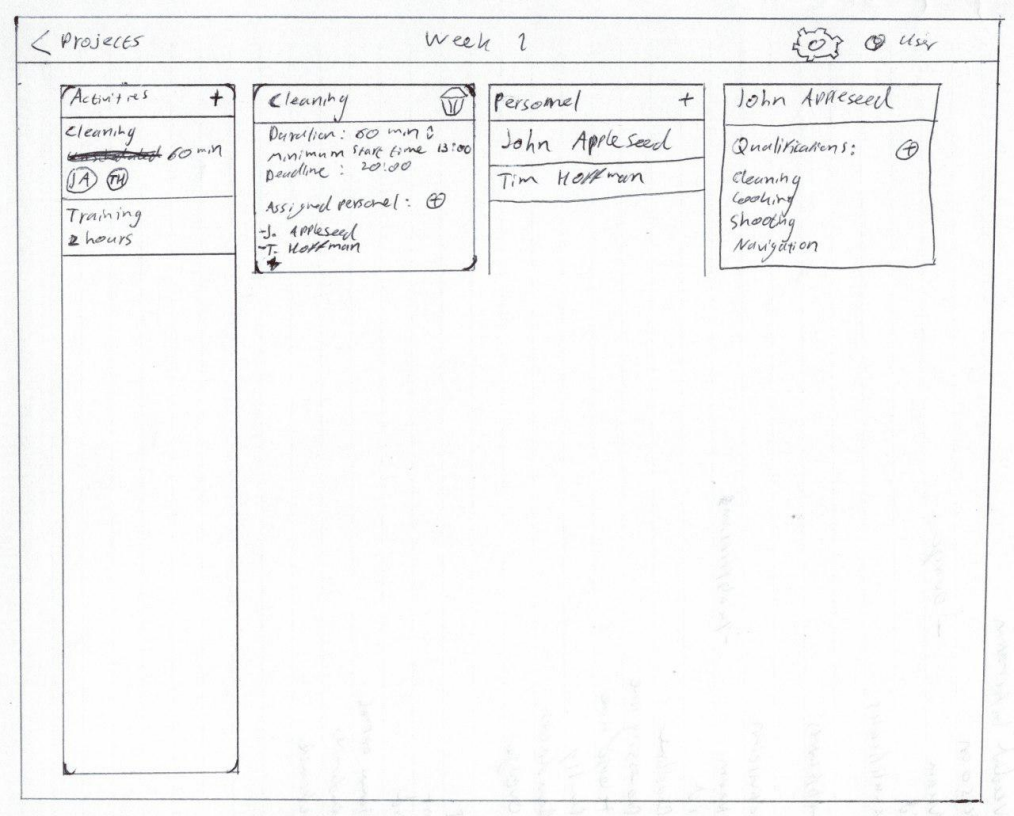


Figure 5.4: Project Details - Drawing

6 System Implementation

In this chapter the implementation of the software system is described. Section 6.1 describes which functionalities specified in section 3.3, have been implemented. Section 6.2 describes the object models used in the system and section 6.3 describes the overall system architecture. Section 6.4 describes the Web App and the Server is described in section 6.5. Each component of the server is explained in its own subsection: data storage 6.5.2, odoo connector 6.5.3, solver connector 6.5.4 and the web API 6.5.5.

6.1. Developed Functionalities

In section 3.3 an overview of the functional requirements is given, which are prioritised using the MoSCoW model. From the list of requirements, all requirements in the category 'Must Have' have been implemented and some requirements in the category 'Should Have' are implemented. The number in brackets used in this section refers to the requirements in section 3.3.

The system is able to automatically generate an initial planning and to visualise this planning, it allows the user to make manual modifications in this visualised planning and to automatically generate a new planning, taking the manual modifications into account. It is possible to create a planning for a project imported from Odoo. The generated planning can afterwards be saved to the server and used later on.

Must Have Requirements

To allow an initial planning to be created, requirement 1, multiple components of the server work together, which are all described in section 6.5.

To visualise a planning, requirement 2, a Gantt Chart and an activity detail view are created, both described in section 6.4.2.3.

To save a planning, requirement 3, a button in the interface is added and a component *data storage* as part of the server is created, which is described in section 6.5.2.

To manually modify the planning, requirement 4, we added different ways to interact with the interface, like a drag-and-drop functionality, which are described in section 6.4.2.3.

Should Have Requirements

To allow the automatic retrieval of persons, requirement 5, and the automatic retrieval of tasks, requirement 6, from Odoo, an import page is added to the interface which is described in section 6.4.3 and a component *Odoo connector* is created as part of the server, which is described in section 6.5.3.

To inform the user when a planning violates constraints, requirement 9, we added a visual indication in the interface for when a person is assigned to an activity they are not qualified for. This is described in section 6.4.2.3. We consider this requirement to be partially implemented. In section 10.1 we explain which other functionalities should be added before we consider this requirement to be implemented.

To allow a planning to be generated which takes the manual modifications into account, requirement 10, the functionalities added for requirement 4 are used together with constraints defined in the problem models, which are described in chapter 4.

In addition to requirements that are implemented, we created some interfaces that allow the user to select projects and view project details. These functionalities are explained in detail in sections 6.4.2.1 and 6.4.2.2 respectively.

6.2. Object Models

In section 5.2.3 an entity relationship model is described. During the implementation of the software system, a derivative of this model was used. This derivative consists of a project, planning, activity and person.

6.2.1. Project

A project has the following properties:

- Start time: The start time of the project
- End time: The end time of the project.
- Unique id: A unique identifier
- Name: The name of the project to display to users.
- Activities: A collection of activities that have to scheduled.
- Persons: A collection of persons that can be assigned to perform activities.
- Plannings: The collection of different plannings.

It is possible for projects to have multiple plannings. These plannings represent different possibilities for scheduling activities and assigning persons to activities.

6.2.2. Planning

A planning has the following properties:

- Unique id: A unique identifier;
- Project id: The identifier of the project this planning belongs to;
- Name: The name of the planning to display to users;
- Starting Times: The starting times for the activities in the project of this planning;
- Assignments: The assignments of persons to activities in the project of this planning;
- Preferred Starting Times: The starting times for certain activities preferred by the planner;

- Preferred Assignments: The assignments of persons to activities preferred by the planner

6.2.3. Activity

An activity has the following properties:

- Unique id: A unique identifier;
- Name: The name of the activity;
- Release Time: The first time work on this activity can be started;
- Deadline: The time by which this activity should be completed;
- Processing Time: The time it takes to perform the activity;
- Penalty Weight: A weight per time unit to indicate how bad it is for the activity to be completed after the deadline;
- Number of Required Person: The number of persons required to perform the activity;
- Required Qualification: The qualification persons should have to be able to perform the activity;
- Dependencies: The activities that should be completed before work on this one can be started.

6.2.4. Person

A person has the following properties:

- Unique id: A unique identifier;
- Name: The person's name;
- Qualifications: The qualifications the person has.

6.3. System Overview

Our software system consists of three main components; the web app, the server and an external solver. The web app and server communicate with each other through a Web API, which means that the back end of the application does not have to be hosted on the same server as the web app.

Figure 6.1 shows how the components of our application communicate with each other. The web app and server communicate with each other through endpoints and components within the server communicate with each other through method calls. Dashed lined arrows indicate commands, for example, the HTTP Server sends the 'Import Project' command to the Odoo connector. Solid arrows indicate data flows, for example, the Solver Manager returns generated solutions to the web API. The web API specifications are described in detail in appendix A

The web app consists of multiple pages that allow the user to view data that is stored on the server, modify existing data, or import new data from Odoo. The web app is described in detail in section 6.4.

6.4. Web App

To allow the user to interact with the system we created a web application. This web application allows the user to open a project, look at project details, related plannings and

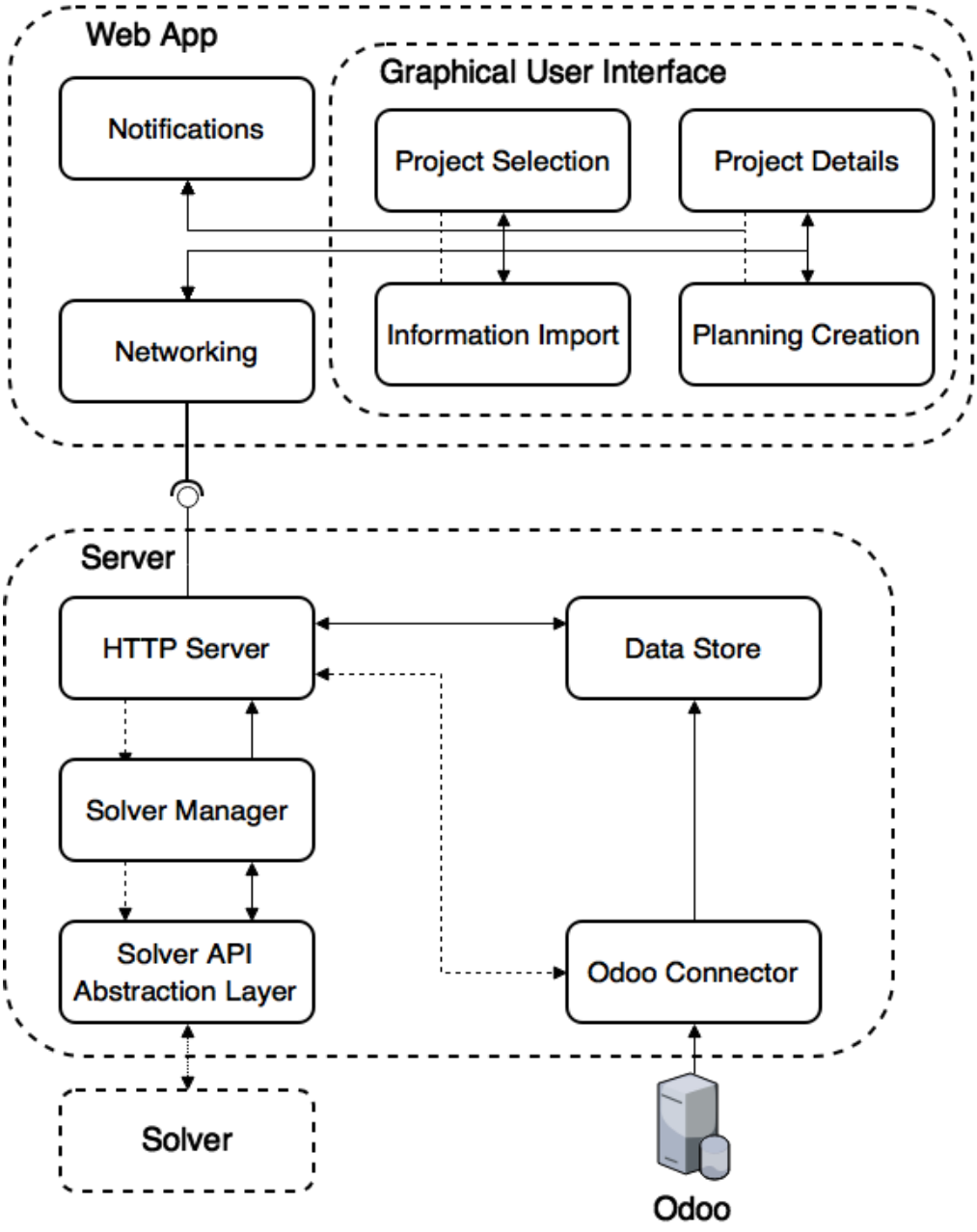


Figure 6.1: System Architecture

import new projects from Odoo. This section describes how the user interface components communicate with each other and with the web server.

6.4.1. Architecture

Creating a web app is not an easy task and maintaining a consistent state within the web app is not trivial when many components need be able to mutate state and display data. For this reason we decided to make use of the JavaScript framework VueJS that manages most information synchronisation for us. An additional benefit of using VueJS is that it allowed us to build different interface components separately through templates.

The interface components each have their own internal data model and view controller logic. These components can be composed of standard html elements and other custom child components. The child components' data models can partially be configured by their parent components through the use of properties. If children want to mutate their properties, they must notify their parent of the change that has to occur. The idea behind this is that, since the parent sets its childrens' properties, it has the most knowledge of the data model. Additionally this allows children to be reused across different parent components as long as the parent component has the correct programming interface.

Components are created by extending the Vue base object. This object offers functionality for a diverse set of view and view controller related functions. This includes view controller life cycle management, computed properties, state synchronisation, templating and dynamic data binding.

For a detailed explanation of the concepts used in VueJS we would like to refer you to the VueJS documentation at: https://vuejs.org/v2/guide/.

As shown in figure 6.1 the web app has a networking component. This component is a plug-in for the Vue instance. Making this a plug-in allows us to access it from any Vue component. The networking plug-in is used by components to retrieve information from, save information on, and to send commands to the server.

6.4.2. Graphical User Interface

The user interface consists of multiple pages. In this section we describe each page and the involved implementation decisions.

Project Selection

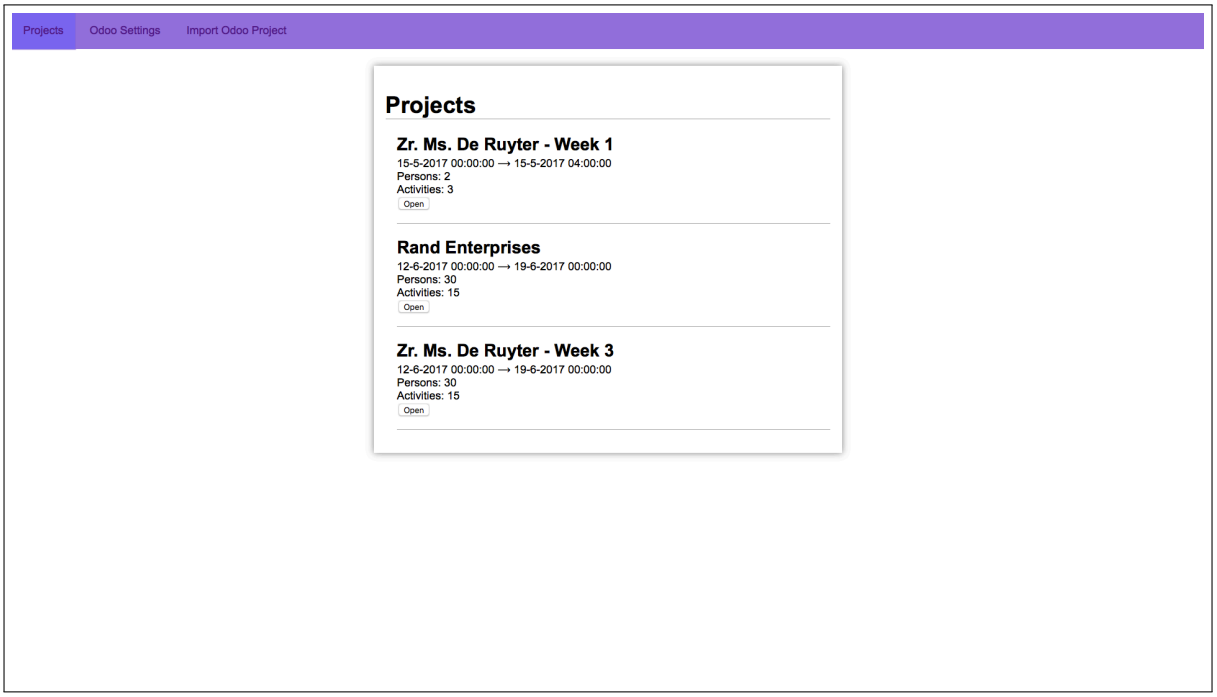


Figure 6.2: Project Selection Interface

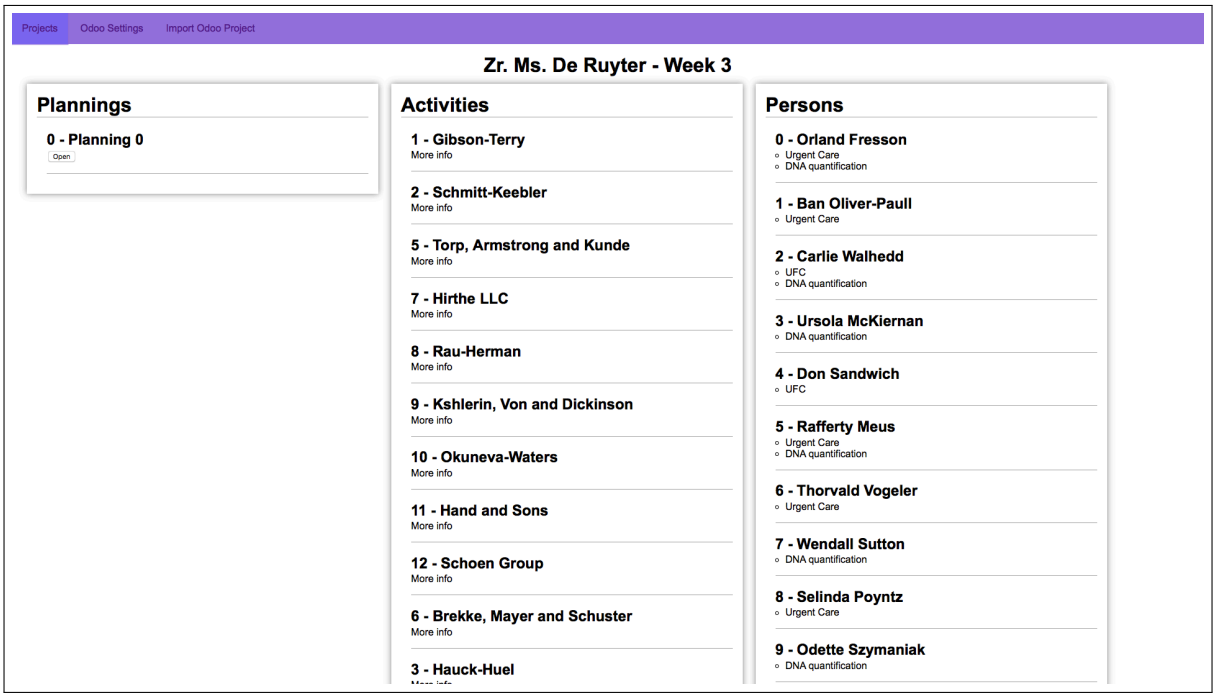


Figure 6.3: Project Details Interface

The project selection page (fig. 6.2) allows users to choose a project that is stored on the server. One of the properties of VueJS is that communication between pages is done through url parameters. Thus every piece of information that must be transferred from one page to another must be included in the url. For this reason, when the user tries to open a project, they are redirected to a url containing the project id.

Project Details

The project details page (fig. 6.3) allows the user to view project details. The details of a project include the persons' details, i.e. which qualifications they have, and the activities in the project. Additionally this page can be used to view the list of plannings in this project and to navigate to a planning.

Planning Creation

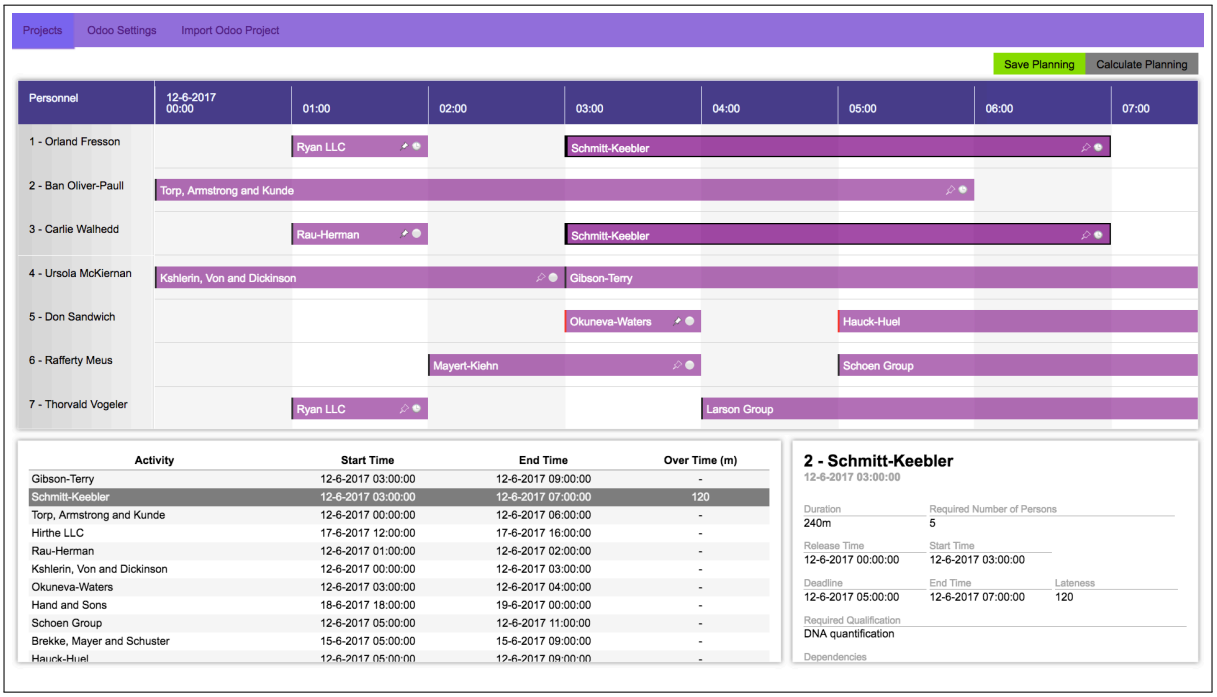


Figure 6.4: Planning Creation Interface

The planning creation page (fig. 6.4) allows the user to have a detailed view on a planning. The page consists of three main elements; the Gantt Chart, a table and a details pane.

The Gantt chart allows a planner to see activities that have been assigned to persons over a time line. Each row in the chart represents a person in the planning and each column represents an hour of time. The purple bars (fig. 6.5) in the person rows represent activities that have been assigned to the person. The user is able to pan around the Gantt chart so they can see more persons or a different time frame.

Each bar in the Gantt chart has two buttons that allows the user to pin an activity to a person or to pin the activity at a certain time. Figure 6.5d shows an activity that has been pinned to the person is it currently assigned to. Figure 6.5c shows an activity that has been pinned at the time it is currently scheduled at.

In addition to the pinning controls, the bar also has an indicator that colours red (fig. 6.5f) when the person it is currently assigned to is not qualified to perform it. To adjust the planning, the user can use drag and drop to move the activity elements to other times or other persons.

(a) Default

(b) Selected

(c) Pinned Time

(d) Pinned Person

(e) Pinned Person and Time

(f) Unqualified Assignment

Figure 6.5: The Gantt Element Styles

The table in the lower left corner of figure 6.4 allows for easy selection and finding of activities in the planning. The first column shows the activity name, the second column the starting time, the third column the ending time and the final column the number of minutes the activity will be completed after its deadline. If the user selects an activity in this table, the Gantt chart will automatically scroll to the activity in the Gantt chart. This allows the user to see the activity in context.

Additionally, when the user selects an activity in either the table or Gantt chart, the activity detail pane (fig. 6.6) will update and show the selected activity's following details:

- Starting time;
- Processing time;
- Release time;
- Number of required persons;
- Required qualification;
- Dependencies;
- Currently assigned persons;
- Lateness

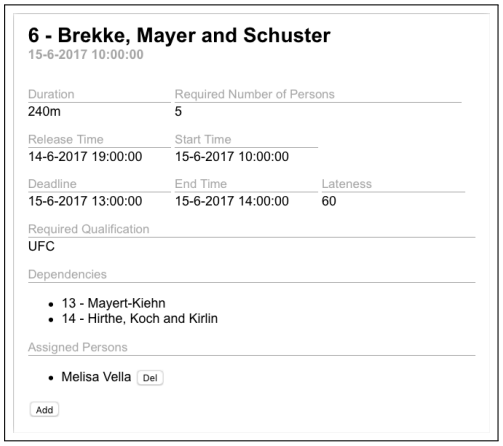


Figure 6.6: Activity Detail

Not only does the activity detail allow the user to view detailed information, it also allows the user to edit the starting time and the assigned persons. The starting time can be edited by clicking on the time. The label then turns into a date picker where the user can edit the starting time. Assigned persons can be edited through a drop down list and a removal button.

The planning creation page also contains two buttons, one for saving the planning and one for generating a new planning.

6.4.3. Information Import

The interface to import projects from Odoo into the application consists of two pages. The first page allows the user configure the Odoo connection configuration. If the user tries to save the configuration, the application first checks whether the configuration is valid, i.e. the password, username, url and database name. The second page is similar to the project selection page, but the displayed projects are projects in Odoo that can be imported. After the project is imported, the user is automatically redirected to the project details page belonging to that project.

6.4.4. Notifications

Every page has the ability to display notifications by calling a method on the Vue plug-in VueToast[1]. Figure 6.7 shows an example notification.

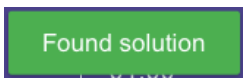


Figure 6.7: Toast Notification

6.4.5. Networking

For the web app to communicate with the web api we have created a Vue plugin 'Networking'. This module allows us to call networking methods from any Vue instance. The networking module has three main methods: 'sendCommand', 'retrieveObject' and 'saveObject'. The 'sendCommand' method sends a command to the web API using the HTTP POST method

[1] https://github.com/AStaroverov/vue-toast

and should return a status code that represents whether the command was successful. The 'retrieveObject' method calls the web API using the HTTP GET method and returns the requested object if it exists. The 'saveObject' makes a PUT request to the server and asks the server to save the included object on the server.

The HTTP requests are made using the JavaScript Fetch API. This API prevented us from getting caught up in a so called 'callback hell', since it makes use of Promises instead of callbacks. A promise is an object that "promises" to either resolve with the requested response or reject if an error occurs.

To make making requests easier in the view controllers, we created separate methods for making requests to endpoints. The endpoints are defined in an environment variable file, so updating the API endpoints should require as little work as possible. The set up used even allows a develop version to use different API endpoints than the production version.

6.5. Server

The web app is not useful without any data to display; a back end is needed that can provide the data and execute commands given by the user through the interface. This section describes how the server components work and how they work together. First, section 6.5.1 describes the overall server architecture. Second, section 6.5.2 describes how data is serialised and stored to disk. Third, section 6.5.3 describes how the the system imports information from Odoo and stores it. Fourth, section 6.5.4 describes how the solver manager translates the internal data model to constraints and how it communicates with solvers. Finally, section 6.5.5 describes the HTTP server and the corresponding web API.

6.5.1. Architecture

As explained in section 6.4, West IT is a vendor of the Odoo software package. Because Odoo is implemented in the Python programming language we chose to implement our back end in Python as well. West IT using Python is not the only reason we have chosen for Python, a member of the team was quite familiar with Python and had good experience with the web server library Flask for Python. Additionally, many MIP solvers offer a Python API which would allow us to interact easily with different solvers. In the end we ended up choosing an abstraction library that handles interfacing with different solvers for us.

The server hosts multiple services that can be used by the user via the web app.

- Data Store: A place where data can be stored and retrieved;
- Odoo Connector: A connection with Odoo to retrieve and store projects;
- Solver Manager: A component that manages the connection with a solver;
- Web API: An HTTP server that hosts the web API endpoints for the web app;

The data store is currently implemented as a component that stores and reads JSON files to and from disk. Other components can retrieve data from the store using project and planning ids, or by requesting all information available.

The Odoo connector makes use of the XMLRPC API that Odoo provides. This API allows for reading tables in the Odoo database through queries. The Odoo connector converts the fetched data to the internal data model.

The solver manager retrieves data from the data store, and enters it into the solver API abstraction layer. After it has done this, the solver can be started and a planning can be generated.

The HTTP Server hosts multiple end points that can be used by the web app to, among others, start the solver, save a planning and to instruct the server to import an Odoo project.

6.5.2. Data Store

As described in section 5.2.3, we have designed an ER model for our system. To use this model in our application, we created the classes: 'Project', 'Planning', 'Activity' and 'Person'. These classes represent the same entities as described in section 5.2.3, so they are not repeated here. A class diagram describing the exact classes can be seen in figure J.6.

To store and retrieve data within our application we created the 'DataStore'. This data store is an abstract class that defines an interface that can be used by any kind of data storage. This interface is agnostic to how storage is implemented. In our case we chose to implement a data store as a JSONDataStore. This JSON data store assumes that objects are JSON encodable. We made sure our objects are JSON encodable by creating an SHPlannerDataObject super class that handles JSON encoding. All subclasses should implement the function 'from_json_dict' to decode JSON dictionaries into objects. Objects should also include replacement keys if they want their variable names to be different in the encoded JSON objects. A specification for the JSON Objects can be found in appendix G.

6.5.3. Odoo Connector

As explained in section 2.1, West IT resells the software package Odoo and a big part of the assignment was that some integration with Odoo had to be made. As a proof of concept we created the possibility to import projects from Odoo into our application. After importing the project, our software can generate a planning for the project.

To make accessing and importing Odoo projects easy from other parts of our application we created the 'OdooConnectorManager'. This manager provides an interface to control importing Odoo projects, but hides implementation details. The manager allows other modules to access the current connection configuration, retrieve importable projects and to import specified projects.

Within the Odoo Connector we store connection details, like database location, name, user name and password, in a json file. To allow storing and retrieving the configuration we created an extension for the standard DataStore called ConfigurationDataStore.

To retrieve information from Odoo, we make use of the XMLRPC API Odoo provides. This API allows us to retrieve data from specified tables and access them as standard Python objects. To access information from the protected parts of Odoo, like the tables where actual information is stored, authorisation is required. To remove the burden of handling the authorisation from the parts that retrieve and parse the information from Odoo, we created the OdooConnection class. This class handles the authorisation with the Odoo instance and automatically refreshes the connection when it expires.

The OdooDataConverter parses the Python dictionaries the Odoo API returns and turns them into the internal data model objects described in section 6.2.

6.5.4. Mixed Integer Programming Solver Connection

This section describes how the internal data model is translated into a MIP model instance, entered into a solver and how the generated solution is translated back into a sensible format.

The solver manager manages planning generation state. It keeps track of all requests that have been made, handles threading and solver state management. To prevent the main server thread from being blocked by a solver, the solver manager creates a thread for each request and allows for observing solver state.

To allow the solver manager to be agnostic to what kind of model has to be built, an abstract class has been created that defines a clear interface that model builder can implement. By implementing this interface, the solver manager can control the model builder by calling methods that start the solving process, abort it or retrieve the status.

During the project, two model builders were implemented: One for MIP model 2 (section 4.2) and one for MIP model 3 (section 4.3). MIP model 1 was not implemented in PuLP as it contained a quadratic constraint, which PuLP does not support.

During initialisation of the server, the model builder that should be used can be specified. The differences between the models are explained in chapter 4 and the difference in performance are explained in section 7.1. These model builders were implemented using the PuLP library to allow them to use different kind of solvers. The reasoning behind the usage of the PuLP library is explained in section 5.2.2 in more detail.

6.5.5. Web API

The web API was set up using the Python library Flask. Unlike Django, which is more of a framework, Flask is comparatively small. The easy access for testing utilities and simple API, which was sufficient for the REST API endpoints, made Flask the better choice.

On initialisation, the ‘__main__. py‘ injects all dependencies into the web server. These dependencies include the data store, the solver manager and the Odoo connector manager. This allows programmers to easily switch MIP models, MIP solvers or data stores without having to change other code. As long as the newly injected objects respect the defined API, no problems occur.

The endpoints we set up allow the web app to manage the importing of Odoo projects, the managing of the solver and basic data retrieval and storage. In appendix A the web API is described in detail with a description for each endpoint describing what methods, parameters and data they accept, what data they return and what status codes can occur.

7 System Analysis

This chapter analyses the system implementation discussed in chapter 6. Section 7.1 analyses the performance of the implemented MIP models as described in chapter 4. The architecture of the product is analysed in section 7.2.

7.1. Performance

During this project the choice was made to use a Mixed Integer Programming solver in order to find a schedule. In order to use this solver, three models were formulated, each slightly different from one another. These differences most likely result in different runtimes and maximum instance sizes.

We define the total runtime to be the time it takes to build a model plus the time it takes to find a solution. This is defined as $\text{total}(t) = \text{build}(t) + \text{solve}(t)$

7.1.1. Hypotheses

From the models defined in chapter 4 we have constructed the following hypotheses:

Hypothesis 1: total(Model 3) < total(Model 2). We expect model 3 to perform better than model 2, because model 3 uses continuous time instead of binary variables. This results in fewer decision variables that have to be constructed and taken into account. The second reason is that model 3 uses continuous variables instead of discrete variables, which should allow the solver to find a solution faster.

Hypothesis 2: solve(Model 2) < solve(Model 1). Model 2 a linearised version of model 1 and as linear constraints are easier to solve than quadratic constraints for solvers, we expect the solver to find solutions for the same instances faster using MIP model 2 than for MIP model 1.

Hypothesis 3: $\lim_{t \to +\infty}$ solve(Model 3) < solve(Model 2). MIP model 3 uses continuous variables for starting times instead of discrete time slots as MIP models 1 and 2 do. For this reason we expect MIP model 3 to scale better when the time span of the project increases.

Hypothesis 4: build(Model x using gurobipy) < build(Model x using PuLP). We expect the build time to be less for all models when they are implemented in GurobiPy than when they are implemented in PuLP, as there is one fewer conversion step.

Hypothesis 5: solve(Model x using gurobipy) = solve(Model x using PuLP). We expect the amount of time needed for finding a solution to be equal for all models whether they are implemented in GurobiPy or PuLP, as the models and the solver are the same.

7.1.2. Test Setup

To analyse the runtime of the models, we created multiple instances that vary in the number of activities, persons and time they have. The used instance parameters can be found in table 7.1.

There are two types of tests we have run. The first test consists of entering the instances into the solver through the abstraction layer. The second test circumvents the abstraction layer and uses the Gurobi Python interface (GurobiPy) directly. We ran these two different types of tests, because we wanted to find out how much the abstraction layer slows down the process. During the tests we measured how long it took to build the model, either directly in GurobiPy or through PuLP, and how long it took Gurobi to find a solution.

Because Model 1 (sec. 4.1) contains quadratic constraints, it was not possible to implement it in PuLP and thus no tests were performed for it using PuLP.

The tests were run on a computer running Windows 10 Pro with 16GB of Corsair® Vengeance LPX CMK16GX4M2A2133C13R RAM, an Intel® Core i5-6600K processor and a Crucial® MX200 2,5" 500GB SSD.

Table 7.1: Instance Values

Instance	Activities	Dependents[1]	Dependencies[2]	Persons	Days	Skills
persons_3	15	2	2	3	7	2
persons_15	15	2	2	15	7	2
persons_30	15	2	2	30	7	2
persons_45	15	2	2	45	7	2
persons_55	15	2	2	55	7	2
persons_75	15	2	2	75	7	2
activities_15	15	2	2	15	7	2
activities_75	75	2	2	15	7	2
activities_150	150	2	2	15	7	2
activities_225	225	2	2	15	7	2
activities_275	275	2	2	15	7	2
days_1	15	2	2	3	1	2
days_3	15	2	2	3	3	2
days_7	15	2	2	3	7	2
days_10	15	2	2	3	10	2
days_14	15	2	2	3	14	2
days_21	15	2	2	3	21	2

[1] Number of activities with dependencies.
[2] Maximum number of dependencies for an activity with dependencies.

7.1.3. Varying Input Size

For the full experiment results, please see the table in appendix K. This section describes the tests that were run to validate hypothesis 1, 2 and 3. These tests were only run using the GurobiPy interface as using PuLP should not yield different results in solving time.

Time span

As can be seen in figure 7.1, model 1 and 2 are very dependent on the amount of time, whereas model 3 seems to be unaffected. This is in line with our hypothesis. We can explain

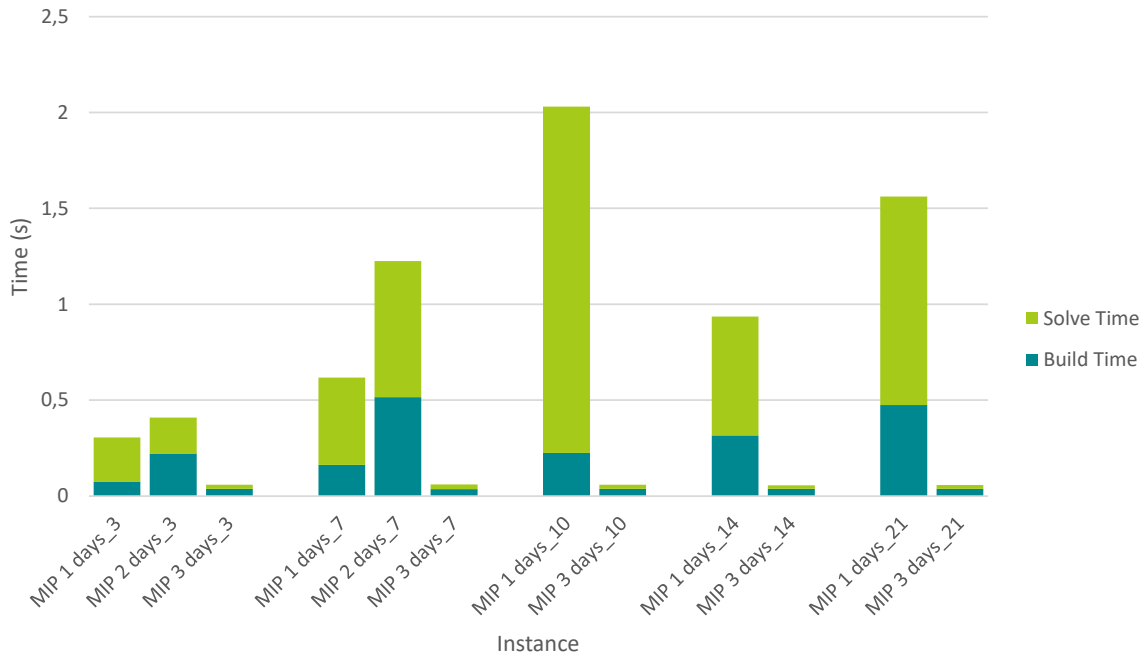


Figure 7.1: Runtimes for Time Varying Instances

this because of the way time is modelled in model 1 and 2 compared to model 3. Model 1 and 2 use a binary matrix that contains an entry for every timeslot and activity. This means that the matrix grows linearly with the amount of time and linearly with the number of activities, thus the matrix grows quadratically. Model 3 uses continuous variables to model time, which means that the number of variables only grows linearly with the number of activities. Since the number of activities is kept constant, the time it takes to solve the instance using model 3 remains fairly constant.

When the timespan of the planning reaches 10 days, model 2 no longer works properly. The solver starts giving warnings that the model is numerically unstable, which means that it cannot calculate solutions with enough precision and classifies the instance to be infeasible. Model 1 and 2 use a constraint with a power of two to make sure that activities cannot start before their dependencies. This leads to very large numbers, which in this case lead to problems. We expect that the problem occurred with model 2, but not with model 1, because model 2 has more constraints next to the large numbers.

Another unexpected result is that the instance with 14 days took less time to solve than the instance with 10 days. A possible explanation for this is that the solver realised it could use a different, faster, approach to find a solution. A different possible explanation is that, since more time is available for scheduling the same number of activities, the problem becomes easier and thus takes less time to solve. This is supported by the fact that the build time did not decrease.

The results from this test support hypothesis 1 and 3, but hypothesis 2 seems to be incorrect. Apparently the larger number of constraints in model 2 is a bigger problem for Gurobi than the quadratic constraints in model 1.

Number of persons

Figure 7.2 shows the results for the test for which the number of persons varied. From the results we can conclude that model 2 is more sensitive to the increase in the number of persons than model 1 and 3. Model 3 seems to not be affected at all.

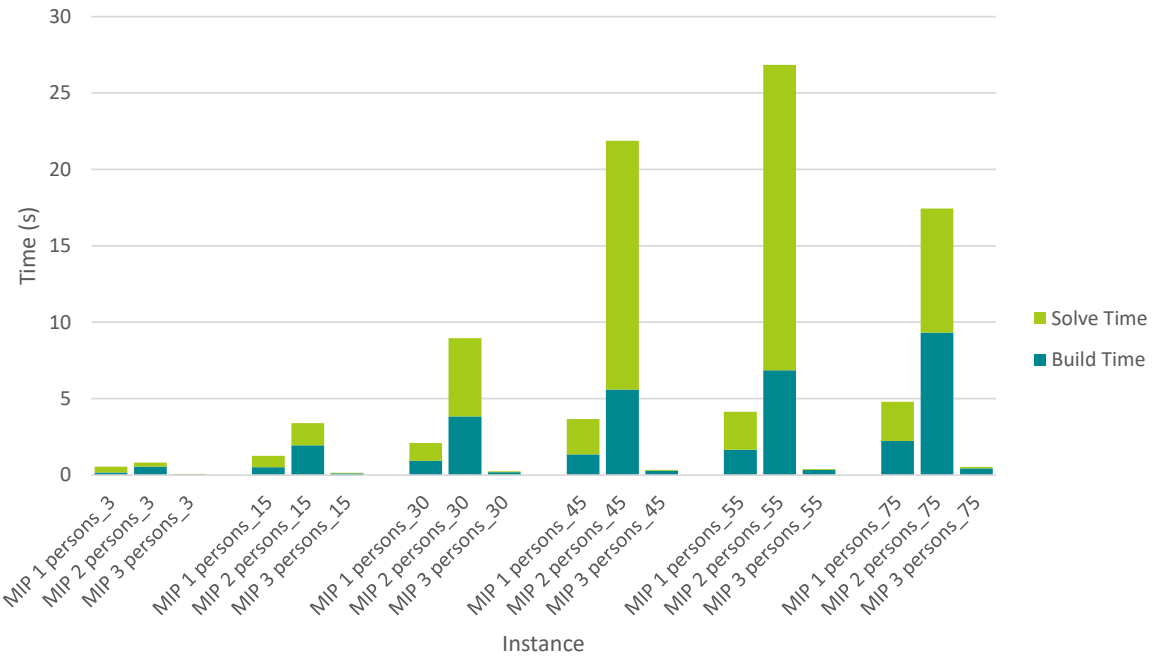


Figure 7.2: Runtimes for Person Varying Instances

Just like in the test where the timespan was varied, model 2 seems to perform worse than model 1, which we did not expect.

Unlike the test where time was varied, model 3 does seem to be influenced by the varying number of persons. It also looks like all models grow linearly in the number of persons.

An unexpected result is that the solve time seems to decrease for model 2 for the instance with 75 persons. This could be explained by the fact that creating a planning becomes easier when there are more persons to carry out activities.

This test supports hypothesis 1, however it seems to support that hypothesis 2 is incorrect.

Number of activities

Figure 7.3 shows the results for the tests where the number of activities is varied. In the results it looks like model 3 is faster than model 1 and 2, but it also seems to suggest that the solve time for model 3 grows more strongly than the solve time for model 2. When the number of activities doubles, the solve time for model 2 quadruples, however the solve time for model 3 becomes almost 70 times higher. This can be explained by the fact that model 3 has a quadratic amount of constraints in the number of activities. One might wonder how much longer model 3 is faster than model 2.

Just like the tests where the amount of time was varied, model 1 starts to become numerically unstable with large numbers. The solver also gave this warning for model 2, thus it might be possible that, even though model 3 grows much faster than model 2, it would be impossible to use model 2 for a bigger number of activities.

This experiment supports hypothesis 1, but it, again, seems to disprove hypothesis 2.

There is another interesting observation we made. One of the activity instances we created, turned out to be infeasible, which led to an interesting result. Figure 7.4 shows that model 1 and 2 allow the solver to figure out quite quickly that an instance is infeasible, finding this within 70 seconds at most. Model 3, however, ran for more than 30 minutes and still did not

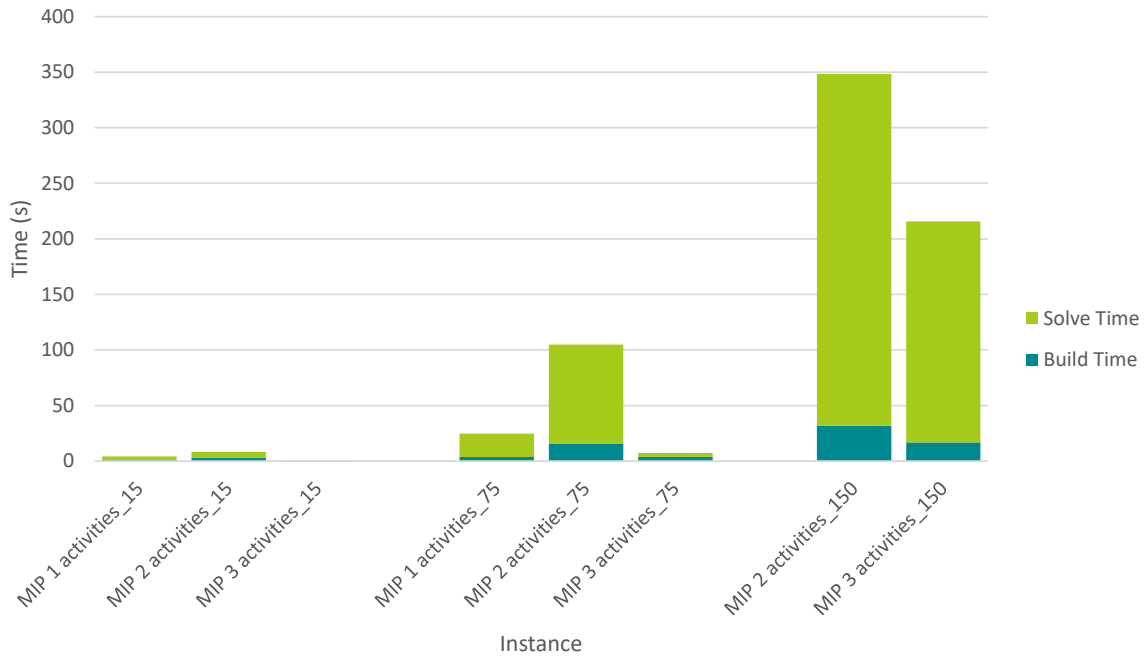


Figure 7.3: Runtimes for Activity Varying Instances

figure out that the instance was infeasible. We stopped the solver after 33 minutes, because it did not look like a useful result would be found.

The fact that model 3 takes so long to find out that an instance is infeasible can be explained by the fact that there is a practically infinite amount of possibilities when continuous variables are used and the solver will try to find a solution that works. There is only a limited number of possibilities for the binary models 1 and 2, and the solver seems to find out quickly that there is no solution.

This observation seems not to align with hypothesis 1.

7.1.4. Comparing PuLP and GurobiPy

This section compares the runtimes between the GurobiPy implementation and the PuLP implementation. The tests allowed us to draw conclusions for hypothesis 4 and 5.

Using an abstraction layer allows the user to switch between solvers without having to change any code, but this comes at an cost. The abstraction layer requires additional build time and memory. Another disadvantage, due to the abstraction in between, is that certain solver specific capabilities are not always available. For example, quadratic constraints or multiple objectives that Gurobi supports, are not supported by PuLP and thus cannot be used.

In order to quantify these costs, we have run some tests. The tests consist of running the varying activities instances using model 3. The results are visualised in figure 7.5. As can be seen, building without the use of PuLP is always faster, which makes sense as there are fewer steps before the solver can start looking for solutions. Unfortunately, these results are not entirely representative. The solve time for the PuLP implementations includes the time to convert PuLP's data model to Gurobi's data model and we did not find a way around this. This data did not affect the conclusion that can be drawn about the build time for the PuLP model being higher than the build time for the GurobiPy model, which is in line with hypothesis 4.

By comparing the results in figure 7.5 we can see that the solving times are mostly equal

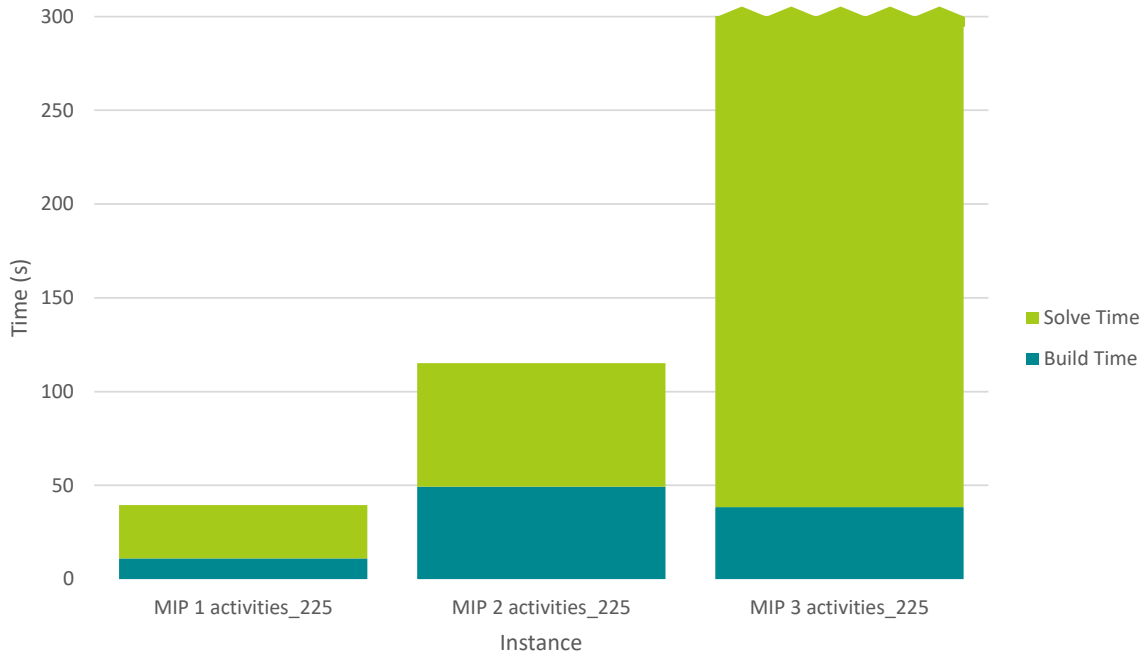


Figure 7.4: Runtimes for an Infeasible Activity Instance

to each other. For activities 75, the total time to find a solution is smaller for PuLP than it is for Gurobi. After looking at the solver logs, we found out that for this instance, the PuLP implementation is able to find a heuristic that it can use successfully. The GurobiPy implementation does not seem to find the possibility to use this heuristic. We believe this explanation does not mean that hypothesis 5 is incorrect, as the other results indicate it to be correct.

Building with Gurobi directly is quicker in every single case and can immediately call to the solver.

The solve time for PuLP is not entirely representative. When calling solve on a model, PuLP first has to convert the internal data model to Gurobi, adding some hidden build time within the solving time.

7.1.5. Conclusion

Looking at our hypothesis, we can conclude that hypothesis 3, 4 and 5 are correct. It looks like hypothesis 2 is incorrect, which means that the number of constraints is more important for Gurobi than whether there are quadratic constraints in the model.

Hypothesis 1 is interesting. For varying the amount of time and persons it holds, but when the number of activities increases, the time to solve grows very fast. An interesting future experiment might be to find out at which point model 2 becomes faster than model 3. Though, it might happen that model 2 becomes so numerically unstable that it starts failing before model 3 start to take more time than model 2.

Another interesting conclusion we can make is that model 3 takes much longer to find out that an instance is infeasible than models 1 and 2. It might be possible to run model 1 and 3 simultaneously if it is not obvious whether an instance is feasible. If model 1 indicates that the instance is infeasible, model 3 can be stopped, and if model 3 gives a planning, model 1 can be stopped. This could give the best of both worlds.

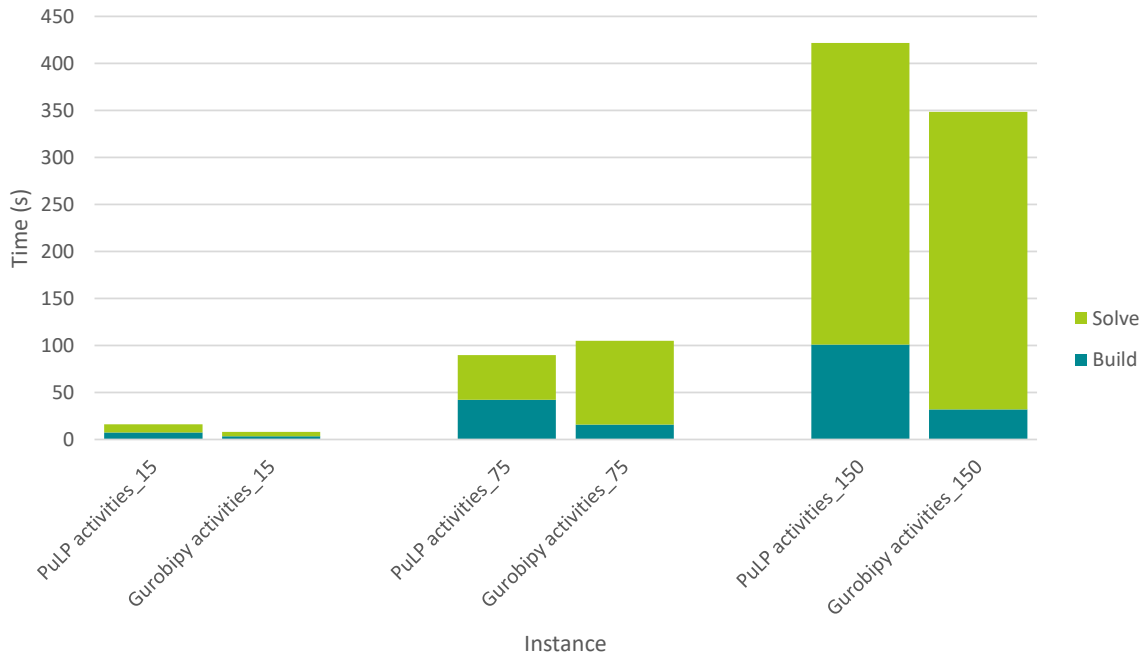


Figure 7.5: Runtimes for Varying Activities for Gurobi and PuLP Implementations

7.2. Architecture Analysis

This section analyses the quality of the produced system by explaining what components can use improvement or components we believe are well designed.

As explained in chapter 6, our software system consists of two main components, the server and the web app. We will first discuss the web app and then discuss the server.

7.2.1. Web App

As explained the web app consists of pages that are composed of components. The implementation of pages like project selection, project details and Odoo information import is quite trivial as these pages only display data. The real interesting components of the web app user interface is the planning creation interface, as this is the page where the user can interact with plannings by mutating or viewing them.

The planning creation page (section 6.4.2.3) consists of multiple subcomponents, the Gantt chart, the activity table, the activity detail pane and save and solve buttons. Currently these components communicate with each other through their parent component, the PlanningCreation object. The PlanningCreation object defines a clear interface that child components can use to mutate the planning object they display. To make the planning mutation interface more easily maintainable and extendible, we propose replacing the interface we defined with a VueX[1] data store. This store would define an interface similar to that in the PlanningCreation component, however it would require much less communication between parent and child components. This decrease in communication needed between components eases the burden of making sure communication between components is done correctly and should improve testability as only component, the data store, has to be mocked instead of all communication channels as is done right now.

In addition to the PlanningCreation page, we also believe the Networking module could use some improvement. It currently contains many different methods used by the many parts

[1] https://vuex.vuejs.org/en/

of the application to call the web API on the server, however we suspect splitting it up into submodules could improve maintainability.

7.2.2. Server

The server consists of a few main components. We believe most components are quite well constructed, but there is one component we believe could be improved. Currently the web API Server has one method for each endpoint. We think it should be split up into submodules that handle different kinds of requests. A way this could be done would be to create submodules for Odoo requests, data storage requests and solver management requests.

8

Testing & Validation

In the previous chapter the software system has been analysed on performance and quality, this chapter discusses how the software system has been tested and validated. Section 8.1 discusses how unit tests were handled. Section 8.2 discusses UI tests and tests for the problem models are described in section 8.3. Finally, section 8.4 validates that the project was completed successfully.

8.1. Unit Testing

To confirm the expected behaviour of methods, unit tests have been written for all non-interface code. For the interface code we performed different tests (8.2). The unit tests confirm the expected behaviour and take into account possible edge cases. If a bug was discovered during the development process, a new unit test was created to confirm the further absence of this bug.

To increase the confidence in the software system, we defined a requirement to have a minimum test coverage, for non-interface code, of 85%. The test coverage at the end of the project was 90.7% for the server and 91.7% for the web app which thus fulfils our requirement.

8.2. UI Testing

For the user interface we make use of VueJS (section 6.4.1). VueJS allows the developer to connect methods with parts of the user interface. We consider these methods with logic as units that require unit tests to confirm their expected behaviour. For testing the user interface we performed manual tests in which we confirmed the working of all elements of the user interface.

Using the weekly feedback of the client, parts of the user interface changed multiple times. As the tests also have to change for every change of the user interface, the costs for automating these tests in the timespan of this project outweigh the benefit they provide.

8.3. Planning Generation Tests

When you want to automatically generate a planning, it is important to know whether the problem model used for generating the planning is correct. For each of the problem models described in chapter 4 test instances were created. Each model had one test that covered all aspects of the case problem that had to be taken into account and several other tests to verify the correctness of the individual aspects of the case problem.

The results of these tests have been manually verified and the conclusion deduced from these results is that all of the problem models are correct.

8.4. Validation

Testing the software system increases the confidence that it works as expected, but acceptance tests are needed to validate whether the software system is the product the client wanted. To validate the product, acceptance tests were held during the weekly client meetings. The acceptance tests consisted of a demo to show the new functionalities added in the past week, a confirmation that they were what the client had asked for, the possibility for the client to try the system himself and a moment of feedback from the client on the system. This short feedback loop allowed to us quickly change direction or change the priorities of new tasks for the next week.

A meeting was also held with representatives of TNO to demonstrate the product, explain the research that has been performed using their case problem and to gather feedback about the product. They were impressed with the product and the research performed during the project and looked forward to the final report for the project. The positive feedback increased our confidence in the product being what the client was looking for.

In the research report (appendix M) we defined a set of success criteria to validate the success of the product of which we gave a short overview in section 2.4. We now discuss each success criterion and whether it was met.

The solution meets all 'Must Haves'

The first success criterion requires all 'Must Haves' from the functional requirements (section 3.3), to be implemented. In section 6.1 we discussed that all these requirements have been met, which means that this success criterion has been met.

The solution meets most 'Should Haves'

The second success criterion requires most 'Should Haves' from the list of functional requirements (section 3.3) to be implemented. From the 6 requirements that belong to this category, only 3 have been implemented, which are requirement numbers: 5, 6 and 10.

Requirement 7 requires the automatic retrieval of qualifications from the Odoo system. During the project it became clear that the standard Odoo installation does not support something like qualifications. This functionality can be added to Odoo through the use of an extension. In deliberation with the client it was decided to only use the standard Odoo installation, which makes this requirement unrealisable.

Requirement 8 requires multiple suggestions for a planning to be given. During the project the client prioritised which functionalities should be added first, leading to the most valuable functionalities to be added before the less valuable ones. Because of the limited timespan of the project and the lower prioritisation of this requirement, there was not enough time left to implement this functionality.

Requirement 9 requires to inform the planner when a planning in the interface violates constraints. There are multiple possible violations of constraints, of which one is implemented (section 6.1). This requirement has been partially implemented, because of the limited timespan of the project.

We conclude that because one of the requirements became unrealisable and one has been partially implemented, that this success criterion has still been met.

The solution is properly documented

The third success criterion requires that documentation has been made that describes the system architecture, how components work together, the external interface, how the endpoints for the server have been defined and that describes how the different components work internally.

To describe the system architecture and how components work together, an overview is been created which describes the communication between the different components, see section 6.3.

To describe the external interface, a document is written which defines each endpoint used on the server with its properties. This document is included as appendix A.

To explain how the different components work internally, multiple class diagrams have been created that explain the communication of the different classes in a component. These class diagrams have been included as appendix J.

We conclude that all documentation to understand the software system has been created and that this success criterion has been met.

SIG feedback is positive

The fourth success criterion requires positive feedback from SIG. In the first SIG feedback a total score of 4 out of 5 was achieved. The SIG feedback is discussed in section 9.3.

The feedback from the first SIG assessment has been taken into account and used to improve the maintainability of the software system. For the second SIG feedback session different quality checks were performed, which makes us expect positive feedback in the second SIG assessment. As this feedback will be given after the deadline for this report, we cannot take it into account for concluding success or not. As we expect positive feedback for the second assessment, we conclude that this success criterion has been met.

The solution is tested well

The fifth success criterion requires the software system to have a test coverage of 85%. In section 8.1 we explained that the current code coverage is 90.7% for the server and 91.7% for the web app, which means that this success criterion has been met.

9 Development Process

This chapter discusses the development process. Section 9.1 describes the development methodology used. Section 9.2 describes the tools used during the development process and section 9.3 covers the feedback from SIG and how we have used it.

9.1. Development Methodology

During the project we made use of Scrum, an iterative agile software development framework. Scrum works with predefined cycles of a specific length, which in our case consisted of a single week. At the end of every sprint, we chose a set of tasks we aimed to perform during the next sprint. Because our sprints were quite short, we had a small feedback cycle with the client. This allowed us to make quick adjustments to the following week's planned tasks without it disrupting the overall planning too much.

To keep track of the client's wishes, a backlog of user stories was kept. This backlog was prioritised and gave us some extra support for what was preferred to be in the final product. During the client meetings, demo's were given, which allowed the client to give feedback. Additionally, the sprint plan for the next week was discussed and adjusted in deliberation with the client.

9.2. Development Tools

During the project we made use of different tools to help us maintain the quality of the project but also to keep track of what was done and what not during sprints. We mention the tools shortly and explain the advantage it gave us.

Trello

Trello is a collaboration tool that helps in keep a project organised. It makes use of a Kanban board that consists of different columns. Our board consisted of the columns: backlog, this week, busy, testing, needs review, and done. The order of these columns is representative of the typical flow for developing new functionalities. New functionalities are represented through cards that can be assigned to persons. This tool allows developers to keep track of their project, similar to how a Scrum board does.

GitHub

GitHub is a code hosting platform for version control and collaboration. It makes it possible to keep the code you are working on remotely accessible for multiple persons. Additionally,

GitHub offers project management tools like Pull Requests and Issues.

Travis CI

Travis CI (Travis) is a continuous integration service to build and test software projects. When new functionalities are added to a software product, Travis helps finding integration problems by building the product in a clean environment. If problems occur developers are notified, so that they can fix the problems as soon as possible.

CodeCov

CodeCov is a tool to help identify which part of the codebase has been tested and which not. Instead of doing this after multiple functionalities have been added, CodeCov measures the code coverage of individual pull requests. This helps to keep the codebase well tested and helps avoid having to test lots of code later on.

9.3. SIG Feedback

During the project there are 2 moments when the code of the software system is evaluated on maintainability, one in the middle of the project and one at the end. As the final feedback will arrive after this report is submitted, only the first feedback will be discussed.

9.3.1. Analysis

The first feedback we received is given here:

> De code van het systeem scoort 4 sterren op ons onderhoudbaarheidsmodel, wat betekent dat de code bovengemiddeld onderhoudbaar is. De hoogste score is niet behaald door een lagere score voor Unit Size.
>
> Het eerste wat opvalt is de duidelijke indeling van de code. Hierdoor wordt de architectuur onmiddelijk duidelijk, en wordt het ook makkelijker om bepaalde functionaliteit te vinden in de code.
>
> Voor Unit Size wordt er gekeken naar het percentage code dat bovengemiddeld lang is. Het opsplitsen van dit soort methodes in kleinere stukken zorgt ervoor dat elk onderdeel makkelijker te begrijpen, te testen en daardoor eenvoudiger te onderhouden wordt.
>
> In jullie geval zou je bijvoorbeeld Planning.from_json_dict kunnen verkleinen. Deze methode bevat nu twee keer dezelfde code voor het parsen van een lijst met timestamps, je zou dat stuk naar een aparte methode kunnen verplaatsen en die vervolgens vanuit from_json_dict kunnen aanroepen. Dit is natuurlijk geen wereldschokkende verbetering, maar omdat jullie qua score al vrij hoog zitten kom je op dit soort kleine refactorings uit.
>
> Tot slot is de aanwezigheid van test-code veelbelovend, hopelijk zal het volume van de test-code ook groeien op het moment dat er nieuwe functionaliteit toegevoegd wordt.
>
> Over het algemeen scoort de code bovengemiddeld, hopelijk lukt het om dit niveau te behouden tijdens de rest van de ontwikkelfase.

9.3.2. Improvements

Taking the feedback from SIG into account, we made some improvements to the code. These improvements concern themselves only with Unit Size, as this is the only aspect that did not get the full marks by SIG.

According to the analysis of SIG, the method *from_json_dict* is too long and it would be best if it were to be split into multiple smaller units. We added a new method for parsing a list with

timestamps and a new method for parsing a list of integers. We also made a general method to test for types of a variable that allows for throwing a custom error message and returns the variable if from the correct type.

For the second SIG feedback moment we performed a quality check on the code of the software system again, taking into account the problems identified in the first analysis. The improvements made during this check hopefully lead to an increase in the score for maintainability.

10 Discussion

This chapter discusses what additions can be made to the product, reflects on the project and concludes this report. Section 10.1 discusses how the product can be extended in the future and makes recommendations on what should be given priority. Section 10.2 reflects on the project and discusses problems encountered during the project and how they could be avoided. Finally, section 10.3 concludes the report and the project.

10.1. Future Work

For further development of the product, we describe which features we believe it can be extended with next. Section 10.1.1 describes features we believe can be added to the user interface. Section 10.1.2 describes features that can be added to the server. Next, section 10.1.3 describes possible enhancements for the model. Finally, in section 10.1.4 we make recommendations to West IT on what aspects of the product can best be improved or extended next.

10.1.1. Web App

Although much work has already been done for the user interface, we believe more features and improvements can be made. First, a list of trivial functions that do not require much explanation is given. After that, features that require more explanation are described.

The project selection page can be improved with the following features:

- Create a new empty project;
- Project creation/modification date.

The project details page can be improved with the following features:

- More details for activities;
- Project detail modification, like ending/starting time;
- Planning creation/modification date;
- Create a new planning;
- Project Details modification, like persons and activities.

The planning creation page can be improved with the following features:

- Style improvements;
- More elaborate conflict detection;

- Visual indications in the Gantt chart for dependencies;
- Visual indications in the Gantt chart for release times and deadline;
- More ways to modify plannings.

In addition to the improvements already described, we believe the following features will also be helpful.

The application allows multiple plannings per project to exist, however there currently is no easy way to compare two plannings. A nice addition to the interface would be to show two plannings at once so the user can compare them easily. An example implementation would be to have two Gantt charts on screen at once. One displaying one planning version and the second displaying another version. Selecting an activity in one, automatically scrolls the activity in view in the other Gantt chart.

Finally, we think it would be nice to have the possibility to enter project/planning data through the interface instead of Odoo being the only possibility to enter data.

10.1.2. Server

As the user does not directly interact with the server, it is harder to think of features that could be added it. However, we believe that a deeper integration with the Odoo software package is a function that could be added. Currently our software product can retrieve data from Odoo, however this data can currently not be exported back to Odoo. The Odoo API allows entering information into Odoo using the API, thus we believe this would be a nice addition.

Another improvement that could be made for the server are adding basic security features. Currently the server does not make use of SSL, encryption or any kind of protected storage. Also, the Odoo login credentials are not stored in a safe way, as it is just stored in a JSON file.

Finally we believe that the JSON data store should be replaced with a real database once larger amounts of information are stored.

10.1.3. Model

This section describes five enhancements for the used Model.

First we believe the objective function can be extended to include the preference of finishing activities as soon as possible. Currently our objective function only takes into account that activities should be finished before their deadlines, as it is not important when activities are completed on a ship, as long as they are completed before their deadlines. In a production environment it may be preferable to finish tasks as early as possible. To achieve this the objective function could be replaced with minimisation of the makespan, i.e. the difference between the latest completion time and earliest start time.

Second, the model considers the preferences of the user to be inflexible. This means that if the user accidentally makes the planning infeasible, the solver will just tell the user no planning is possible. To introduce this enhancement, the MIP model wouldn't have to be extended very much. The objective function could be extended by introducing a penalty when an activity is started after or before the preferred start time. This way the solver would try to adhere to the user's preference, but it cannot be a cause for the schedule being infeasible.

Third, when the user makes an adjustment to the generated planning and generates a new one, the new planning could be completely different with only the user's preferences kept the same. A possible enhancement for the MIP model would be to introduce a penalty when the new planning deviates a lot from the previous planning. This would mean many more input variables, though.

Fourth, following from the functional requirement described in section 3.3, multiple planning versions could be generated and proposed to the user. This is not really an enhancement for the MIP model, but for the implementation of the solver manager. Certain solvers, like Gurobi, allow the user to specify how many solutions they want to find.

Finally, the planning a solver generates will usually be as optimal as possible, but there is no way to explain to the user why a planning is good. MIP models are hard to understand and it can also be hard to understand why a solver chooses a certain planning over another. We suspect this is an area of Mixed Integer Programming that is still being researched quite heavily.

10.1.4. Recommendations for Future Development

To conclude this section we would like to make recommendations to West IT on what functionality to develop next. The biggest shortcoming in our application at the moment is the usability. Most features described in section 10.1.1 improve the ease of usage. Things like creating a new empty planning right from the project details page make it easier for the user to use the application. We would also recommend the implementation of conflict detection. We believe the challenge in implementing this is on the user interaction front and not on the algorithm front as this is usually just a trivial comparison. For the server side we would strongly recommend security improvements, as the focus of our project was not on security, but on presenting a prototype of the planning generation.

10.2. Reflection

During the project we made different choices, in this section we discuss our experience with the choices made.

MIP

During the project we chose to use Mixed Integer Programming to model our problem. We looked at other problems for which solvers and algorithms are available, but most problems missed a few key components, which would make a mapping very complex. At the end of the project we found that using Mixed Integer Programming had been a great choice. It allowed us to express our problem in a way that was not too hard to understand and was very flexible. Using Mixed Integer Programming in this project taught us a new technique to model problems and find solutions and as this kind of modelling can be applied to almost any problem, it might help us in the future with other projects.

Additional Research

The product we made is capable of automatically generating a planning. The performance of generating this planning depends, among other factors, on how the problem model is formulated. During the development phase we chose to put more time in additional research for the problem model, to increase the performance of the product. This additional research also includes experimental comparison of improvements of the problem model to identify weak points and thus parts to focus on. This additional research resulted in less time being available for implementing new functionalities. Although the functionalities the client asked for are implemented, it might have been interesting to continue with a less-optimal problem model and to see how much more functionalities could have been added to extend the product. We do believe that we were justified in choosing to spend more time on research, as it did improve the performance of the model and because we found it very interesting to compare the models experimentally.

Code reviews

We consider the quality of the software we deliver to be an important aspect, which is why we chose at the beginning of the project to perform strict reviews on all code added. These strict code reviews do increase the quality of the product, but also take a considerable amount of time to perform. Because of the time required for the code reviews and possible modifications, it occurred that merging features was delayed during a sprint because of the code review. Instead of aiming for the best code quality possible, using a lower standard for code quality is a possibility. This might make adding features faster at the risk of ending up with a lower level of code quality.

Technical Debt

In the design phase of the project we created an ER model that describes 5 different entities: Activities, Persons, Planning, Project and Qualification. At the start of the development phase, we chose to delay the implementation of the project entity as, at the time, it would add little extra value. When we wanted to add some features concerning projects, we needed to implement the project entity to keep a logical structure in the system. This required modifications in the system as it had been built using a different structure. Looking back, it would have been better to use the project entity from the start, as refactoring the code to include it, took a considerable amount of time.

Web App

Our choice to create a web app instead of a native application allowed us to make use of knowledge we already had, which sped up the development process for the interface. We also believe that creating a web app was the right choice, because West might want to embed the application in Odoo. The choice to use VueJS also turned out very well, as it took care of most synchronisation between interface components.

10.3. Conclusion

In the research report (appendix M) a list of evaluation criteria for both the project and the product are given. In section 8.4 the evaluation criteria for the product have been discussed. Following from this we can conclude that the success criteria for the product have been met.

The following are the success criteria for evaluation the project:

- The deadlines during the project have been met,
- The client and coach are pleased with the performed work,
- The process is properly documented,
- The students have met the educative goals of the project,
- The solution fulfils its success criteria.

At the start of the project we made a planning to meet all deadlines. We have delivered all deliverables in time and can conclude that this criterion has been met.

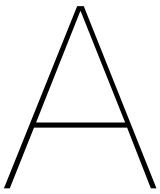
Towards the end of the project we had a discussion with our client about what he thought of the delivered product. He was pleased with the delivered work and sees that a lot of work and effort has been put into the project. However, he had hoped that the interface would have error detection, but he understands that there wasn't enough time. West is interested in further developing the product to demonstrate even more possibilities to their customers.

Throughout the project, the process should be documented to be used for analysis or evaluation. The aspects to be documented can be found in the research report, see appendix M. A plan of action has been made at the start of the project, after the research phase a research

report has been handed in and after each meeting *Meeting Minutes* have been written to be used by us for evaluating the current state of the project. For each sprint a sprint plan and sprint retrospective have been written with the latter one being written to improve ourselves throughout the project. This final report is the last of the required documents to be written and by that concludes that this criterion has been met.

This final report should indicate that we can carry out an entire software development process with success, that we can effectively choose a development strategy and execute a development process according to that strategy and that we can present a compete and convincing explanation of the development process and the product results. The research report should indicate that we can establish the necessary quality requirements for a product and carry out the tests necessary to determine whether the product fulfils these requirements. From this we can conclude that we have met the educative goals of the project and that this success criterion has been met.

As we earlier concluded that the success criteria for the product have been met, all the success criteria for both the project and the product have been met. We can conclude that this bachelor project has been completed successfully.



Web API

A.1. Project

A.1.1. Retrieval

URL `/api/project/[<int:id>]?`

Parameter A comma separated list of ints that are id's for projects. If this parameter is not provided, all projects are returned

Method GET

Body -

Returns JSON project object if the id was specified, or a list of all projects if the id was not defined

Status Codes 200 Projects returned

400 The specified project ids were not a comma separated list of ids

404 None of the specified project ids were found

A.2. Planning

A.2.1. Retrieval

URL `/api/project/<int:project_id>/planning/[<int:id>]?`

Parameters A comma separated list of ints that are id's for plannings. If this parameter is not provided, all plannings in the project are returned

Method GET

Body -

Returns A JSON dictionary containing the keys 'projects' and 'plannings'. The key projects contains the project object and the key plannings key contains the asked plannings, or an empty list if they were not found

Status Codes 200 Plannings returned

400 The specified planning ids were not a comma separated list of ids

404 None of the requested plannings were found

A.2.2. Storage

URL /api/project/<int:project_id>/planning/<int:id>

Method PUT

Body JSON Planning object

Returns The planning id used to store the planning on the server

Status Codes 201 Planning stored
400 Invalid planning object
500 Planning object was valid, but storage failed

A.3. Solver

A.3.1. Starting

URL /api/project/<int:project_id>/planning/<int:id>/solve/start

Method GET, POST

Body -

Returns -

Status Codes 202 Solver started
404 Unknown planning id
412 (Solver not available: not used in practice)

A.3.2. Aborting

URL /api/project/<int:project_id>/planning/<int:id>/solve/abort

Method GET, POST

Body -

Returns -

Status Codes 200 Solver stopped
404 Unknown planning id

A.3.3. Status

URL /api/project/<int:project_id>/planning/<int:id>/solve/status

Method GET, POST

Body -

Returns {message, planning:JSONPlanningObject if message == 'done' and a solution is possible
}

Status Codes 200 Solver status available
404 Unknown planning id

A.4. Odoo

A.4.1. Configuration

Retrieval

URL `/api/odoo/config`

Method GET

Body -

Returns Configuration object

Status Codes 200 Configuration returned

404 Odoo connection not configured

Setting

URL `/api/odoo/config`

Method PUT

Body Configuration JSON object

Returns -

Status Codes 201 Configuration stored

400 Invalid configuration object

403 Invalid configuration

500 Configuration was valid, but storing configuration failed

Validation

URL `/api/odoo/config/test`

Method PUT

Body Configuration JSON object

Returns -

Status Codes 200 Valid Configuration

400 Invalid configuration object

403 Invalid configuration

A.4.2. Project

Projects to Import

URL `/api/odoo/projects`

Method GET

Body -

Returns A list of projects available to importing

Status Codes 200 Projects returned

403 Odoo connection not (properly) configured

Import Project

URL `/api/odoo/projects/import/<int:id>`

Method POST

Body -

Returns The internal project id

Status Codes 201 Project imported

404 Unknown project id

B

Polynomial Algorithm for Verifying Schedules

Algorithm 1 A Polynomial Algorithm to Verify the Correctness of Schedules

```
1:  cost ← 0                                                              ▯ O(1)
2:  for i in A′ do                                                        ▯ O(n^J)
3:      cost ← cost + ω_i · max{c_i − d_i, 0}                             ▯ O(1)
4:  if cost > K then                                                      ▯ O(1)
5:      return No
6:
7:  for i in A′ do                                                        ▯ O((n^J)^2 m)
8:      if |μ_i^c| ≠ m_i^r then                                           ▯ O(1)
9:          return No
10:     if s_i < r_i then                                                 ▯ O(1)
11:         return No
12:     if c_i − s_i ≠ p_i then                                           ▯ O(1)
13:         return No
14:     for i′ in pred(i) do                                              ▯ O(n^J)
15:         if s_i < c_i′ then                                            ▯ O(1)
16:             return No
17:     for M_l in μ_i^r do                                               ▯ O(m)
18:         if M_l ∉ μ_i^c then                                           ▯ O(1)
19:             return No
20:
21: for M_l in M do                                                       ▯ O((n^J)^2 m)
22:     for i in A′ do                                                    ▯ O((n^J)^2)
23:         if M_l ∈ μ_i^c and M_l ∉ μ_i^r and M_l ∉ μ_i then             ▯ O(1)
24:             return No
25:         for i′ in A′ where i′ ≠ i do                                  ▯ O(n^J)
26:             if M_l ∈ μ_i^c and M_l ∈ μ_i′^c then                      ▯ O(1)
27:                 if i and i′ are scheduled simultaneously then         ▯ O(1)
28:                     return No
29: return Yes
```

Algorithm 1 describes how a schedule can be verified to be correct. Some checks require some additional explanation. On lines 1 through 5 the algorithm checks whether the objective function is smaller than the specified value K. On lines 14 through 16, it is checked whether activity does not start before its dependencies have completed. On lines 17 through

19, the algorithm makes sure that all machines that were required to be assigned, are assigned. On line 23 the algorithm checks whether all machines that are assigned to i are either qualified, or were mandatorily assigned. On lines 25 through 28 the algorithm checks whether machines are assigned to two activities at the same time. The check whether activities overlap can be done in constant time by making a fixed number of comparisons between completion and starting times.

Assuming that $n = n^J + m$, the complexity of algorithm 1 is equal to $O(n^J) + O\left((n^J)^2 m\right) + O\left((n^J)^2 m\right) = O(n^3)$. This shows that verifying whether a schedule is valid can be done in polynomial time, thus our problem is a member of the complexity class NP.

C

Mixed Integer Programming Model Version 1

C.1. Indices and Sets

$i, i' \in A \ (i \neq i')$	Activity	(C.1)
$j \in W$	Person	(C.2)
$t \in \{1, \dots, T_n\}$	Time interval	(C.3)
A	The set of activities	(C.4)
W	The set of persons	(C.5)
T	The set of time intervals	(C.6)
$n = \lvert A \rvert$		(C.7)
$m = \lvert W \rvert$		(C.8)
$T_n = \lvert T \rvert$		(C.9)

C.2. Parameters

$r_i \in \mathbb{N}_0$	Release time for activity i	(C.10)
$p_i \in \mathbb{N}_{>0}$	Processing time for activity i	(C.11)
$d_i \in \mathbb{N}_{>0}$	Deadline for activity i	(C.12)
$\omega_i \in \mathbb{N}_0$	Penalty per time unit for missing deadline d_i	(C.13)
$r_{iW} \in \mathbb{N}_{>0}$	Required number of persons for activity i	(C.14)
$\delta_{ii'} = \begin{cases} 1 & \text{if } i \to i' \\ 0 & \text{Otherwise} \end{cases}$	i' is dependent on i	(C.15)
$\theta_{ij} = \begin{cases} 1 & \text{if } j \text{ can perform } i \\ 0 & \text{Otherwise} \end{cases}$	Whether j has the required skill to perform i	(C.16)

C.3. Decision Variables

$$b_{it} = \begin{cases} 1 & \text{if activity } i \text{ is scheduled at time } t \\ 0 & \text{Otherwise} \end{cases} \tag{C.17}$$

$$v_{ij} = \begin{cases} 1 & \text{if person } j \text{ is assigned to activity } i \\ 0 & \text{Otherwise} \end{cases} \tag{C.18}$$

C.4. Constraints

$$v_{ij} = 0 \qquad \text{if } \theta_{ij} = 0 \text{ for } i=1,\dots,n,\ j=1,\dots,m \tag{C.19}$$

$$v_{ij} = 1 \qquad \text{if preferred} \tag{C.20}$$

$$b_{it} = 0 \qquad \text{if } t < r_i \text{ for } t = 1,\dots,T_n \tag{C.21}$$

$$b_{it} = 1 \qquad \text{if preferred} \tag{C.22}$$

$$\sum_t b_{it} = p_i \qquad \text{for } i = 1,\dots,n \tag{C.23}$$

$$\sum_j v_{ij} = r_{iW} \qquad \text{for } i = 1,\dots,n \tag{C.24}$$

$$\sum_i b_{it} v_{ij} \le 1 \qquad \text{for } j=1,\dots,m,\ t=1,\dots,T_n \tag{C.25}$$

$$\sum_{t=2}^{T_n} b_{it} b_{i,t-1} = p_i - 1 \qquad \text{for } i = 1,\dots,n \tag{C.26}$$

$$\sum_t^{T_n} 2^{t-1}(b_{it} - b_{i't}) < 0 \qquad \text{if } \delta_{ii'} = 1 \text{ for } i=1,\dots,n,\ i'=1,\dots,n;\ i' \neq i \tag{C.27}$$

$$b_{it} + b_{i't} \le 1 \qquad \text{if } \delta_{ii'} = 1 \text{ for } i=1,\dots,n,\ i'=1,\dots,n;\ i \neq i',\ t=1,\dots,T_n \tag{C.28}$$

Input C.19 v_{ij} is set to 0 when person j cannot perform activity i. Since v_{ij} signifies the assignment of person j to activity i, this means that person j can not be assigned to activity i.

Input C.20 v_{ij} is set to 1 if person j is preferred to work on activity i by the planner.

Input C.21 b_{it} is set to 0 if timeslot t takes place before the release time of activity i. An activity cannot and should not be performed before it is available.

Input C.22 b_{it} is set to 1 for $t = s, s+1, \dots, s+p_i$ if the planner wants activity i to be scheduled at time s.

Constraint C.23 enforces that activity i is performed fully and only once. The sum of timeslots scheduled for activity i should be equal to the required number of timeslots to complete activity i.

Constraint C.24 enforces that an activity has the required number of persons assigned to carry it out. The sum of assignments - the number of assigned persons - should be equal to the required number of persons.

Constraint C.25 enforces that person j is scheduled at one activity at a time at most. By multiplying and summing the decision variables b and v over every activity, it ensures that at time t person j is assigned to one activity at most. For the following example $t = t_0$ holds.

$$b_{i,t_0} = 1$$
$$b_{i',t_0} = 1$$
$$v_{ij} = 1$$
$$v_{i'j} = 0$$
$$\sum_i b_{it_0} v_{ij} \leq 1 = 1 \cdot 1 + 1 \cdot 0 = 1 \leq 1$$

Thus this is a valid assignment.

If a person is assigned to two activities at time t_0 the following occurs:

$$b_{i,t_0} = 1$$
$$b_{i',t_0} = 1$$
$$v_{ij} = 1$$
$$v_{i'j} = 1$$
$$\sum_i b_{it_0} v_{ij} \leq 1 = 1 \cdot 1 + 1 \cdot 1 = 2 \not\leq 1$$

Since the constraint is now no longer true, this will not be a feasible solution.

Constraint C.26 enforces that once activities are started they cannot be paused and continued later[1]. This works because b_{it}, with i being fixed, can be seen as a sequence of bits that indicates whether activity i is scheduled to be performed at time t. As an example we show a valid and an invalid planning for activity i with $p_i = 4$ and $T_n = 6$.

$$b_{i*} = 0111100$$
$$\sum_{t=2}^{T_n} b_{it} b_{i,t-1} = 1 \cdot 0 + 1 \cdot 1 + 1 \cdot 1 + 1 \cdot 1 + 0 \cdot 1 + 0 \cdot 0 = 3 = p_i - 1 = 3$$

In a case with an additional interruption, with

$$b_{i*} = 0110110$$
$$\sum_{t=2}^{T_n} b_{it} b_{i,t-1} = 1 \cdot 0 + 1 \cdot 1 + 0 \cdot 1 + 1 \cdot 0 + 1 \cdot 1 + 0 \cdot 1 = 2 \neq p_i - 1 = 3$$

[1] Pausing and continuing later is also called pre-emption in context of planning.

For every interruption of the activity, the summation will decrease by 1. An additional interruption means that at least one 0 will be in the middle of the sequence of 1's, causing a previously sequential 11 to be separated. The multiplication for the 1 that is now the start of a new sequence of 1's is therefore no longer one, the sum will therefore reduce by one. This will cause the constraint to become false, which results in an infeasible solution.

Constraint C.27 enforces that activities are planned after the start time of their dependent activities. It does this by interpreting every scheduling bit string as a reverse binary number. We use the same representation of b_{it} as before and give an example to clarify how this works. In this case $T_n = 6$, $p_i = 3$, $p_{i'} = 1$ and $\delta_{ii'} = 1$.

$$b_{i*} = 011100$$
$$b_{i'*} = 000010$$
$$\sum_{t}^{T_n} 2^{t-1}(b_{it} - b_{i't}) = 2^0(0-0) + 2^1(1-0) + 2^2(1-0) + 2^3(1-0) + 2^4(0-1) + 2^5(0-0) = -2 < 0$$

This means that it is a valid schedule.

$$b_{i*} = 001110$$
$$b_{i'*} = 010000$$
$$\sum_{t}^{T_n} 2^{t-1}(b_{it} - b_{i't}) = 2^0(0-0) + 2^1(0-1) + 2^2(1-0) + 2^3(1-0) + 2^4(1-0) + 2^5(0-0) = 26 \not< 0$$

This is an invalid schedule which follows from the summation. However; constraint C.27 does have one problem:

$$b_{i*} = 001110$$
$$b_{i'*} = 000111$$
$$\sum_{t}^{T_n} 2^{t-1}(b_{it} - b_{i't}) = 2^0(0-0) + 2^1(0-0) + 2^2(1-0) + 2^3(1-1) + 2^4(1-1) + 2^5(0-1) = -28 < 0$$

Without an additional constraint this schedule would be considered valid. Activity i' should not be started before activity i has been completed. In order to avoid this problem constraint C.28 will ensure that they do not run simultaneously.

The sum starts at $t = 2$ as $t = 1$ is the first value in the sequence and cannot be calculated due to the requirement of the value at $t - 1$.

Constraint C.28 is introduced to prevent activities that have a dependency relationship from being planned at the same time. This solves the problem of constraint C.27.

If activity i and activity i' are active at the same time for a given t; $b_{i,t}$ and $b_{i',t}$ are 1. Therefore the sum of the two will result in violating this constraint: $b_{i,t} + b_{i',t} = 2 \not\leq 1$.

C.5. Objective Function

$$\gamma = \min \sum_{i} \sum_{t=d_i+1}^{T_n} \omega_i b_{it} \tag{C.29}$$

Function C.29 means that we want to minimise the amount of tardiness. Tardiness is defined as the amount of time spent on an activity after its deadline has passed. Which is in this case implemented as the sum of timeslots spent on an activity after its deadline, summed up for every activity. The penalty weight ω indicates how bad it is for an activity to be planned after its deadline.

D

Mixed Integer Programming Model Version 2

The first iteration of the model contains a multiplication between two binary variables (to behave like an **and**). This multiplication can be replaced by an additional variable and additional constraints. For example, given two binary variables a and b: $c = a * b$ behaves identical to $0 \leq a + b - 2c \leq 1$.[1]

All three variables are binary variables, thus verifying correctness by exhaustion is possible.

If $a = 0$ and $b = 0$ then subtracting 2 will always make the result negative, making the constraint false. As the constraint has to be true, 2c should equal 0 thus c will have to be 0.

If either $a = 0$ or $b = 0$ then subtracting 2 will always make the result negative, making the constraint false. As the constraint has to be true, 2c should equal 0 thus c will have to be 0.

If $a = 1$ and $b = 1$ the sum of these two variables will equal to 2, which makes the constraint false if $c = 0$. c will therefore have to be 1.

D.1. Indices and Sets

$i, i' \in A \ (i \neq i')$	Activity	(D.1)
$j \in W$	Person	(D.2)
$t \in \{1, \ldots, T_n\}$	Time interval	(D.3)
A	The set of activities	(D.4)
W	The set of persons	(D.5)
T	The set of time intervals	(D.6)
$n = \lvert A \rvert$		(D.7)
$m = \lvert W \rvert$		(D.8)
$T_n = \lvert T \rvert$		(D.9)

[1] From https://cs.stackexchange.com/questions/12102/express-boolean-logic-operations-in-zero-one-integer-linea using an approach similar to https://www.ncbi.nlm.nih.gov/pmc/articles/PMC1865583/

D.2. Parameters

$$r_i \in \mathbb{N}_0 \quad \text{Release time for activity } i \tag{D.10}$$

$$p_i \in \mathbb{N}_{>0} \quad \text{Processing time for activity } i \tag{D.11}$$

$$d_i \in \mathbb{N}_{>0} \quad \text{Deadline for activity } i \tag{D.12}$$

$$\omega_i \in \mathbb{N}_0 \quad \text{Penalty per time unit for missing deadline } d_i \tag{D.13}$$

$$r_{iW} \in \mathbb{N}_{>0} \quad \text{Required number of persons for activity } i \tag{D.14}$$

$$\delta_{ii'} = \begin{cases} 1 & \text{if } i \to i' \\ 0 & \text{Otherwise} \end{cases} \quad i' \text{ is dependent on } i \tag{D.15}$$

$$\theta_{ij} = \begin{cases} 1 & \text{if } j \text{ can perform } i \\ 0 & \text{Otherwise} \end{cases} \quad \text{Whether } j \text{ has the required skill to perform } i \tag{D.16}$$

D.3. Decision Variables

$$b_{it} = \begin{cases} 1 & \text{if activity } i \text{ is scheduled at time } t \\ 0 & \text{Otherwise} \end{cases} \tag{D.17}$$

$$v_{ij} = \begin{cases} 1 & \text{if person } j \text{ is assigned to activity } i \\ 0 & \text{Otherwise} \end{cases} \tag{D.18}$$

$$bv_{ijt} = \begin{cases} 1 & \text{if person } j \text{ is assigned to activity } i \text{ and } i \text{ is performed at time } t \\ 0 & \text{Otherwise} \end{cases} \tag{D.19}$$

$$\beta_{i,t} = \begin{cases} 1 & \text{if activity } i \text{ is scheduled at time } t \text{ and } t+1 \\ 0 & \text{Otherwise} \end{cases} \tag{D.20}$$

D.4. Constraints

$$v_{ij} = 0 \quad \text{if } \theta_{ij} = 0 \text{ for } \begin{matrix} i=1,\dots,n \\ j=1,\dots,m \end{matrix} \tag{D.21}$$

$$v_{ij} = 1 \quad \text{if preferred} \tag{D.22}$$

$$b_{it} = 0 \quad \text{if } t < r_i \text{ for } t = 1,\dots,T_n \tag{D.23}$$

$$b_{it} = 1 \quad \text{if preferred} \tag{D.24}$$

$$\sum_t b_{it} = p_i \quad \text{for } i = 1,\dots,n \tag{D.25}$$

$$\sum_j v_{ij} = r_{iW} \quad \text{for } i = 1,\dots,n \tag{D.26}$$

$$0 \le b_{it} + v_{ij} - 2bv_{ijt} \le 1 \quad \text{for } \begin{matrix} i=1,\dots,n \\ j=1,\dots,m \\ t=1,\dots,T_n \end{matrix} \tag{D.27}$$

$$\sum_i bv_{ijt} \le 1 \quad \text{for } \begin{matrix} j=1,\dots,m \\ t=1,\dots,T_n \end{matrix} \tag{D.28}$$

$$0 \le b_{it} + b_{i,t+1} - 2\beta_{i,t} \le 1 \quad \text{for } \begin{matrix} i=1,\dots,n \\ t=1\dots,T_n-1 \end{matrix} \tag{D.29}$$

$$\sum_{t=1}^{T_n-1} \beta_{it} = p_i - 1 \quad \text{for } i = 1,\dots,n \tag{D.30}$$

$$\sum_t^{T_n} 2^{t-1}(b_{it} - b_{i't}) \le 0 \quad \text{if } \delta_{ii'} = 1 \text{ for } \begin{matrix} i=1,\dots,n \\ i'=1,\dots,n;\ i' \ne i \end{matrix} \tag{D.31}$$

$$b_{it} + b_{i't} \le 1 \quad \text{if } \delta_{ii'} = 1 \text{ for } \begin{matrix} i=1,\dots,n \\ i'=1,\dots,n;\ i \ne i' \\ t=1,\dots,T_n \end{matrix} \tag{D.32}$$

Input D.21 v_{ij} is set to 0 when person j cannot perform activity i. Since v_{ij} signifies the assignment of person j to activity i, this means that person j can not be assigned to activity i.

Input D.22 v_{ij} is set to 1 if person j is preferred to work on activity i by the planner.

Input D.23 b_{it} is set to 0 if timeslot t takes place before the release time of activity i. An activity cannot and should not be performed before it is available.

Input D.24 b_{it} is set to 1 for $t = s, s+1, \dots, s+p_i$ if the planner wants activity i to be scheduled at time s.

Constraint D.25 enforces that activity i is performed fully and only once. The sum of timeslots scheduled for activity i should be equal to the required number of timeslots to complete activity i.

Constraint D.26 enforces that an activity has the required number of persons assigned to carry it out. The sum of assignments - the number of assigned persons - should be equal to the required number of persons.

Constraints D.27-D.28 enforces that person j is scheduled at one activity at a time at most. By AND-ing and summing the variables b and v over every activity, it is checked that at time t person j is assigned to one activity at most. For the following example $t = t_0$ holds and AND represents the value of bv_{ijt} as a result from constraint D.27.

$$b_{i,t_0} = 1$$
$$b_{i',t_0} = 1$$
$$v_{ij} = 1$$
$$v_{i'j} = 0$$
$$\sum_i bv_{ijt} = 1 \text{ AND } 1 + 1 \text{ AND } 0 = 1 \leq 1$$

Thus this is a valid assignment.

If a person is assigned to two activities at time t_0 the following occurs:

$$b_{i,t_0} = 1$$
$$b_{i',t_0} = 1$$
$$v_{ij} = 1$$
$$v_{i'j} = 1$$
$$\sum_i bv_{ijt} = 1 \text{ AND } 1 + 1 \text{ AND } 1 = 2 \nleq 1$$

Since the constraint is now no longer true, this will not be a feasible solution.

Constraints D.29-D.30 enforces that once activities are started they cannot be paused and continued later[2]. This works because β_{it}, with i being fixed, can be seen as a sequence of bits that indicates whether activity i is scheduled to be performed at time t and at time $t+1$. As an example we show a valid and an invalid planning for activity i with $p_i = 4$ and $T_n = 6$. For this example AND represents the value of $\beta_{i,t}$ as a result from constraint D.29.

$$b_{i*} = 0111100$$
$$\sum_{t=1}^{T_n-1} \beta_{it} = 1 \text{ AND } 0 + 1 \text{ AND } 1 + 1 \text{ AND } 1 + 1 \text{ AND } 1 + 0 \text{ AND } 1 + 0 \text{ AND } 0 = 3 = p_i - 1$$

In a case with an additional interruption, with

$$b_{i*} = 0110110$$
$$\sum_{t=1}^{T_n-1} \beta_{it} = 1 \text{ AND } 0 + 1 \text{ AND } 1 + 0 \text{ AND } 1 + 1 \text{ AND } 0 + 1 \text{ AND } 1 + 0 \text{ AND } 0 = 2 \neq p_i - 1$$

[2]Pausing and continuing later is also called pre-emption in context of planning.

The constraint counts the number of successive ones in the sequence of bits. An interruption means that at least one 0 will be in the middle of the sequence of 1's, causing a previously sequential 11 to be separated. As a pair of sequential 11 gets separated an AND loses its value of 1, making the summation getting below the value of $p_i - 1$. This will cause the constraint to become false, which results in an infeasible solution.

Constraint D.31 enforces that the ending time activities is planned after the ending time of their dependent activities. It does this by interpreting every scheduling bit string as a binary number. We give an example to clarify how this works. In this case $T_n = 6$, $p_i = 3$, $p_{i'} = 1$ and $\delta_{ii'} = 1$.

$$b_{i*} = 011100$$
$$b_{i'*} = 000010$$
$$\sum_{t}^{T_n} 2^{t-1}(b_{it} - b_{i't}) = 2^0(0-0) + 2^1(1-0) + 2^2(1-0) + 2^3(1-0) + 2^4(0-1) + 2^5(0-0) = -2 < 0$$

This means that it is a valid schedule.

$$b_{i*} = 001110$$
$$b_{i'*} = 010000$$
$$\sum_{t}^{T_n} 2^{t-1}(b_{it} - b_{i't}) = 2^0(0-0) + 2^1(0-1) + 2^2(1-0) + 2^3(1-0) + 2^4(1-0) + 2^5(0-0) = 26 \not< 0$$

This is an invalid schedule which follows from the summation. However; constraint D.31 does have one problem:

$$b_{i*} = 001110$$
$$b_{i'*} = 000111$$
$$\sum_{t}^{T_n} 2^{t-1}(b_{it} - b_{i't}) = 2^0(0-0) + 2^1(0-0) + 2^2(1-0) + 2^3(1-1) + 2^4(1-1) + 2^5(0-1) = -28 < 0$$

Without an additional constraint this schedule would be considered valid. Activity i' should not be started before activity i has been completed. In order to avoid this problem constraint D.32 will ensure that they do not run simultaneously.

The sum starts at $t = 2$ as $t = 1$ is the first value in the sequence and cannot be calculated due to the requirement of the value at $t - 1$.

Constraint D.32 is introduced to prevent activities that have a dependency relationship from being planned at the same time. This solves the problem constraint D.31 has.

If activity i and activity i' are active at the same time for a given t; $b_{i,t}$ and $b_{i',t}$ are 1. Therefore the sum of the two will result in violating this constraint: $b_{i,t} + b_{i',t} = 2 \not\leq 1$.

D.5. Objective Function

$$\gamma = \min \sum_{i} \sum_{t=d_i+1}^{T_n} \omega_i b_{it} \tag{D.33}$$

Function D.33 means that we want to minimise the amount of tardiness. Tardiness is defined as the amount of time spent on an activity after its deadline has passed. Which is in this case implemented as the sum of timeslots spent on an activity after its deadline, summed up for every activity. The penalty weight ω indicates how bad it is for an activity to be planned after its deadline.

E

Mixed Integer Programming Model Version 3

The first iteration of the model contains a multiplication between two binary variables (to behave like an **and**). This multiplication can be replaced by an additional variable and additional constraints. For example, given two binary variables a and b: $c = a * b$ behaves identical to $0 \leq a + b - 2c \leq 1$ [1]. This method is still applied within this third model, just like the second model.

All three variables are binary variables, thus verifying correctness by exhaustion is possible.

If $a = 0$ and $b = 0$ then subtracting 2 will always make the result negative, making the constraint false. As the constraint has to be true, 2c should equal 0 thus c will have to be 0.

If either $a = 0$ or $b = 0$ then subtracting 2 will always make the result negative, making the constraint false. As the constraint has to be true, 2c should equal 0 thus c will have to be 0.

If $a = 1$ and $b = 1$ the sum of these two variables will equal to 2, which makes the constraint false if $c = 0$. c will therefore have to be 1.

E.1. Indices and Sets

$i, i' \in A \ (i \neq i')$	Activity	(E.1)
$j \in W$	Person	(E.2)
$0 \leq t \leq T_n$	Time interval	(E.3)
A	The set of activities	(E.4)
W	The set of persons	(E.5)
$n = \lvert A \rvert$		(E.6)
$m = \lvert W \rvert$		(E.7)

[1] From https://cs.stackexchange.com/questions/12102/express-boolean-logic-operations-in-zero-one-integer-linea using an approach similar to https://www.ncbi.nlm.nih.gov/pmc/articles/PMC1865583/

E.2. Parameters

$$r_i \in \mathbb{N}_0 \quad \text{Release time for activity } i \tag{E.8}$$
$$p_i \in \mathbb{R}_{>0} \quad \text{Processing time for activity } i \tag{E.9}$$
$$d_i \in \mathbb{R}_{>0} \quad \text{Deadline for activity } i \tag{E.10}$$
$$\omega_i \in \mathbb{N}_0 \quad \text{Penalty per time unit for missing deadline } d_i \tag{E.11}$$
$$r_{iW} \in \mathbb{N}_{>0} \quad \text{Required number of persons for activity } i \tag{E.12}$$
$$\delta_{ii'} = \begin{cases} 1 & \text{if } i \to i' \\ 0 & \text{Otherwise} \end{cases} \quad i' \text{ is dependent on / should be scheduled after } i \tag{E.13}$$
$$\theta_{ij} = \begin{cases} 1 & \text{if } j \text{ can perform } i \\ 0 & \text{Otherwise} \end{cases} \quad \text{Whether } j \text{ has the required skill to perform } i \tag{E.14}$$

E.3. Decision Variables

$$v_{ij} = \begin{cases} 1 & \text{if person } j \text{ is assigned to activity } i \\ 0 & \text{Otherwise} \end{cases} \tag{E.15}$$
$$s_i\ (\mathbb{R}) \quad \text{Starting time of activity } i \tag{E.16}$$
$$a_i\ (\mathbb{R}) \quad \text{For objective function, see section E.5} \tag{E.17}$$
$$c_{ii'j} = \begin{cases} 1 & \text{if person } j \text{ is assigned to activity } i \text{ and } i' \\ 0 & \text{Otherwise} \end{cases} \tag{E.18}$$
$$c_{ii'} = \begin{cases} 1 & \text{if any person } j \text{ is assigned to both activity } i \text{ and } i' \\ 0 & \text{Otherwise} \end{cases} \tag{E.19}$$
$$o_{ii'} = \begin{cases} 1 & \text{if activity } i \text{ goes before activity } i' \\ 0 & \text{Otherwise} \end{cases} \tag{E.20}$$

Unlike the previous models, there are two continuous variables, s_i and a_i.

E.4. Constraints

In the constraints below L is number that is large enough to make the constraint true in every case.

$$s_i + p_i \leq T_n \quad \text{for } i = 1, \ldots, n \tag{E.21}$$

$$v_{ij} = 0 \quad \text{if } \theta_{ij} = 0 \text{ for } \begin{matrix} i=1,\ldots,n \\ j=1,\ldots,m \end{matrix} \tag{E.22}$$

$$v_{ij} = 1 \quad \text{if preferred} \tag{E.23}$$

$$s_i \geq r_i \quad \text{for } i = 1, \ldots, n \tag{E.24}$$

$$s_i = t \quad \text{if preferred} \tag{E.25}$$

$$\sum_j v_{ij} = r_{iW} \quad \text{for } i = 1, \ldots, n \tag{E.26}$$

$$0 \leq v_{ij} + v_{i'j} - 2c_{ii'j} \leq 1 \quad \text{for } \begin{matrix} i=1,\ldots,n \\ i'=i+1,\ldots,n \\ j=1,\ldots,m \end{matrix} \tag{E.27}$$

$$-m + 1 \leq \sum_j (c_{ii'j}) - m\, c_{ii'} \leq 0 \quad \text{for } \begin{matrix} i=1,\ldots,n \\ i'=i+1,\ldots,n \end{matrix} \tag{E.28}$$

$$L(1 - c_{ii'}) + L\, o_{ii'} + s_i \geq s_{i'} + p_{i'} \quad \text{for } \begin{matrix} i=1,\ldots,n \\ i'=i+1,\ldots,n \end{matrix} \tag{E.29}$$

$$L(1 - c_{ii'}) + L\,(1 - o_{ii'}) + s_{i'} \geq s_i + p_i \quad \text{for } \begin{matrix} i=1,\ldots,n \\ i'=i+1,\ldots,n \end{matrix} \tag{E.30}$$

$$s_{i'} \geq s_i + p_i \quad \text{if } \delta_{ii'} = 1 \text{ for } \begin{matrix} i=1,\ldots,n \\ i'=1,\ldots,n \end{matrix} \tag{E.31}$$

Input E.21 enforces that an activity cannot go past the end of the project.

Input E.22 v_{ij} is set to 0 when person j cannot perform activity i. Since v_{ij} signifies the assignment of person j to activity i, this means that person j can not be assigned to activity i.

Input E.23 v_{ij} is set to 1 if person j is preferred to work on activity i by the planner.

Input E.24 enforces that an activity cannot start before its release time.

Input E.25 s_i is set to t if the planner wants activity i to start at time t.

Constraint E.26 enforces that an activity has the required number of persons assigned to carry it out. The sum of assignments - the number of assigned persons - should be equal to the required number of persons.

Constraint E.27 guarantees the meaning of $c_{ii'j}$ by performing an AND between v_{ij} and $v_{i'j}$. If both v_{ij} and $v_{i'j}$ are 1 then $c_{ii'j}$ has to be 1. If one of them equals 0, $c_{ii'j}$ has to be 0, as explained on the front page.

Constraint E.28 guarantees the meaning of $c_{ii'}$. If at least a single person performs both activities i and i', $c_{ii'}$ will equal 1 and otherwise 0.

If at least a single person performs both activities, making the AND for $c_{ii'j}$ true, this constraint can only be satisfied if $c_{ii'}$ is 1. If $c_{ii'}$ would be 0, while $c_{ii'j}$ is 1, the sum would surpass the 0, making this constraint false. When no person performs both activities $c_{ii'j}$ will be 0, which means that if $c_{ii'}$ would be 1 the sum would become smaller than $-m+1$. As this makes the constraint false, $c_{ii'}$ has to be 0. For the following example $m = 5$ and activities i and i' do not overlap:

$c_{ii'j} = 0 \; \forall j$ (no overlap, therefore no overlap for any person) thus $\sum_j (c_{ii'j}) = 0$.
$-4 \leq 0 \quad -5c_{ii'} \leq 0$ has to be satisfied, thus $c_{ii'} = 0$, as $-4 \nleq -5$ is not true.
$|\{z\}$ $|\{z\}$
$-m+1$ $\sum_j (c_{ii'j})$

Constraint E.29 and E.30 enforce that activities i and i' do not overlap when a person is assigned to both. L is a large constant number chosen to make a constraint to be always satisfied.[2]. Depending on $o_{ii'}$, an extra term with L is added to either constraint E.29 or constraint E.30. This will cause one of them to be true in every case.

Overlapping is allowed when there is no overlap in the persons assigned to the activities, $c_{ii'} = 0$. To satisfy both E.29 and E.30 we therefore add L with the 'inverse' of $c_{ii'}$ to constraint E.30.

Constraint E.31 enforces that the ending time $s_{i'} + p_{i'}$ of the the dependencies i' of an activity i come before the starting time s_i of activity i.

E.5. Objective Function

$$a_i \leq s_i + p_i \tag{E.32}$$
$$a_i \leq d_i \tag{E.33}$$
$$\gamma = \min \sum_i \omega_i (s_i + p_i - a_i) \tag{E.34}$$

Function E.34 is meant to minimise the amount of tardiness. Tardiness is defined as the amount of time spent on an activity after its deadline has passed. The weight ω indicates how bad it is for an activity to be planned after its deadline, so that more important activities are less prone to going past the deadline than less important activities. Note that ω_i is a constant predetermined factor, therefore this objective is linear.

Since the objective to be implemented is minimising the tardiness, a max should be applied on every activity to avoid compensating for being early. In order to do this a_i is introduced as a helper variable. Together with constraint E.32 and E.33, this acts like a max function on an individual activity basis. Due to the objective function, there is pressure on a_i to be as high as possible. If $s_i + p_i \leq d_i$ - the activity is on time - this amounts to $a_i = s_i + p_i$ since E.32 is the strictest of E.32 and E.33, causing the component of the objective for the "costs" of this activity to be zero. Once $s_i + p_i \geq d_i$, a_i will at most have the value of d_i since now E.33 is the stricter one, causing the costs to be $s_i + p_i - d_i$. This means that the cost function of an activity is equal to the tardiness - the amount of time spent on a activity after the deadline.

Finally the objective can be described as the minimising the sum of tardiness over all activities.

[2] Since every starting time is before T_n, $L = T_n$ will work.

E.6. Remarks

Unlike the first model - which contained quadratic constraints - and the second model - which was sensitive to the scale of time in terms of constraints and variables. This model is neither, which makes it less memory intensive and faster.

F

Abstraction Layer Comparison

Modeling Language Software

Overview:

- AMPL
- CMPL
- Google Optimization Tools (OR-Tools)
- JuMP (JuliaOpt)
- PuLP

Name	*AMPL*	*CMPL*	*OR-Tools*	*JuMP*	*PuLP*	*Pyomo*
License	Proprietary	GPLv3	Apache 2.0	Mozilla 2.0	Free	BSD
Platform Windows	Yes	Yes	Yes	Yes	Yes	Yes
Platform Linux	Yes	Yes	Yes	Yes	Yes	Yes
Platform Mac OS	Yes	Yes	Yes	Yes	Yes	Yes
Number of supported solvers	10	5	7	16	6	6
Supports Gurobi	Yes	Yes	Yes	Yes	Yes	Yes
Supports Python	Yes	Yes	Yes	Partially (Not native)	Yes	Yes
Supports Quadratic	Yes	No	?	Yes	No	Yes

Requirements from our side:

- Support for Gurobi has to be included
- It should run on all platforms
- It should not increase the costs of the software system.

Conclusion:

Taking the requirements mentioned above into account, AMPL is excluded as it would increase the cost of the software system.

For this project we wish to stay as close to the Python programming language as we use it to build the rest of the software system. JuMP (JuliaOpt) requires a library for the Julia programming language, in which JuMP has been written. This would add more complexity to the code and is the reason we excluded JuMP from the choices.

The second choice is Google Optimization Tools (OR-Tools) as it supports python, the language our solver connector will be written in, and supports 7 solvers. After trying to install it we ran into many problems, some which should have been fixed years ago according to an online conversation between a user and a developer. As we want our system to not depend on software that we think is not reliable, we excluded OR-Tools from our choices.

The next choice is PuLP (Pyomo has been found later and was not taken into account when deciding on which abstraction layer to use). The only disadvantage about PuLP is that it does not support quadratic constraints, as we have developed Mixed Integer Programming (MIP) models that consist of only linear constraints, this is not seen as an obstacle. As the installation of PuLP was easy, just as its API, we chose for PuLP for our solver abstraction layer.

AMPL:

Type: Proprietary
Platforms: Windows, Linux, MacOS
Supported Solvers:

- Bonmin
- Couenne
- CONOPT
- CPLEX
- Gurobi
- Ipopt
- KNITRO
- MINOS
- SNOPT
- Xpress

Python: yes
Quadratic: yes
Extra: Able to connect your own solver using an open-source AMPL-solver library

CMPL:

Type: GPLv3 Licence
Platforms: Windows, OS X, Linux, Raspbian
Supported Solvers:

- CBC
- CPLEX
- GLPK
- Gurobi
- SCIP

Python: yes
Quadratic: no

Google Optimization Tools (OR-Tools):

Type: Apache 2.0 License
Platforms: Windows, Linux, MacOS
Supported Solvers:

- CBC
- CLP
- CPLEX
- GLOP
- GLPK
- Gurobi
- SCIP

Python: yes

Quadratic: Could not find

JuMP (JuliaOpt):

Type: Mozilla Public 2.0 License
Platforms: Windows, OS X, Linux
Supported Solvers:

- BARON
- Bonmin
- CBC
- CLP
- Couenne
- CPLEX
- ECOS
- GLPK
- Gurobi
- Ipopt
- KNITO
- MOSEK
- NLopt
- Parajito
- SCS
- Xpress

Python: partially, through a library for the Julia language.
Quadratic: Yes

PuLP:

Type: Free (Copyright notice that grants all rights to user)
Platforms: Windows, Linux, Mac OS
Supported Solvers:

- CBC
- COIN
- CPLEX
- GLPK
- Gurobi
- Xpress

Python: yes
Quadratic: No

Pyomo:

(Pyomo was found afterwards, and was therefore not included with picking an abstraction layer)
Type: BSD License
Platforms: Windows, Linux, Mac OS
Supported Solvers:

- CBC
- CPLEX
- GLPK
- Gurobi
- Ipopt
- Pico

Python: yes
Quadratic: Yes

G

JSON Objects

```
{
    "id": Number,
    "odooId"?: Number,
    "name": String,
    "startTime": Unix Timestamp,
    "endTime": Unix Timestamp,
    "persons": [Person],
    "activities": [Activity]
}
```

Listing G.1: Project JSON Object Definition

```
{
    "id": Number,
    "projectId": Number,
    "name": String,
    "preferredAssignments": { Activity.id : [ ... : Person.id ] },
    "preferredStartingTimes": { ... Activity.id : Unix Timestamp },
    "assignments"?: { Activity.id : [ ... : Person.id ] },
    "startingTimes"?: { ... Activity.id : Unix Timestamp }
}
```

Listing G.2: Planning JSON Object Definition

```
{
    "id": Number,
    "name": String,
    "releaseTime": Unix Timestamp,
    "deadline": Unix Timestamp,
    "processingTime": Number - Seconds,
    "penaltyWeight"?: Number,
    "numberOfRequiredPersons": Number,
    "requiredSkill": String
    "dependencies": [ Activity.Id ]
}
```

Listing G.3: Activity JSON Object Definition

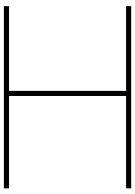
```
{
    "id": Number,
    "name": String,
    "skills": [ ... : String ]
}
```

Listing G.4: Person JSON Object Definition

JSON Objects are stored in the 'storage' folder in the project root following the following layout:

- configuration

- odoo
  - xmlrpc_parameters.json
- projects
  - project_<project_id>
    - project_<project_id>.json
    - plannings
      - planning_<project_id>_<planning_id>.json



Original Project Description

Original Project Description

Support for human planners: scheduling and visualisation

Project description

Given: a production system for human (project) planners. The system in question is Odoo, an open-source suite for Enterprise Resource Planning (ERP).

Assignment:
1. Visualize the manual planning made in Odoo
2. Facilitate manually adapting this planning
3. Propose an automatically generated improvement of the manual planning

Iteration between steps 2 and 3 might be desirable, because the manual planner might have some preferences that weren't specified to the solver.

This type of planning is in a class of hard problems (to be studied in the course of Complexity Theory), and is captured in a variety of models, such as the “job shop” problem. There exists a large body of algorithmic research for dealing with them. Step 3 would tap into this research to adapt and apply a suitable algorithm to this situation.

There is an optional (but not required!) tie-in with the TI3706 Bachelorseminarium assignment “Algorithms for scheduling”, which has the same supervisors.

Company description

West IT Solutions was founded in 1985 and has always been on the forefront of technological innovation. As an example, West was involved in the development of one of the first internet banking solutions.

Started as a venture in late 2016, one of the newer focus areas of West IT Solutions is "Smart Planning". The key here is to develop innovative modules which, based on available (big) data, (semi-)automatically optimise resource, asset, and maintenance planning.

A cooperation with the Algorithmics group at Delft University of Technology was started in 2017.

For a practical application of these solutions, West is investigating a partnership with companies such as HTM, Mammoet, Westland Infra and Nedtrain.

Database Model

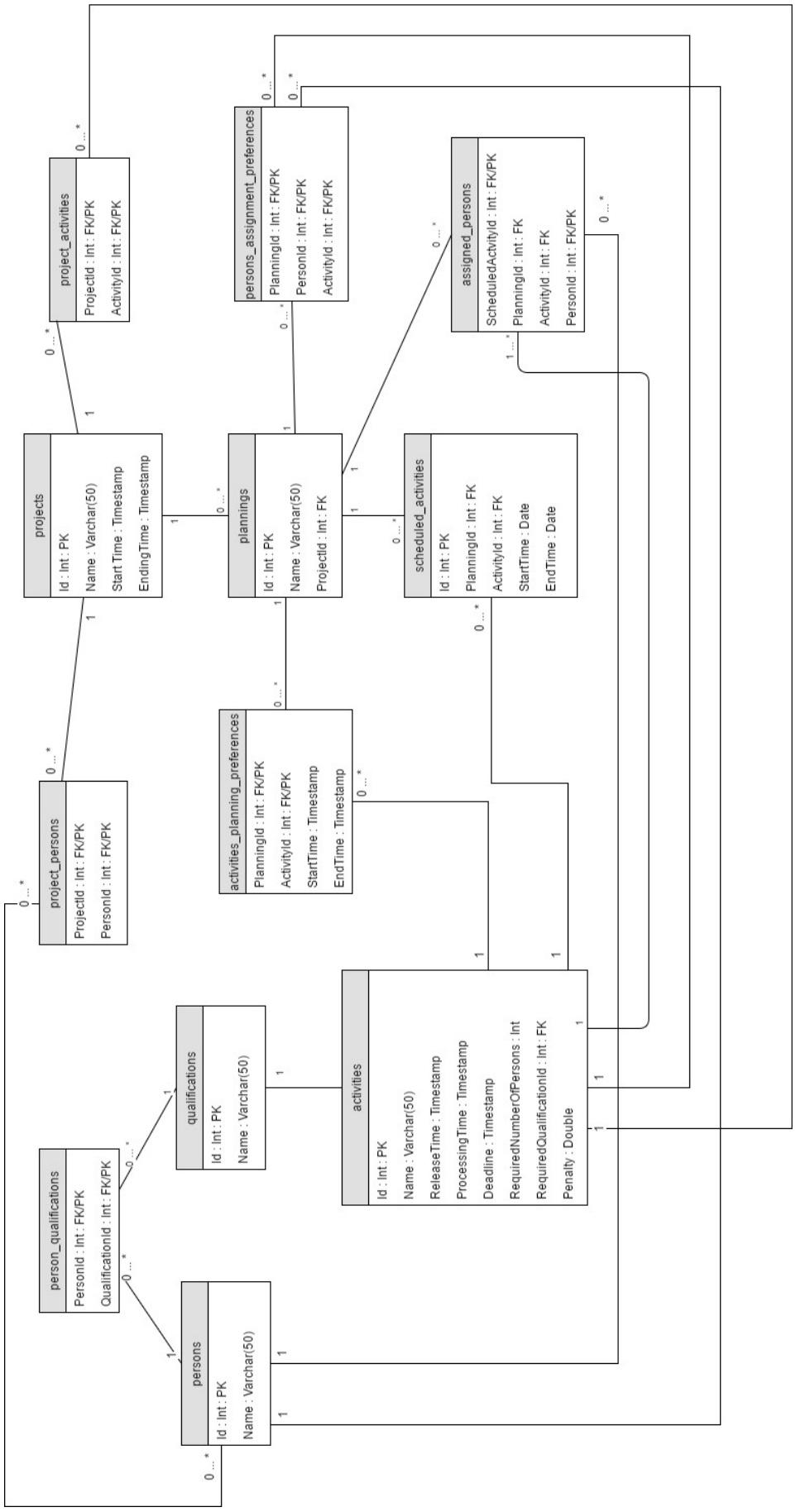


Figure I.1: Relational Database Model

J

Class Diagrams

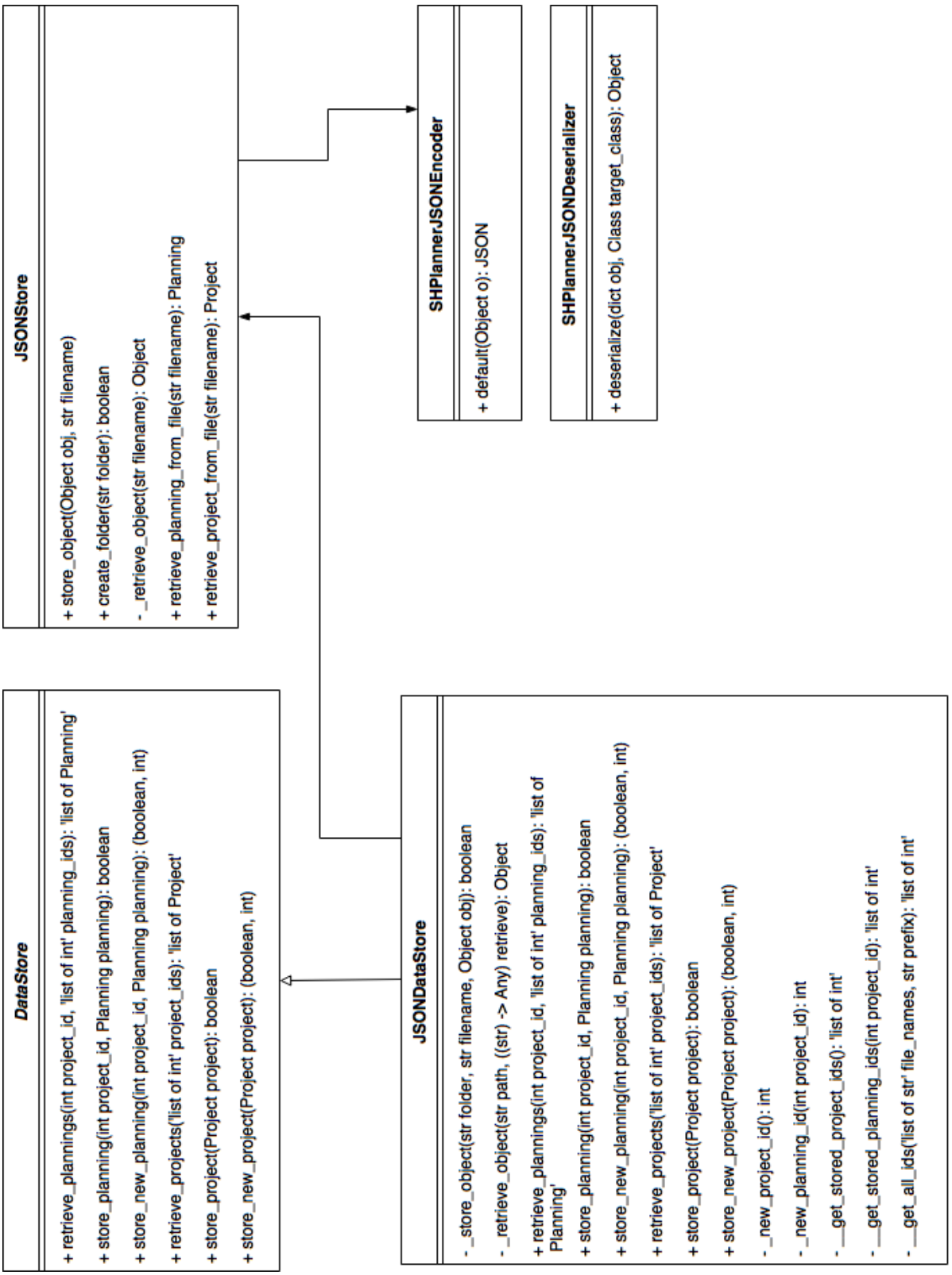


Figure J.1: Data Storage

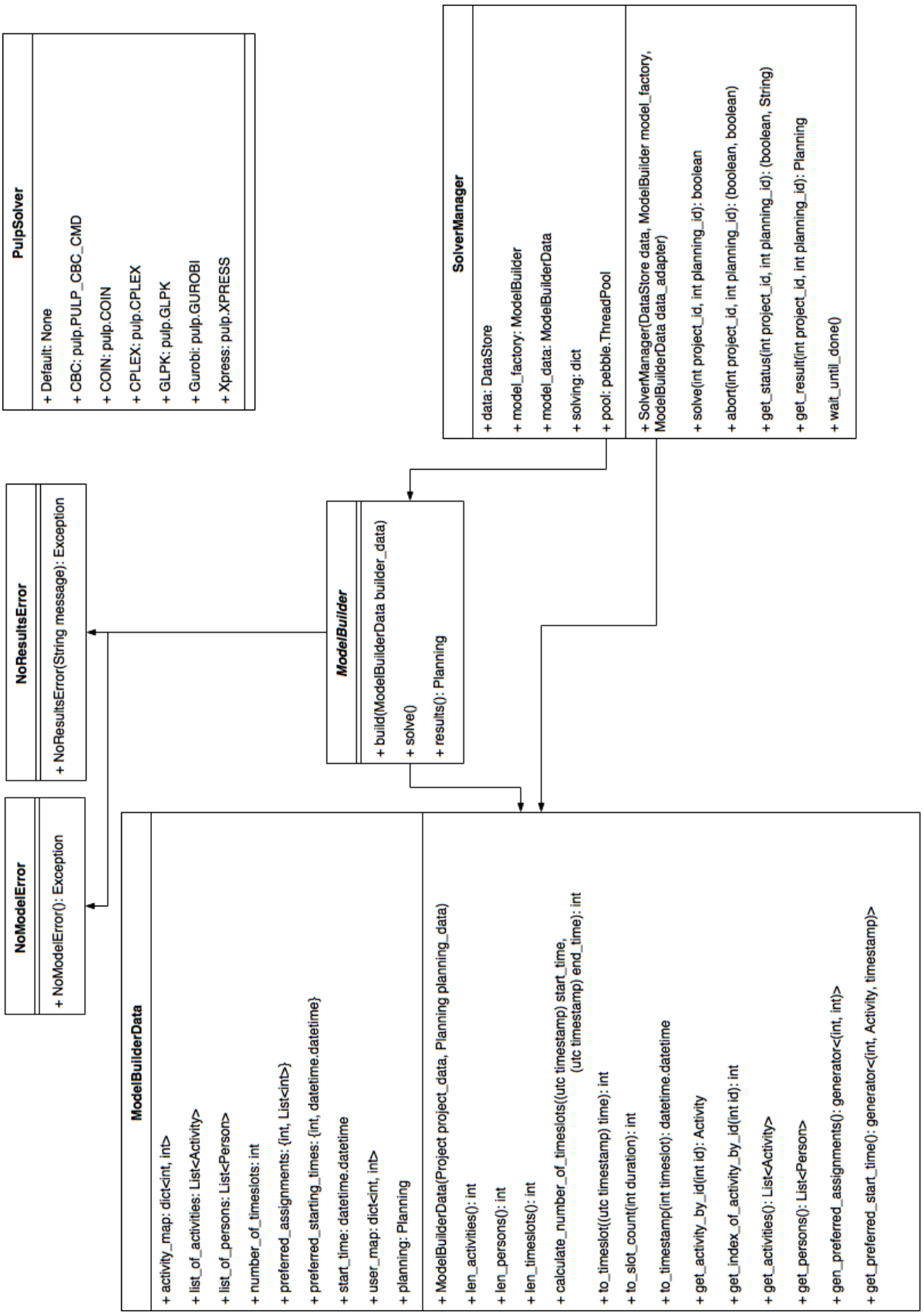


Figure J.2: Solver Connector

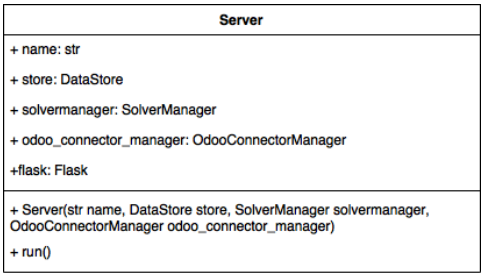


Figure J.3: API Server

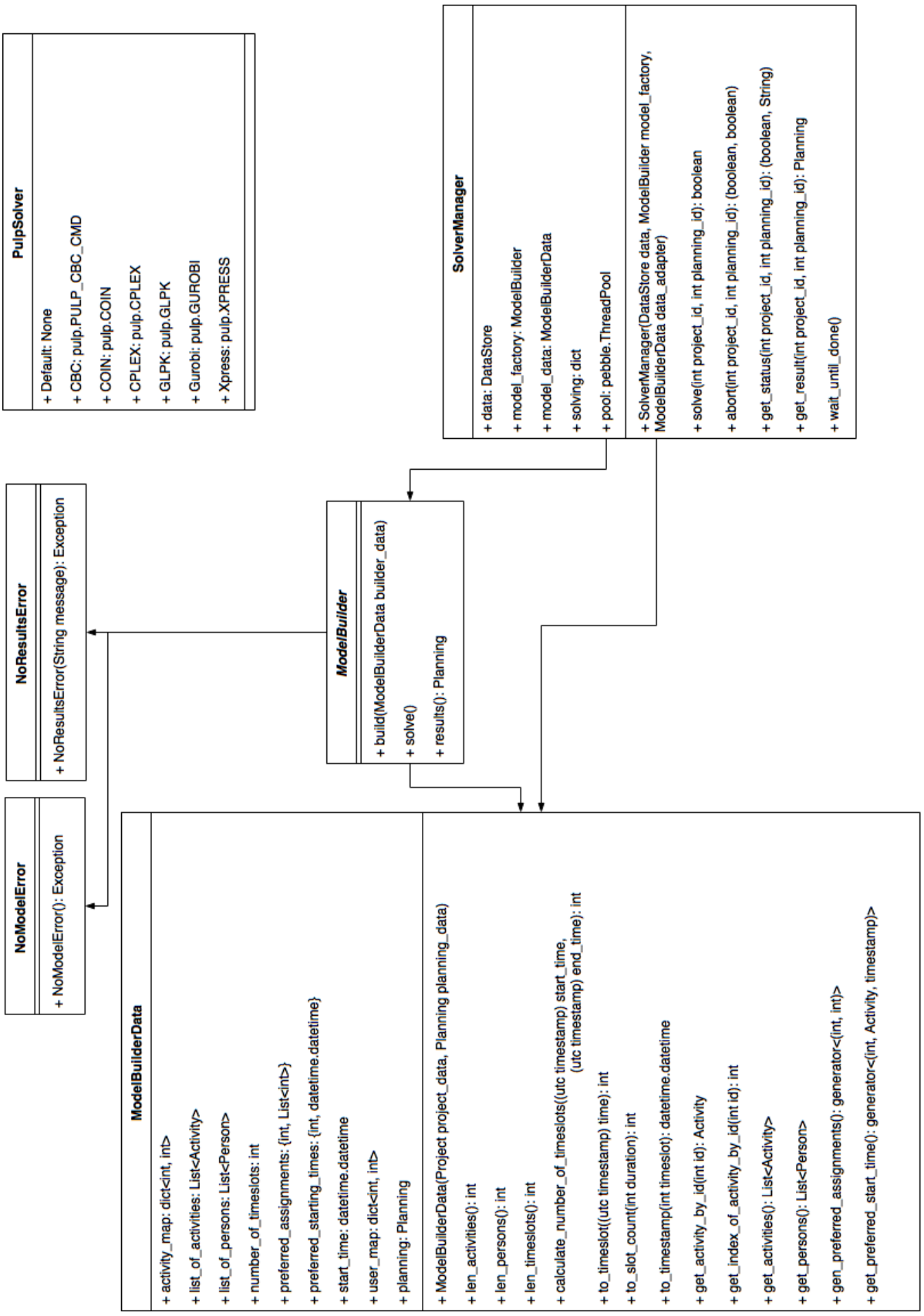


Figure J.4: Solver Connector

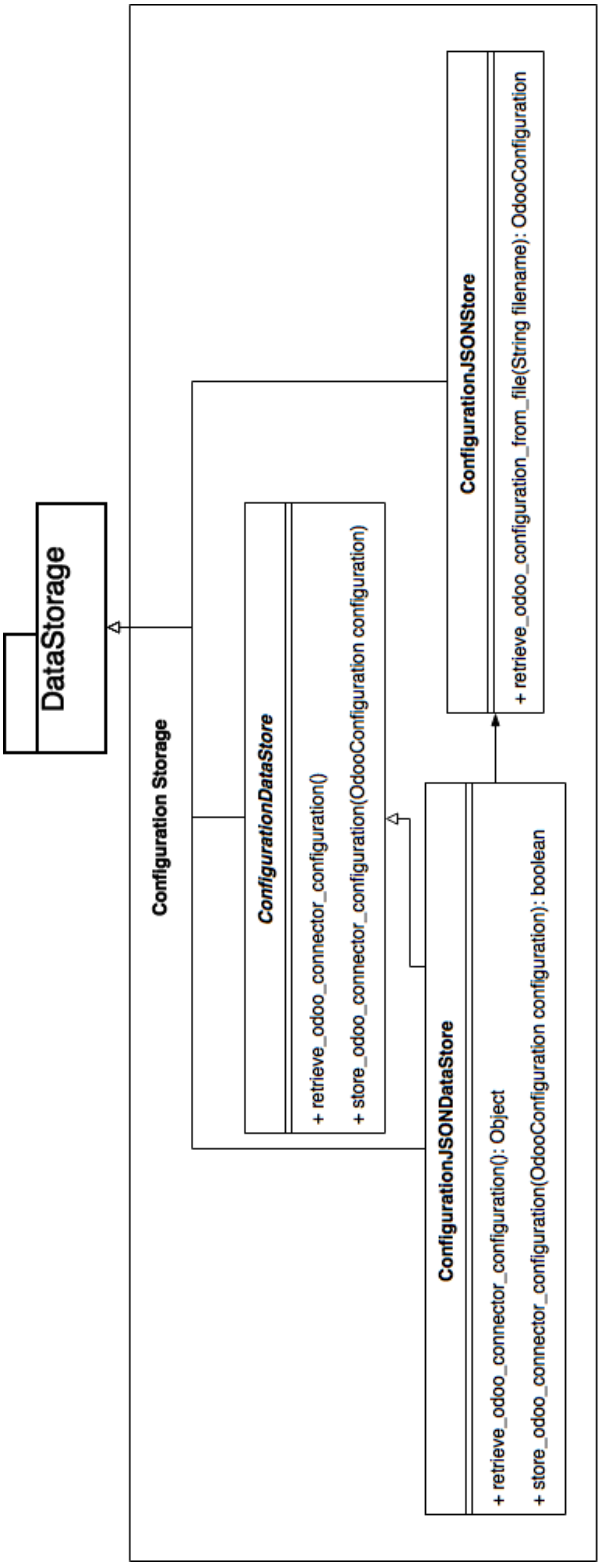


Figure J.5: Solver Connector Configuration Storage

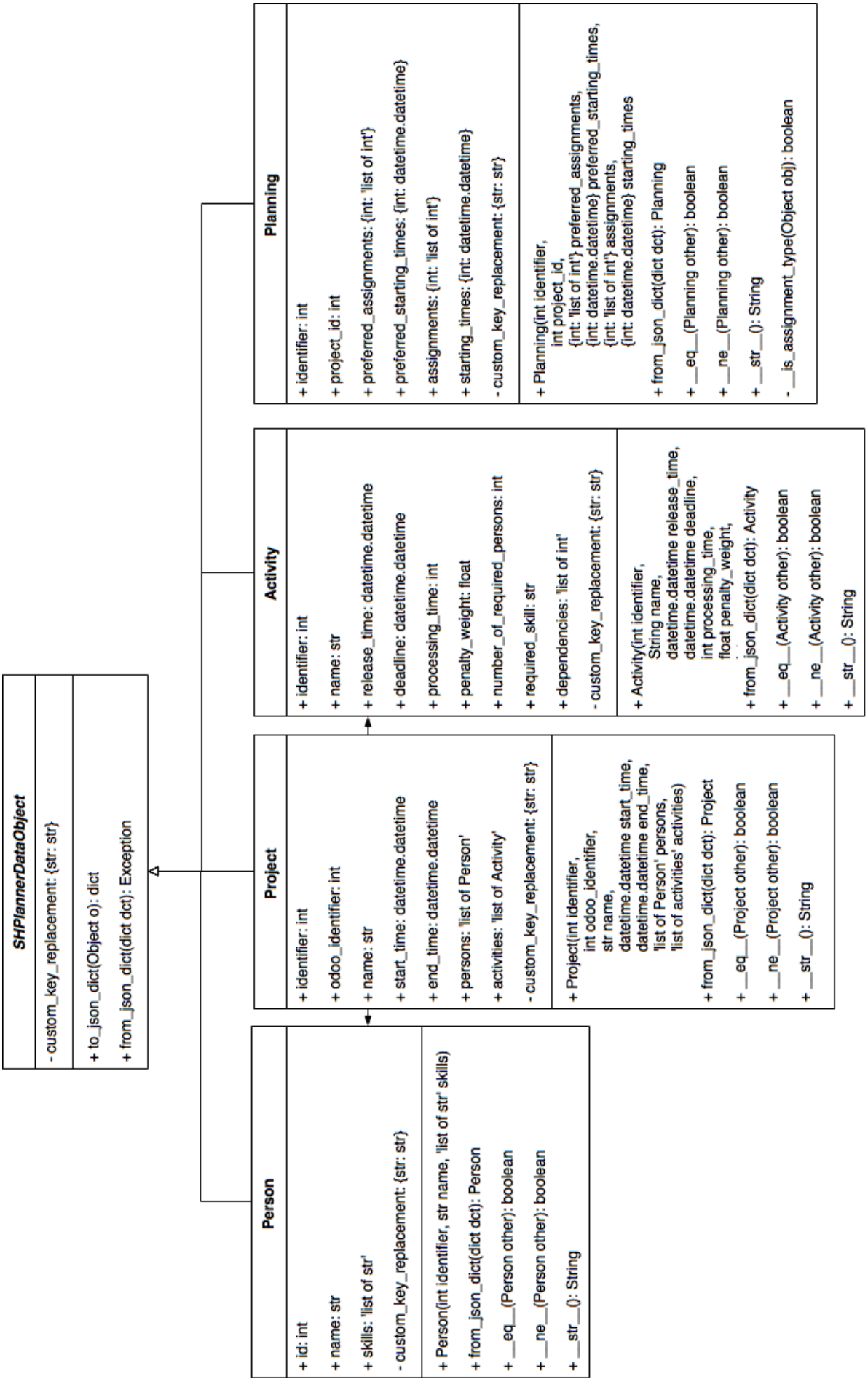


Figure J.6: Models

K

Experiment Results

Table K.1: Results for GurobiPy

Instances	Model 1 Build (s)	Model 1 Solve (s)	Model 1 Solved	Model 2 Build (s)	Model 2 Solve (s)	Model 2 Solved	Model 3 Build (s)	Model 3 Solve (s)	Model 3 Solved
activities_15	0.79	3.30	TRUE	3.19	4.88	TRUE	0.16	0.05	TRUE
activities_75	3.66	21.01	TRUE	15.86	89.07	TRUE	4.04	3.30	TRUE
activities_150	7.33	10.87	FALSE	31.71	316.67	TRUE	16.77	199.00	TRUE
activities_225	11.06	28.40	FALSE	49.25	65.95	FALSE	38.38	2066.62	FALSE
days_1	0.02	0.52	FALSE	0.08	0.02	FALSE	0.04	0.01	FALSE
days_3	0.08	0.23	TRUE	0.22	0.19	TRUE	0.04	0.02	TRUE
days_7	0.16	0.45	TRUE	0.52	0.71	TRUE	0.04	0.02	TRUE
days_10	0.23	1.80	TRUE	0.75	0.19	FALSE	0.04	0.02	TRUE
days_14	0.32	0.62	TRUE	1.08	0.14	FALSE	0.04	0.02	TRUE
days_21	0.47	1.09	TRUE	1.62	0.25	FALSE	0.04	0.02	TRUE
persons_3	0.16	0.38	TRUE	0.54	0.28	TRUE	0.04	0.02	TRUE
persons_15	0.51	0.74	TRUE	1.95	1.45	TRUE	0.10	0.04	TRUE
persons_30	0.93	1.16	TRUE	3.83	5.13	TRUE	0.19	0.06	TRUE
persons_45	1.35	2.31	TRUE	5.60	16.27	TRUE	0.28	0.06	TRUE
persons_55	1.66	2.49	TRUE	6.85	20.00	TRUE	0.33	0.07	TRUE
persons_75	2.24	2.54	TRUE	9.33	8.11	TRUE	0.44	0.09	TRUE

Table K.2: Results for PuLP

	Model 2			Model 3		
Instance	**Build (s)**	**Solve (s)**	**Solved?**	**Build (s)**	**Solve (s)**	**Solved?**
activities_15	7.30	8.73	TRUE	0.22	0.27	TRUE
activities_75	42.22	47.30	TRUE	5.69	10.39	TRUE
activities_150	100.96	320.94	TRUE	22.69	279.21	TRUE
activities_225	180.39	121.22	FALSE	52.82	2899.14	FALSE
activities_275	241.83	149.52	FALSE	78.52	3728.50	FALSE
days_1	0.16	0.16	FALSE	0.05	0.06	FALSE
days_3	0.54	0.44	TRUE	0.05	0.07	TRUE
days_7	1.52	1.50	TRUE	0.05	0.08	TRUE
days_10	2.43	1.47	FALSE	0.05	0.07	TRUE
days_14	4.10	2.01	FALSE	0.05	0.07	TRUE
days_21	6.84	3.04	FALSE	0.05	0.07	TRUE
persons_3	1.48	1.03	TRUE	0.05	0.07	TRUE
persons_15	4.66	4.42	TRUE	0.14	0.18	TRUE
persons_30	8.55	9.78	TRUE	0.26	0.32	TRUE
persons_45	12.70	18	TRUE	0.38	0.45	TRUE
persons_55	15.06	23.80	TRUE	0.45	0.54	TRUE
persons_75	20.47	22.75	TRUE	0.61	0.75	TRUE

Info Sheet

Support for Human Planners

Scheduling and Visualisation

Client
West IT Solutions

Presentation Date
July 3th, 2017 at 12:00h

Project Description
Creating a work planning for a project is a challenging task as multiple factors and optimisations have to be taken into account. For this reason West IT Solutions is looking into creating tools to support human planners in creating plannings. The tool should automatically propose a planning, visualise it and allow the user to modify this planning. The tool should use existing information from Odoo, an enterprise resource planning system, of which West is a vendor. The goal of this project is to create a tool, with the aforementioned functionalities, that can be used to demonstrate the possibilities to potential customers.

During the research phase we interviewed a client of West, analysed the case problem they provided, and proved that it is NP-Hard. We looked at similar products and interviewed our client. Using the things we learned, we created a problem model and defined a list of functional requirements.

During the project we used an Agile development process. The project consisted of seven sprints with each sprint adding new functionalities to the product. To improve performance of planning generation, additional models were created. Despite the fact that the problem was NP-Hard, we managed to automatically generate plannings using a Mixed Integer Programming formulation and a state-of-the-art solver.

The final product allows planners to automatically generate plannings, taking into account the planners' preferences, with information from Odoo, visualise this planning and modify it. We used unit tests, static analysis tools and continuous integration to increase confidence in the quality of the product. Acceptance tests were used to make sure the developed product was what our client wanted.

There are many ways to extend the product, which include: extending the models to make it more intuitive to iterate between manual modification and automatic generation of a planning, extending error detection in the visualised planning and fully integrating the product with Odoo.

Toward the end of the project, the developed product was demoed to West's client and they were quite impressed. West has indicated they are interested in further developing the product.

Client:	Léon Planken	West IT Solutions
Supervisor:	Mathijs de Weerdt	TU Delft Algorithmics Group

All team members worked on all parts of the project and product. The parts members contributed to most are highlighted below. The final report can be found at `repository.tudelft.nl`

Cas Buijs
Interests:

- Artificial intelligence;
- Cybersecurity;
- Virtual reality

Biggest Contribution:

- Odoo connection

Contact:
info[at]casbuijs[dot]nl

Arthur Guijt
Interests:

- Problem solving;
- Nature walks

Biggest Contribution:

- MIP Models

Contact:
arthur.guijt12[at]gmail[dot]com

Lars Stegman
Interests:

- Software architecture;
- Human interface design;
- Baking bread

Biggest Contribution:

- User interface

Contact:
larsstegman[at]outlook[dot]com

M

Research Report

Research Report

Support for Human Planners: Scheduling and Visualization

Cas Buijs
S.J.M.Buijs@student.tudelft.nl

Arthur Guijt
A.Guijt@student.tudelft.nl

Lars Stegman
L.S.Stegman@student.tudelft.nl

May 2017

Contents

1. Introduction

The goal of this project is to support planners by implementing an algorithm that supports them by giving suggestions for certain parts of their job. Our client is West IT Solutions, a company that concerns itself with technological innovation. One of their focus areas is "Smart Planning", in which the key is to develop innovative modules which (semi-)automatically optimise resource, asset, and maintenance planning in Odoo. Odoo is an Enterprise Resource Planning (ERP) suite in which you can add modules with extra functionalities like adding applications to your smartphone.

During this project, we will be dealing with a case presented to our client by TNO. TNO is a research institute that on behalf of governments and other parties puts scientific knowledge into practice to solve real-world problems.

In this research report we want to answer the following questions:

- What is the problem we have to solve exactly?
- Why is this a problem?
- Who will benefit from this solution?
- What is the complexity of our problem?
- What are possible approaches for solving our problem?
- How can we evaluate our solution?

Chapter 2 describes the case of TNO and the extracted problem definition from it. This chapter also contains the answers to the questions of why this is a problem, who will benefit from a solution for this problem and what the complexity of this problem is. In chapter 3 possible approaches for solving our problem are discussed together with problems concerning each approach. Chapter 4 discusses existing solutions to similar problems and provides a brief explanation of our evaluation of tools that can be used during the development phase. Chapter 5 discusses possible approaches concerning the design of the software and chapter 6 explains how the project and the solution can be evaluated at the end of the project.

2. Problem Definition

2.1. TNO Case Problem Description

During the research phase of our project, we had an interview with Patrick Hanckmann from TNO concerning the case problem. A summary of the interview can be found in appendix A.

In an organisation there are many different activities that have to be carried out to keep everything running. These activities take place at a certain time and are carried out by certain persons. In the case of TNO this organisation is a navy ship with on board a group of soldiers that have to keep the ship up and running. Besides performing work activities, soldiers also need some private time and need to eat.

In the case specified by TNO, there are two types of activities soldiers can be busy with:

- Personal activities
- Work activities

Each of these two types of activities cover a larger set of activities among which the following are some examples to give a better idea:

- Personal activities:
  - Sleeping
  - Free time
  - Eating
- Work activities:
  - Cleaning
  - Cooking
  - Training
  - Etc...

To assign activities to soldiers, some constraints need to be taken into account. Just randomly assigning qualified soldiers to activities will result in some practical problems on the ship.

First, soldiers can only perform one activity at a time, since it is not possible for them to be at two places at the same moment. Thus a soldier cannot be scheduled for more than one activity at a certain moment in time.

Second, the activities are not isolated entities, a dependency relationship can exist between them. For example, soldiers can only eat when something has been cooked, thus there exists a dependency relationship between the cooking activity and the eating activity.

Third, multiple soldiers can be qualified to perform an activity, though not all qualified soldiers should be assigned to an activity at the same time. An activity requires only a limited amount of soldiers to carry it out, which means that only a subset of all qualified soldiers should be assigned to the activity. To give some examples, navigation could require 4 soldiers and cooking could require 2 soldiers.

Fourth, the soldiers should perform an activity at all times, it does not matter whether this is a personal activity or a work activity.

Fifth, soldiers should have a diverse schedule. Carrying out the same activities all the time can reduce their dexterity for other activity. Because of this, the schedule should keep in mind that soldiers need to carry out every activity they are qualified for regularly.

Sixth, human planners can have certain reasons to plan an activity at a certain time or to assign certain soldiers to certain activities. The schedule should plan around the planner's wishes.

Seventh, soldiers cannot all sleep or eat at the same time, as the ship cannot sail without soldiers controlling it. To ensure that the ship can keep sailing, soldiers work in shifts. These shifts prescribe when and which soldiers sleep and eat, which prevents them from doing it all at the same time.

Finally, the free time of soldiers consists of 4 hours a day. These 4 hours can be split into blocks, but there should be one block of at least 2 hours and there should be at least one block before sleeping. The remaining hours of free time can be distributed over the day.

2.1.1. Extracted Problem Definition

The TNO case provides many requirements, but we are not able to implement all of them within the time frame of this project. We have chosen a subset of requirements from the TNO case to fulfil. We describe what choices we have made and how we incorporate this into a model.

The first requirement we drop is that soldiers, further called persons, need to be scheduled in shifts automatically. There exists literature that deals with scheduling persons into shifts, see the "Nurse Scheduling Problem" [4]. This part of the problem requires its own analysis, while our client is more interested in the task assignment part of the problem. We assume that the assignment of persons into shifts has already been made and that

this assignment is available to the scheduling algorithm. As we assume the assignment into shifts to be part of the input to the algorithm, it should be possible in the future to generate the assignment automatically and to use those results as input to the algorithm.

The second requirement we drop is that persons have to be busy with an activity at all times. It could happen that there are too many people on the ship to keep them all busy all the time. Since the client prefers some redundancy in people assignment, dropping this requirement should almost automatically fulfil this wish.

Keeping the TNO case in mind, we describe the activities and persons in the following way:

$A = (r, p, d, \omega, r_W, q_S, W', R_A)$ - Activity:

- r - Release time; The time from when the activity may be performed.
- p - Processing time; The amount of time it takes to perform the activity.
- d - Due date; The time by which the activity must be completed.
- ω - Delay penalty weight; The weight for a penalty in case this activity is scheduled to finish after the due date.
- r^W - Required amount of persons; The number of persons needed to perform this activity.
- $q_S \in Q$ - Required qualification; The qualification a person needs to perform this activity.
- W' - Mandatory assigned persons; Persons who must be assigned to this activity. $W' \subseteq W$
- R_A - Activity Dependencies; The activities this activity depends on. $R_A \subseteq A$

$W = (S)$ - Person:

- $S \subseteq Q$ - The qualifications of a person.

Q - The set of all possible qualifications.

Objective function:

$$\gamma = \min \sum_i \omega_i T_i \tag{2.1}$$

$$= \min \sum_{i=1}^{|A|} (\omega_i \cdot \max\{0, C_i - d_i\}) \tag{2.2}$$

With T_i being the tardiness and C_i the end time of the activity.

One thing has to be noted. If a person is required to be assigned to an activity, it does not mean that that person has to be qualified to carry out that activity. This is entirely at the planner's discretion.

The objective function γ minimises the total amount of tardiness [2]. This means that if an activity is finished after the deadline, the value of the objective function increases by the penalty weight multiplied by the amount of time it is finished after the deadline.

Certain activities are part of the shift system. This can be enforced in this model by creating special activities. For these special activities the following can hold:

- $\omega = \infty$; to prevent algorithms from deviating from the shift system.
- $r_W = |W'|$, so the algorithms will not try to add persons to a shift related activity they don't belong in.
- W' consist of all persons in a shift.

All other attributes can be used as normal. For example for the sleep activity the following holds: r is arbitrary, p indicates how long people sleep and $d = r + p$.

For the last requirement of the free hours, we choose to split the 4 hours into three blocks, one of 2 hours and 2 of one hour with each a dependency between them, preventing them from being planned right after each other.

2.2. Why Is This A Problem?

Currently planning for a navy ship happens manually by a group of people, they are busy for multiple days to create a planning on paper. When something unexpected happens that makes their planning incorrect they create a new schedule. This process does not only use of a lot of man power but it takes also a lot of time. The amount of manpower and the amount of time spent creating this planning lead to large expenses for the navy.

Another problem with the manually created planning is that there might be more persons on board of the ship than necessary, just to make creating a planning a feasible task. A current planning makes use of more than 170 crew members, while a more optimised planning could make use of less.

2.3. Who Benefits From the Solution?

The original problem of TNO can be split into two parts, part one concerns itself with assigning persons to shifts and the second part concerns itself with assigning persons in a shift to activities. By dropping the first requirement that people need to be assigned into shifts we dropped the first part of the original problem. This means that our solution only takes care of assigning persons to activities while assuming they are already assigned to shifts.

When the shifts are already known, which can be the case if the human planner assigned the persons into shifts himself, our solution can be used to assign the persons to activities

they are qualified for. Not only the navy but also companies that want to schedule employees and activities can use our solution as long as they provide the assignment to shifts themselves.

2.4. Problem Complexity

The NP-complete problem 2-PARTITION [5] can be reduced to the decision variant of our problem.

Given an instance g of 2-PARTITION you have a set $X = \{x_1, \ldots, x_n\}$ of items and a function

$$G : X \to \mathbb{Z}^+$$

which gives every $x \in X$ a positive value. The question is whether a division exists of all the items over 2 bins, so that $G(X') = G(X - X')$ with $X' \subset X$.

Given an instance of our problem, the question is whether there exists an assignment of activities to persons that will allow all activities to be completed before their due date, thus the tardiness being equal to 0. In formal notation $\min \gamma = 0$.

2.4.1. Polynomial Reduction

$$
\begin{aligned}
A &= \{(r_i, p_i, d_i, \omega_i, r_{iW}, q_{iS}, W_i', R_{iA}) \mid i = 1, \ldots, n\} \\
r_i &= 0 && \text{for } i = 1, \ldots, n \\
p_i &= G(x_i) && \text{for } i = 1, \ldots, n \\
d_i &= \frac{1}{2}v && \text{for } v = \Sigma_{x_i \in X} G(x_i);\ i = 1, \ldots, n \\
\omega_i &= 1 && \text{for } i = 1, \ldots, n \\
r_{iW} &= 1 && \text{for } i = 1, \ldots, n \\
q_{iS} &= * && \text{for } i = 1, \ldots, n \\
W_i' &= \emptyset && \text{for } i = 1, \ldots, n \\
R_{iA} &= \emptyset && \text{for } i = 1, \ldots, n \\
W &= \{(S_1), (S_2)\} \\
S_1 &= S_2 = \{*\}
\end{aligned}
$$

Let $n = |X|$. First calculating v costs $O(n)$ time. Creating every element in A costs $O(1)$ time which happens $O(n)$ times, thus instantiating all elements costs $O(n)$ in total. This means that this reduction costs $O(n)$ and is polynomial.

2.4.2. $g \in$ 2-PARTITION $\rightarrow f(g) \in$ Our Problem

Assume that $g = (A, G)$ is a yes-instance of 2-PARTITION.

We transform every item in A to an activity and set the deadline equal to half the sum of the values. We also create an arbitrary required qualification, since it not important in this context. Every task only needs to be performed by one person. There are also only two people to carry activities who each have the generic qualification.

The activities must be distributed equally over the two persons, otherwise, the deadline cannot be met. Because g is a yes-instance of 2-PARTITION, the assignment that distributes the tasks equally must exist. This means that a schedule can be made that satisfies $\min \gamma = 0$ where every person has to perform exactly half of the total processing time.

Thus $f(g)$ is a yes-instance of Our Problem.

2.4.3. $f(g) \in$ Our Problem $\rightarrow$ g $\in$ 2-PARTITION

Assume that $f(g) = (A', G')$ is a yes-instance of Our Problem with $A = \{a_1, \ldots, a_n\}$. We show that $g = (A, G)$ is an instance of 2-PARTITION.

The deadline for every activity is equal to half of the summed processing times. This means that a person cannot perform all activities, but that both persons each have to perform exactly half of the total processing time.

By using the same assignment of activities to people for items to people, a yes-instance of 2-PARTITION is created. This is because the processing time per person is equal to exactly half the total processing time. Since processing time is equal to the value of an item, the received value per person is exactly half of the total value.

Using this we have proven 2-PARTITION can be reduced to our problem. Because of this and the fact that 2-PARTITION is NP-complete our problem must be NP-hard as well.

3. Possible Approaches

In this chapter, we discuss possible approaches for our problem and possible mappings from our problem to existing problems for which algorithms exist to solve them. Our problem might be only partially mappable to an existing problem, in this case we will discuss what requirements the mapping is not taking care of.

3.1. Job Shop with Multi-Purpose Machines

The problem has quite a lot in common with the NP-hard[1] "Job Shop with Multi-Purpose Machines" problem as described by Hurink et al. [6]. Except the machines are persons instead. After this simple change there are a few requirements that still have to be met:

- A mapping should take into account the human planner's preference for assigning a certain person to a certain activity
- A mapping should take into account fixed time frames for activities
- A mapping should take into account that more than one person could be required for carrying out the activity
- A person should not be assigned to the same tasks all the time, the schedule should be diverse

In the following sections, we will explain these problems more in depth. The first two issues can be resolved using a mapping without many problems, the third issue however, has a mapping that can in some cases increase the problem size dramatically.

3.1.1. Formalized Mapping

A Job Shop with MPM problem consist of the following terminology [2]:

- J_i for $i = 1, \ldots, r$ jobs
- O_{ij} operations $j = 1, \ldots, n$ for jobs $i = 1, \ldots, r$
- p_{ij} processing time of O_{ij}
- r_i release time of job i

[1] The Job Shop Problem with normal machines is a restriction of this problem.

- $\mu_{ij} \subseteq \{M_1, \ldots, M_m\}$ the machines capable of performing O_{ij}
- f_i(t) function which gives the cost of completing job i at time t
- d_i the deadline of job i
- $G = (V, A)$ a precedence graph that indicates that job dependencies are.

Our problem maps in the following way to this terminology:

- J_i for $j = 1, \ldots, r$ activities
- O_{i1} activities consist only of one operation in our case.
- p_i processing time of activity i
- r_i release time of activity i
- $\mu_i \subseteq \{M_1, \ldots, M_m\}$ a subset of the m total persons. This persons in this subset can perform job i.
- f_i(t) function which gives the cost of completing activity i at time t
- d_i the deadline of activity i
- $G = (V, A)$ a precedence graph that indicates what activities are dependent on other activities.

3.1.2. Fixed Assignments

In the normal situation, every activity can be completed by a certain set of persons that are qualified for carrying out that activity. However, there can be cases in which the human planner wants to assign a certain person to a certain activity.

To add this restriction to the problem aforementioned we add a special qualification for this person and let the specific activity require this specific qualification. In the model, this will result in the set of total persons capable of performing the activity being reduced to a single person.

3.1.3. Predetermined Time Frames

In a planning there could be activities that have to be carried out at a specific time, in our described problem this could be an activity like eating.

The idea to map this to Job Shop with MPM problem is to let the function that gives the cost of completing an activity at time t, return a high cost for this specific activity if it is completed after the deadline. The release time can be seen as the required starting time and the deadline as the required ending time, thus creating the deadline in the following way: $\mathrm{r}_i + \mathrm{p}_i = \mathrm{d}_i$.

By doing this, the activity has to start at the release time, as otherwise a high penalty is incurred.

3.1.4. More Than One Person On An Activity

Some activities cannot be carried out by a single person, thus multiple persons are required at the same time. A standard Job Shop problem requires one person to be used [2], but in our case the situation can occur that an activity needs multiple persons.

Our idea to get this working in the Job Shop with MPM problem is by using "virtual machines". These machines would consist of subsets of all capable persons. Basically, by having "machines" be teams we can assign multiple persons to a task.

This idea is ruled out because of the increase in time required for a calculation and problem restrictions. These "virtual machines" can be assigned to an activity, but at the same time the persons in this "virtual machine" cannot be assigned to another activity themselves, as they are already busy with a group activity. This cannot be modelled in this problem.

Besides, by having every possible selection of k people from a pool of n you get $\frac{n!}{(n-k)!k!}$ possible selections. For example having a pool of 30 cooks, and needing 10 cooks, has a total of more than 3 million possible subsets. This certainly does not help with the runtime, nor the feasibility of the mapping. It stops the mapping from being polynomial in both time and space.

Alternatively, the activity that requires multiple persons to carry it out at the same time could be duplicated. By duplicating the activity, multiple persons can be assigned to the same activity. This does have its downsides however: this approach can cause entangled activities to shift which means that persons would stop working together and perform sub-activities different times. Avoiding this requires you to have fixed starting and ending times, which is usually not possible as the algorithm should generate the schedule.

3.1.5. Diverse Schedule

When persons are carrying out the same activities all the time, they might lose their dexterity for other activities. A person should thus not be scheduled for the same set of activities all the time.

We could not think of a way we could map this requirement onto the Job Shop problem.

3.2. Shop Problem with Multiprocessor Tasks and Arbitrary Processing Times

This, again, is a shop problem. It is very closely related to our problem, but there is a small difference. With multiprocessor tasks, tasks can use multiple processors, "machines". However; according to the definition [2] tasks need to use all processors capable of performing the task at once. This does not hold for our problem, where activities (tasks) only use a subset of all qualified persons (processors).

3.3. Resource Constrained Project Scheduling Problem

As the name suggests, the Resource Constrained Project Scheduling Problem (RCPSP) is about creating a schedule for a project that has to be performed using constrained resources [1]. A project consists of multiple activities that each use a predefined amount of resources at predefined stages. Resources can be renewable or not. Being renewable means that once a sub-activity is finished with the resource, it is released to be used again by another sub-activity.

Activities can also have precedence relations with other activities. This creates constraints on the order in which sub-activities are carried out. The goal of RCPSP is to find a starting time for each activity so that the number of resources needed at any time is not greater than the number of resources available, while minimising the makespan.

The formal notation for RCPSP is [3]: $I = (A, R, P, d, u, c)$.

- A - Activities
- R - Resources
- $P \subseteq A \times A$ - Precedence constraints for activities. $a_i \prec a_j$ implies that activity i must be completed before j is started.
- $d : A \to \mathbb{N}^+$ - A function that maps activities to resources needed.
- $u : (A \times R) \to \mathbb{N}$ - A function that maps resource usage of an activity of a specific resource.
- $c : R \to \mathbb{N}^+$ - A function that specifies how much of a specific resource is available.

Our problem maps in the following way to RCPSP:

- A - A collection of activities.
- R - A collection of persons.
- P - The precedence of activities that need to be completed, e.g. cooking before eating.

- d - A function that specifies which persons are needed for certain activities.
- u - A function that indicates how many persons an activity needs.
- c - A function that indicates how many persons are still available that can carry out a certain activity.

At first glance, this problem looks a lot like the problem described by TNO. The persons can be seen as the renewable resources; the sub-activities can be seen as the activities, a lot of similarities. However; there are also some differences that make it difficult, if not impossible, to use this mapping in practice.

3.3.1. The Objective Function

The objective function of RCPSP is C_{max}, which means that algorithms should try to minimise the makespan, the amount of time needed to complete the project overall [2]. However; the objective of the TNO problem is not to minimise the makespan, because it consists of making a schedule for a ship. On a ship, it doesn't matter how early activities are completed as long as they are completed before their deadlines. There is no additional benefit from completing activities ahead of time. Unfortunately, minimisation of the makespan is at the bottom of possible objective function mappings [2], so any other objective function we could use, cannot be mapped to minimising the makespan.

3.3.2. Persons Represent Multiple Kinds of Resources

Persons can carry out multiple kinds of activities. For example, a communications officer could also be capable of cleaning or cooking. This means that each person can represent multiple kinds of resources. However; this also means that resources are entangled with each other. Once one resource of one kind is used, the availability of other kinds of resources could decrease as well. This is not part of RCPSP and is very likely to make heuristics used in algorithms ineffective.

These limitations make it impractical to map our problem to RCPSP and makes it impossible to use existing (heuristic) algorithms designed for solving RCPSP problems.

3.4. Mixed Integer Programming

> A (linear) mixed integer program (MIP) is an optimization problem over a set of integer variables (unknowns) and a set of real-valued (continuous) variables, the constraints are all linear equations or inequalities, and the objective is a linear function to be minimized (or maximized). [10]

This quote from Wolsey explains clearly what a Mixed Integer Program (MIP) is, however it is not exactly clear yet how our problem maps to this problem. In theory defining

how our problem would be represented in a MIP would be to solve our problem. The constraints we come up with can be put into a MIP solver which would try to find a solution that fits within the constraints. For a full formal explanation of what MIP is, we recommend paper [10] from Wolsey as we found it very clearly explaining the concept. In the remainder of this section, we explain it using examples that fit our problem.

At the moment of writing, we believe we should be able to model our entire problem using MIP. The only problem we might encounter is that our problem consists of too many variables or constraints which would result in an unreasonable runtime. Besides dropping more requirements, we might try to minimise the number of constraints we use.

3.4.1. Example Mapping

As an example, we model a small part of our problem as a MIP.

$$a_{ijt} = \begin{cases} 1 & \text{Person } j \text{ carries out activity } i \text{ at time } t \\ 0 & \text{Otherwise} \end{cases}$$

Lets assume we have n activities, m persons and T blocks of times. This means that $a \in \{0,1\}^{mnT}$. The constraint that persons can only perform one activity at maximum at a time can be modelled like this:

$$\sum_{i=1}^{n} a_{ijt} \leq 1 \text{ for } j = 1, \ldots, m; t = 1, \ldots, T$$

The constraint enforces that a person is only assigned to one activity at most by summing over all assigned activities for each person and time interval and requiring the result to be less than or equal to 1.

3.5. Constraints Satisfaction Problem

During the research phase we have looked into the Constraints Satisfaction Problems (CSP) [11]. A CSP instance consists of a three tuple (X, D, C) with:

- X a set of variables
- D a function that maps the variables to a set of finite values
- C a set of constraints.

Each variable has a domain of values that cannot be empty. The constraints represent a subset of the variables and specify which combination of values are allowed for this subset.

A solution to the problem is an assignment of values to variables without violating a single constraint [8].

3.5.1. Problem Example

- $X = \{x, y\}$
- $D : X \to \mathbb{N}^+$
- $C : \{X \neq Y\}$

A solution for this problem example is $\{x = 1, y = 2\}$

CSP is very similar to MIP. There exist some good solvers for MIP, like Gurobi[2], that are very good in solving numerical constraints. Since our problem is quite numeric, using a numerical solver, for problems like MIP, is faster than turning it into a logic problem for another kind of solver, for problems such as CSP. For this reason, it seems better to focus on MIP instead of CSP.

3.6. Summary

Job Shop with Multi-Purpose Machines (JSMPM)
Mapping our problem to JSMPM is not feasible because our problem can require a task to use multiple persons at once, however, the standard Job Shop problem does not support this. The only way we could think of to solve this causes an exponential blow up in the search space, which makes using it infeasible.

Shop Problem with Multiprocessor Tasks and Arbitrary Processing Times (SPMTAPT)
SPMTAPT is closely related to our problem, but it requires all machines capable of processing a task to work on it at once. For our problem, activities can use less than all qualified persons.

Resource Constrained Project Scheduling Problem (RCPSP)
It seemed that RCPSP looks very similar to our problem, but there are two differences that make the mapping difficult in practice. The first is the objective function as our problem is not concerned with the minimisation of the makespan. The second is that persons can have multiple qualifications, which requires a dependency relationship between the resources. This makes the mapping impracticable and the use of existing (heuristic) algorithms impossible.

Mixed Integer Programming (MIP)
We think that we can create a model for our problem in MIP, for which existing solvers exist (see section 4.3). The only problem we can think of is that we have too many variables or constraints, which can put pressure on the runtime.

[2] `http://www.gurobi.com/products/gurobi-optimizer`

Constraints Satisfaction Problem (CSP)
CSP seems to be very similar to MIP and we think that it is also possible to map our problem to CSP. Processing numerical constraints is often easier than processing logical constraints. For this reason, it seems better to focus on MIP instead of CSP.

4. Related Technologies

4.1. Existing Solutions To Similar Problems

Planning is a common problem in many business ventures, from scheduling tasks and shifts to creating a production planning. There are a lot of problems in practice that are inherently a planning problem. Because of this, there is a lot of development and investment in this field, some of which lead to products that solve problems similar to ours.

1. **Odoo Production / Projects**[1] - A production and projects planning tool
2. **frePPLe**[2] - A production planning tool
3. **EzShift**[3] [4] - A shift and human planning tool
4. **Genius Resource Management**[5] - A resource management tool, allows for human scheduling.
5. **Float**[6] - Project scheduling tool.
6. **Comindware Resource Management**[7] - Project resource management tool, allows for human scheduling.
7. **Deltek Project Management**[8] - Has planning interface that uses Gantt charts[9].

Each of these tools has some interesting attributes and choices that can be taken into account while developing our own software. Every single one of them is a planning tool, though each in their own way.

Most tools are aimed at human planning, though frePPLe (2) is aimed at production planning on machines. FrePPLe (2) however, uses some more advanced algorithms to keep the planning valid; it reschedules items to keep constraints satisfied automatically. Odoo Production, Genius, Float and Deltek do not perform any automatic scheduling.

[1] `https://www.odoo.com`
[2] `https://frepple.com`
[3] `http://ezshift.com`
[4] Website copyright was last updated in 2010, might no longer be developed
[5] `https://www.geniusproject.com/resource-management-software`
[6] `https://www.float.com`
[7] `https://www.comindware.com`
[8] `https://www.deltek.com/en/products/project-and-portfolio-management`
[9] website is very business oriented and does not contain much info about the tool itself.

But Float and Deltek have very modern interfaces and are therefore interesting for ideas for the interface. EzShift seems to be able to perform automatic scheduling according to rules, but only to predefined shifts, and is seemingly no longer maintained.

None of these fully cover the problem, frePPLe is for production planning - not user planning - Odoo Projects (1), Genius (4), Float (5), Comindware (6) and Deltek (7) all do not have automatic planning features and EzShift (3) only works to create shift plannings. In many cases the planning is quite coarse: some tools have a timespan of a single day, rather than hours. Hence another solution is required to solve the problem.

4.2. What Technology Can Be Used?

In order to develop a product, certain technologies, besides the algorithm, have to be used. For example: programming language, libraries, testing software, build tools, version control, and even which browsers to keep in mind in case of websites or web apps.

Our product has a few components, the algorithm, the server and the client side interface, each of which has their own set of used technologies. Additionally, there are global tools which are used in general development and for all components. Due to requirements, a connector is also a component that has to be taken into account.

4.2.1. Development Tools

In order to be able to perform programming tasks in general a version control system, a place to host the code, and a service to monitor the current state of the code is needed.

Version Control System

Our choice: **Git**[10] - *Alternatives: Mercurial*[11]*, Subversion*[12]

Git is the version control system we have been using since the beginning of our studies. Git supports having both a shared remote and a local copy of the code base, allowing us to work while the shared remote is not accessible, though still allowing us to work together in a convenient manner, unlike Subversion. Mercurial is a good alternative to Git. Git is also used within West IT, combining this with the fact that no-one in our group has any experience with Mercurial, we chose Git: the system we have the most experience with.

[10] `https://git-scm.com/`
[11] `https://www.mercurial-scm.org/`
[12] `https://subversion.apache.org/`

Version Control System Hosting

Our choice: **GitHub**[13] - *Alternatives: GitLab*[14]*, BitBucket*[15]

In order to be able to work together, we need remote storage for our version control. For this there are a few most prominent choices: GitHub and GitLab which only support Git, and BitBucket which supports Git and Mercurial. GitHub has an API with an ecosystem of apps surrounding it, from CI (Travis CI) to Code Quality (Code Climate). BitBucket and GitLab have an API too, but fewer applications use it. GitHub hosts a lot of projects - including some of ours, private repositories are free for students and we have performed a lot of projects using it. Recently GitHub has been improving their code review tools, making them about equal or better than GitLab and BitBucket in this regard.

We chose GitHub, especially since the largest benefit for choosing GitLab or BitBucket seem to be user/permission control, which is generally not important for us as students, but maybe more so for an enterprise. Our experience with using GitHub has been positive and setting it up has been quite easy.

Continuous Integration

Our choice: **Travis CI**[16] - *Alternatives: CircleCI*[17]*, Wercker*[18]*, GitLab CI*[19]*, Semaphore*[20]*, Codeship*[21]

In combination with GitHub, using Travis is as easy as having a config file for Travis - what to do - and turning it on, unlike the alternatives. CircleCI, Wercker, Semaphore and Codeship however, also support BitBucket, which means you are less bound to one platform. GitLab CI can only be used with GitLab itself, as it is integrated. We have had experience with both Travis CI and Wercker, we found Travis CI to be easier to set up and had fewer problems in the longer run. However, Wercker has more advanced capabilities which might be interesting for larger projects. Codeship has a very limited amount of free builds available and Semaphore is only free for open source, hence they are both out of the question.

With Travis CI being free for students, being less restricted and in our experience easier to set up than the other options, like CircleCI and Wercker, we chose Travis CI as the tool of choice.

[13] `https://github.com/`
[14] `https://gitlab.com/`
[15] `https://bitbucket.com/`
[16] `https://travis-ci.com/`
[17] `https://circleci.com/`
[18] `https://www.wercker.com/`
[19] `https://about.gitlab.com/features/gitlab-ci-cd/`
[20] `https://semaphoreci.com/`
[21] `https://codeship.com/`

Code Coverage

Our choice: **CodeCov**[22] - *Alternatives: Coveralls*[23]

Since Coveralls is only free for open source, we went with CodeCov as they have a single free private project. Coveralls integrates with GitHub and BitBucket, while CodeCov also integrates with GitLab - in addition to GitHub and BitBucket.

We chose CodeCov since it has more features and is free in our case, unlike the alternative.

Code Quality

Our choice: None yet - *Alternatives: Code Climate*[24]*, Tools-per-programming language*

Having a tool that automatically keeps track of certain code quality properties is useful as it marks issues before they become unmanageable. Manually setting this up with already existing tools for every programming language is an option, though is labour intensive and generally does not give a centralised overview. Code Climate is a service that simplifies this, by giving per commit quality information and having a centralised interface, however: for non-open source software this is paid and getting a student license might not be applicable.

4.2.2. Algorithm

Environment

Our choice: **External Solvers** - *Alternatives: Writing a custom implementation in Python, Rust, C++*

During our search for algorithms to solve our problem we found that solving a problem can be very time consuming, a good solver is therefore a must if there are time requirements. In our case we have some time requirements and therefore writing a good solver ourselves could end up costing a lot of effort for sub-par speed, an already developed solver - for example Gurobi[25] for Mixed Integer Programming - could let us obtain better results faster, though with the negative side effect of being dependent on such a solver.

Implementing algorithms ourselves is not the best choice. Implementing, debugging and integrating algorithms will not be an efficient use of our time, because we don't have enough knowledge to do this quickly enough. Additionally, external solvers are developed by specialised companies and will be optimised in ways we will not be able to understand with our limited knowledge.

[22] `https://codecov.io/`
[23] `https://coveralls.io/`
[24] `https://codeclimate.com/`
[25] Gurobi (`http://www.gurobi.com/products/gurobi-optimizer`) a solver for multiple problems. More examples and a benchmark are given under 4.3 Existing Solvers

The best choice for us is to reuse work done by others, allowing us to focus on developing other parts of the product.

4.2.3. Server

Programming Language

Our choice: **Python** - *Alternatives: NodeJS (Javascript), Java, Rust, C++, PHP, ...*

The server has to be an API endpoint and the part that serves the web app. It also needs to perform the translation for the External Solver if we use one. Gurobi - an example for a solver - has bindings available for Python, and writing a web server in Python is reasonably easy using libraries, making it a good choice.

Testing

Our Choice: None yet - *Alternatives: Unittest, Nose, PyTest*

For python there are three testing frameworks that are most commonly used and supported: Unittest, Nose and PyTest.

Unittest is included by default, but does not have a built-in test runner and requires you to run a function to run the tests. Because of this, it requires - in addition to the test boilerplate - some additional boilerplate for running the tests.

PyTest and Nose both have a test runner and do not require a class as part of test boilerplate - though it is an option in both cases. PyTest goes with convention - predefining function names for actions such as set up for tests, while Nose uses annotations to declare these special actions.

Since we have no prior experience with testing with python but prefer to not have excessive amounts of boilerplate our choice comes down between PyTest and Nose.

4.2.4. User Interface

Our Choice: Vue[26], Bootstrap[27] - *Alternatives: React[28], Angular[29]*

To create user interfaces, we need some kind of view controller system. Since modifying the DOM is a heavy operation a system that utilises a virtual DOM has our preference.

React is a JavaScript library developed by Facebook for building web UI's. User interfaces are created by implementing render functions on components or by adding components to a virtual DOM using a templating language. This framework does not have our

[26] `https://vuejs.org/`
[27] `https://getbootstrap.com/`
[28] `https://facebook.github.io/react/`
[29] `https://angular.io/`

preference, since views and view controllers are very tightly coupled. View declaration is done in controllers which makes view reuse hard.

Vue is another JavaScript UI library that is based on the MVVM pattern. The library allows building templates separately from the JavaScript script which means that views and logic are decoupled. Just like React, Vue makes use of a virtual DOM which allows for high performance when modifying the DOM.

Angular is similar to Vue and React as it also has support for components and data binding. However; Angular has a much steeper learning curve and is less performant.

Because Vue has the highest performance, a clear separation between views and view controllers and easy templating, this is the library we will use. We have also chosen to use Bootstrap for styled components like buttons. This is merely an aesthetic choice.

4.3. Existing Solvers

As explained in the section 4.2.2, our problem can be turned into a MIP problem, and therefore an existing MIP solver[30] can help us. While we could implement algorithms such as Branch-and-Bound ourselves, implementing this algorithm properly is quite time consuming and excellent implementations are already available, both commercial and non-commercial.

1. **Gurobi**[31] - Commercial
2. **IBM ILOG CPLEX**[32] - Commercial
3. **Cbc**[33] - Non-commercial
4. **SCIP**[34] - Academic - Non-commercial
5. **GLPK**[35] - GPL - Open Source

Gurobi is the current state of the art solver. In a recent benchmark [7] it solved 87 problems out of 87 total, within a minute on average.

CPlex is a close second in performance, being a few seconds slower and solving one problem less when using multiple threads. It can be a good alternative to Gurobi.

Gurobi and CPlex are not the only solvers available however, Gurobi is commercial and costs money outside of academic usage. Cbc, SCIP are both non-commercial solvers and although slower and solving fewer problems they are generally available for free. GLPK

[30] MIP has a lot of already existing solvers, as listed, while other problems do not have this benefit.
[31] `http://www.gurobi.com/products/gurobi-optimizer`
[32] `https://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/index.html`
[33] `https://projects.coin-or.org/Cbc`
[34] `http://scip.zib.de/`
[35] `https://www.gnu.org/software/glpk/`

is an open source solver, but seemingly hasn't been updated in a while and an older benchmark has shown sub-par performance in some cases.

5. Design

5.1. System Design

We would like to develop our software solution in a modular way. Different components, like the user interface, a connection to Odoo and the planning algorithm will be separate pieces of software that communicate with each other through a clearly defined interface. This section will discuss the advantages and disadvantages of each module being a separate module.

5.1.1. Independent Algorithm Module

Advantages

One of the most compelling reasons to implement the planning algorithm outside of Odoo is that it can be easily reused in environments where Odoo is not used. During the research phase of the project, an interview with Patrick Hanckmann from TNO was conducted. From this meeting, it became apparent that Odoo is not used in the Dutch Royal Navy at the moment. This means that the solution would be very hard to use by them if it is entirely implemented in Odoo. To make sure that at least some of the solution can be used, the planning generation algorithm will be written in a separate module.

Another benefit of writing the algorithm separately is that we are free to choose a programming language to implement the algorithm in. Odoo is written in a combination of Python for the back-end and JavaScript for the front-end. Since Python is an interpreted language, its performance is not as high as compiled languages' performance.

Disadvantages

Developing the algorithm outside Odoo means we can no longer make use of Odoo's test framework MQT. We need to use some other test framework depending on how we implement the algorithm.

5.1.2. User Interface Outside Odoo

Advantages

The user interface might be implemented as an Odoo module, since West IT is a distributor of Odoo software and is currently looking into smart planning solutions. We are considering building the user interface as a web app that can be used in any browser. This means that even the user interface can be used without using Odoo. It is still possible to integrate the user interface in Odoo through the use of an iframe[1].

Disadvantages

To embed a user interface in Odoo an Odoo module would have to be developed, but this module would contain minimal functionality and implementation would most likely be trivial.

Running the user interface outside Odoo would require us to set up a (local) web server. However; this is not a very big problem since libraries, like Flask[2], would only require us to register endpoints similar to how Odoo does this.

5.2. User Interface Design - Visualising A Planning

To work with a planning the user should be able to understand what you give to him. There are different ways to visualise a planning, we researched those we thought to be most feasible for visualising a planning.

5.2.1. Calendar

A calendar shows what tasks are done at what times. A very simple example of this is the standard calendar application in macOS (figure 5.1a). It shows agenda items over time including locations.

Currently, there are some existing planning products that make use of calendar views to visualise their planning. From section 4.1, there are two of these products: EzShift and Float.

Dependencies between items cannot be shown in this manner, but we could do this ourselves by simply adding arrows between dependent tasks. Assigned persons could be shown similarly to how locations are displayed in the calendar. However; this might become messy when many people are assigned to a certain task, or when tasks are short.

[1] `https://www.w3schools.com/tags/tag_iframe.asp`
[2] `http://flask.pocoo.org`

5.2.2. Gantt Chart

A Gantt chart (figure 5.1b[3]) is a way to visualise a planning that can be used in two ways [2]. A Gantt chart consists of rows and a time line that is represented along the horizontal axis. The current time can be indicated through a line perpendicular to the time line.

The first way allows planners to see their planning from the perspective of the activities, i.e. every row represents an activity. Using this configuration the planner can see when sub-activities are performed and by whom.

Dependencies between activities can also be shown using arrows between the end time of the first activity and the start time of last activity.

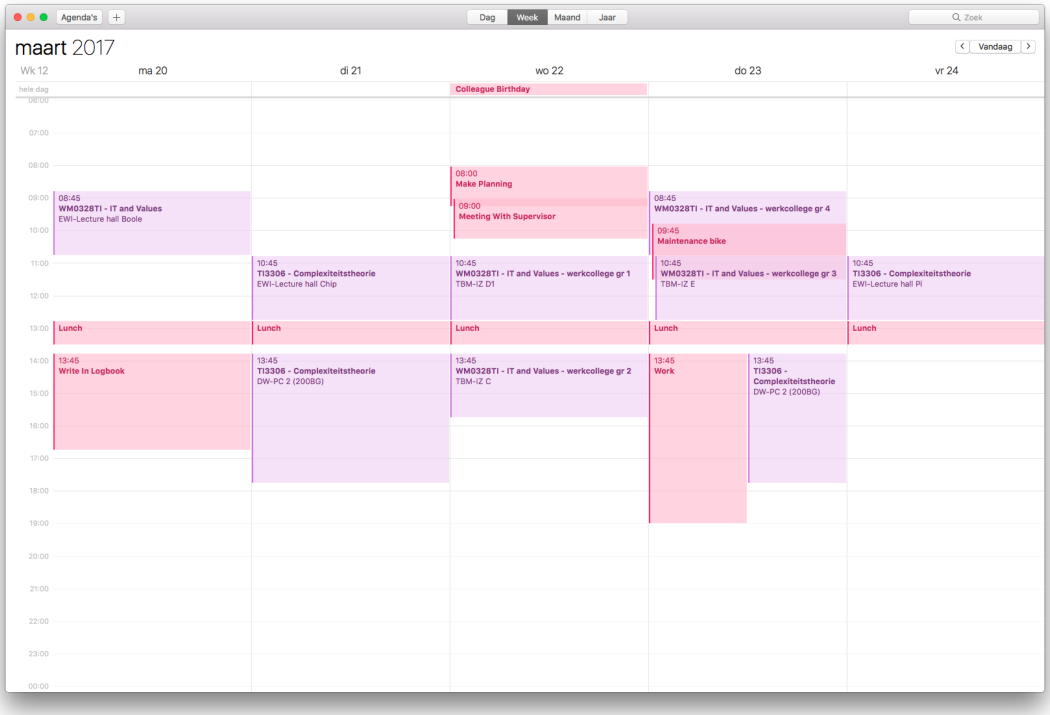
The second way a Gantt chart can be used is by the rows representing persons. This can roughly be seen as a stack of personal timetables. In this representation it is possible that an activity shows up multiple times in the chart, because an activity can be performed by multiple people at once. Sub-activities do not always have to be shown and can be abstracted away by creating a bar that represents the entire activity. This bar spans from the start time of the first sub-activity to the end time of the last sub-activity.

Currently, there are different software products that make use of Gantt Charts for visualising a planning. From the solutions of similar problems, section 4.1, frePPLE, Genius Resource Management and Comindware Resource Management are all using Gantt Charts for their visualisation. It shows that it is a common way to visualise a planning in the industry.

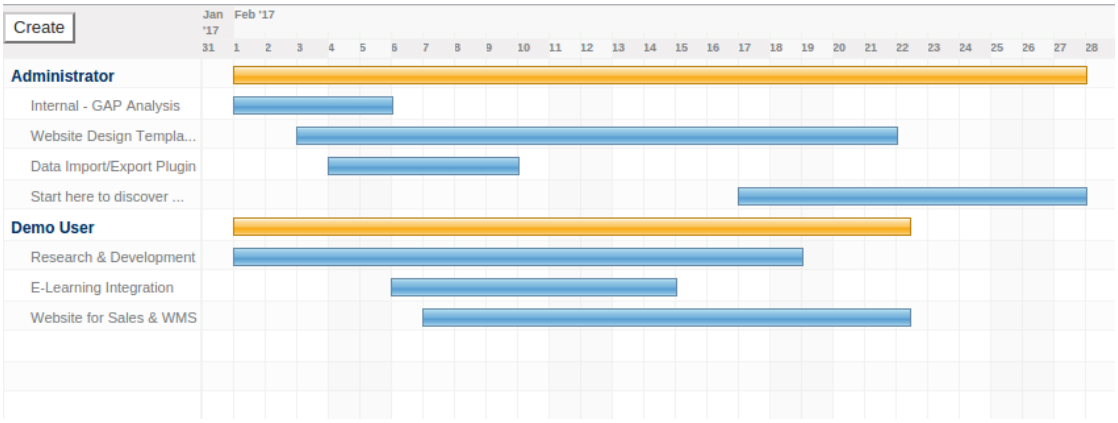
In our project, the second representation is more useful as we have to create an assignment of persons to activities. Indicating the current time along the horizontal axis might be helpful when looking at an already generated schedule, but is not very useful when creating a new schedule as this is always done for the future.

If a row in a Gantt Chart represents a person, the total number of rows will be equal to the amount of persons a planning has to be created for. As a planner you want to see the entire schedule, thus there is no way to get around this growth without removing persons from the planning overview.

[3] `https://www.odoo.com/apps/modules/10.0/project_gantt_view/`



(a) Calendar Application in macOS



(b) Gantt Chart in Odoo

Figure 5.1.: Planning Visualisations

6. Evaluation

To evaluate the project and the solution, we have defined success criteria for both. In the first section we discuss the success criteria for the project and in the second section the success criteria for the solution.

6.1. Evaluation Of The Project

This section describes the success criteria to be used to evaluate the project and our reasons for having them.

The following success criteria are to be used:

- The deadlines during the project have been met,
- The client and coach are pleased with the performed work,
- The process is properly documented,
- The students have met the educative goals of the project,
- The solution fulfils its success criteria.

6.1.1. Deadlines Have Been Met

At the start of the project different deadlines have been set, documented in the plan of action. The deadlines include deadlines for reports for the university and development milestones for the client. The deadlines should be met to achieve our goals for this project.

6.1.2. The Client and Coach Are Pleased With the Performed Work

At the end of the project there will be a meeting with both the client and coach. From this meeting, we can conclude whether they are happy with how we performed during the project, and with the final product. We can use their assessment to decide whether the project was a success.

6.1.3. The Process Is Properly Documented

The process of the project consists of different aspects and every aspect should be properly documented so it can be evaluated later on. The following aspects should be documented:

- Project Plan that describes how we managed the project
- Meeting Minutes that describe what has been discussed during meetings with the client or coach
- Research Report that describes our research of the problem
- Sprint Plans that describe the work that was planned for a certain sprint
- Sprint Retrospectives that describe our evaluation of the previous sprint
- Final Report that gives a complete overview of the project.

6.1.4. The Students Have Met The Educative Goals

This project knows a list of learning goals[1] set by the university at the beginning of the project. These learning goals should be met as otherwise the students failed to gain the knowledge this project was designed to provide them with.

6.1.5. The Solution Fulfils Its Success Criteria

The solution itself has a list of success criteria as well. In order for the project to succeed, the solution should also succeed with few remarks. A success criterion that has not been met for the solution does not mean the project has been a failure. Situations can occur in which a success criterion cannot longer be met. If properly motivated why the success criteria could not be met, insight is still gained and this success criterion is still met.

6.2. Evaluation of the Solution

This section describes the success criteria to be used to evaluate the solution and our reasoning for having them.

The following success criteria are to be used:

- The solution meets all 'Must Haves',
- The solution meets most 'Should Haves',
- The solution is properly documented,

[1]The learning goals can be found in the general guide of the bachelor project for computer science studentsL `https://bepsys.herokuapp.com/files/GeneralGuideTUDelftCSBachelorProject.pdf`

- SIG feedback is positive,
- The solution is tested well.

6.2.1. The Solution Meets All Must Haves

The solution must meet all 'Must Have' requirements as these requirements describe the minimal functionality. If the solution does not meet these requirements, the solution is not really a solution and the product cannot be considered a success.

6.2.2. The Solution Meets Most Should Haves

Even though the 'Must Have' requirements describe the minimal functionality of the product, it is a big advantage when the solution meets most 'Should Haves' as well. If this is true, the solution is more complete and is more complete for the client.

6.2.3. The Solution Is Properly Documented

To make sure that the client can extend and maintain the product easily in the future the code should be documented properly. This means that the overall system architecture, external interfaces and the most important components are documented in detail. The system architecture documentation gives an overview of how different components work together, e.g. the user interface and the algorithm. The external interface documentation describes the endpoints of the server, e.g. for communicating with the UI. The component documentation gives an overview of how these components work internally, e.g. how the Odoo connector transforms Odoo's data into an internal representation.

6.2.4. SIG Feedback Is Positive

A few weeks before the end of the project, the code base has to be uploaded to SIG. They will evaluate our product and advise on what parts need improvement from the perspective of code quality. SIG looks at 10 different aspects of our code base [9]:

1. Unit length,
2. Unit simpleness,
3. Duplicate code,
4. Small interfaces,
5. Separate concerns in separate components,
6. Loosely coupled top level components,

7. Well balanced components,
8. Small code base,
9. Automated testing,
10. Clean code

Every component will get a rating together with an explanation. A positive evaluation at the end of the project steers us to conclude that the project is a success.

6.2.5. The Solution Is Tested Well

SIG recommends that tests should cover at least 80% of the code base. To make sure this number is actually useful, we write unit tests, integration tests and, in case we discover bugs, regression tests [9]. Additionally, we organise at least one acceptance test during which the client will assess whether the product works as desired. Being convinced the product is tested well is another reason to believe the project is a success.

Bibliography

[1] J. Blazewicz, J. K. Lenstra, and A. H G Rinnooy Kan. Scheduling subject to resource constraints: classification and complexity. *Discrete Applied Mathematics*, 5(1):11--24, 1983.

[2] Peter Brucker. *Scheduling Algorithms*. Springer-Verlag, Berlin, 5 edition, 2006.

[3] Frits de Nijs. *Project Scheduling: The Impact of Instance Structure on Heuristic Performance*. PhD thesis.

[4] K.A. Dowsland and J.M. Thompson. Solving a nurse scheduling problem with knapsacks, networks and tabu search.

[5] Michael R Garey and David S Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1990.

[6] Johann Hurink, Bernd Jurisch, and Monika Thole. Tabu search for the job-shop scheduling problem with multi-purpose machines. *OR Spektrum*, 15(4):205--215, 12 1994.

[7] Hans Mittelmann. Mixed Integer Linear Programming Benchmark (MIPLIB2010) 14 Apr 2017, 2017.

[8] Peter Norvig and Stuart Russel. Constraint Satisfaction Problems. In *Artificial Intelligence: A Modern Approach*, chapter 5 - Constr, pages 137 -- 160. 2 edition, 2002.

[9] Joost Visser. *Building maintainable software : ten guidelines for future-proof code*.

[10] Laurence A Wolsey. Mixed Integer Programming. *Wiley Encyclopedia of Computer Science and Engineering*, (1):1--10, 2008.

[11] Makoto Yokoo, Edmund H. Durfee, Toru Ishida, and Kuwabara Kuwabara. The distributed constraint satisfaction problem: formalization and algorithms. *IEEE Transactions on Knowledge and Data Engineering*, 10(5):673--685, 1998.

A. TNO Interview Summary

TNO Interview Summary

Present:
Cas Buijs (TU Delft)
Arthur Guijt (TU Delft)
Léon Planken (West IT Solutions)
Patrick Hanckman (TNO)
Lars Stegman (TU Delft)

Date: 26/04/2017
Time: 10:30 - 12:00

Patrick described the following situation of the Dutch Royal Navy:
A ship requires a certain amount of staff members that represents a certain set of skills. Every staff member has certain skills and thus can be assigned to a certain set of tasks. Currently a planning is created by a Personnel Officer and a group of other people, mostly by hand. The process is quite inefficient and it takes a long time to complete, as it is made by humans the created planning does not have to be as efficient as it could be. There is an aim to reduce the amount of redundancy in terms of staff members on board of a ship, which requires an efficient use of staff members. For every task there should be a limited redundancy (for example, in case of sickness of the qualified staff member). There are tasks that are so specialised that only a few people are able to perform them.

Patrick described the following case:
Create a planning tool that can assist the Personnel Officer in creating a planning for the ship. This planning consists of a set of tasks with certain assigned staff members.

The planning should take into account the following conditions:

- Staff members have to sleep 8 hours every 24 hours and have 3 meals a day.
- Staff members have to train every skill once in a while, to not forget how to perform a certain task.
- All staff members have to be busy at all times, there is no spare time

Which leads to the condition that the planning generator cannot create a planning that can be reused every day.

The tool should suggest several plannings that the Personnel Officer can assess and adapt to his/her own insights. The Personnel Officer should be able to make an adjustment to the planning and to generate a new planning, taking into account this adjustment as not flexible. It would be very nice if the tool could explain its reasoning for why certain decisions have been made as people tend to not trust a new system blindly. This explanation is not necessary as it might be very difficult to create. The tool is allowed to take some time for creating a planning, but not too long. An hour is okay, a day might be too long as people will start making the plannings themselves. If the planning is not optimal and some tasks are not fulfilled, the tool should clearly indicate what goals have not been met and which ones have been met. The Personnel Officer could indicate in the tool the priority of tasks, so that the tool can attempt to fulfil these goals first.

If the tool suggests multiple plannings, it should clearly indicate the differences between them and only suggest plannings that have significant differences between them.
The time scale of the planning is dependent on the situation, in a battle situation it is in the scale of 30 seconds, if schedules are made for a person in a normal situation the scale can be 10 minute, 1 hour or a day. The battle scenario is out of scope for this project.
Depending on the time scale, travel time has to be taken into account. A time scale of 1 hour or longer can ignore the travel time. But at smaller time scales - 10 minutes- this can be quite important.

We should keep in mind that some redundancies have to be taken into account as staff members might get sick and their task has to be fulfilled by someone else with the required qualification(s).

With tasks seen from a high level, like navigation, people are interchangeable since they are all trained to perform all subtasks.

Patrick is interested to see what we will come up with, especially in terms of the user interface and how we model the problem.

Agreed tasks:

- Cas, Arthur, Lars:
    - Summarize meeting and send to all present and Mathijs de Weerdt.
    - Inform Patrick of how we use his input in the end.
    - Model the problem to a planning problem.
- Patrick:
    - Make a more detailed list of procedures on a ship.

Bibliography

[1] Peter Brucker. *Scheduling Algorithms*. Springer-Verlag, Berlin, 5 edition, 2006. ISBN 9783540695158.

[2] Bryce Goodman and Seth Flaxman. European Union regulations on algorithmic decision-making and a "right to explanation". 2016. URL http://arxiv.org/abs/1606.08813.

[3] Abdelmonem Mahmoud (https://cs.stackexchange.com/users/34914/abdelmonem mahmoud). Express boolean logic operations in zero-one integer linear programming (ilp). Computer Science Stack Exchange. URL https://cs.stackexchange.com/q/43884. (version: 2015-06-25).

[4] Jon Kleinberg and Éva Tardos. Algorithm Design. In *Algorithm Design*, chapter 3 Graphs, page 100. Pearson Education Limited, 2014.

[5] Sigrid Knust and Peter Brucker. Complexity Results for Scheduling Problems. URL http://www2.informatik.uni-osnabrueck.de/knust/class/.