PLAN A

Denis Panor
Nadav Ravid
Max Brener

Technical/Academic Advisor:
Niv Gilboa

CHAPTER 1: Introduction **3**
1.1 Vision 3
1.2 The problem domain 4
1.3 Software Context 5
1.4 Stakeholders 5
CHAPTER 2: Usage Scenarios **6**
End Users 6
CHAPTER 3 : Functional Requirements **7**
CHAPTER 4 : Non-Functional Requirements **8**
Appendices **9**
Glossary 9
Linear Programming (LP) 9
Zimpl 9
Scip 10

CHAPTER 1: Introduction

1.1 Vision

Rare is the person who at some point worked in a shift based workplace that didn't get a shift in a time he specifically requested not to get a shift on, or a student that got a shift on an exam day, even though he provided his exam schedule to HR.
Creating a shift schedule is hard. Take a work place of 40 people for example, and to simplify lets say each day has 3 shifts which need 4 people, including weekends. The HR rep's job now is to fill out a week with 3x4x7 = 84 places, while accounting for each and every person's limitation. No 2 shifts in a row, no shifts on personal vacation days, sick days or worker requests, while trying to equalize the total number of work hours per person as much as possible.
Now let's unsimplify and say the workers are all students with their own study and exam schedule, and there are more than 3 shifts per day with overlap between them, and include Shabat as a constraint for some workers- now it seems almost impossible for a man made schedule to be 100% correct without additional revisions.
This is where Plan A comes into play. Instead of letting some poor soul handle all the logistics of a complex plan, our project aims that given a problem such as shift scheduling, it will immediately suggest the most optimal solution possible, no revisions required.

1.2 The problem domain

The problem consists of translating a real world problem into a Mixed Integer Program (MIP), which is a known and well researched problem before letting a solver handle the extraction of the solution, if it exists.
Following is the top down view of the problem our project solves, using the example described before:

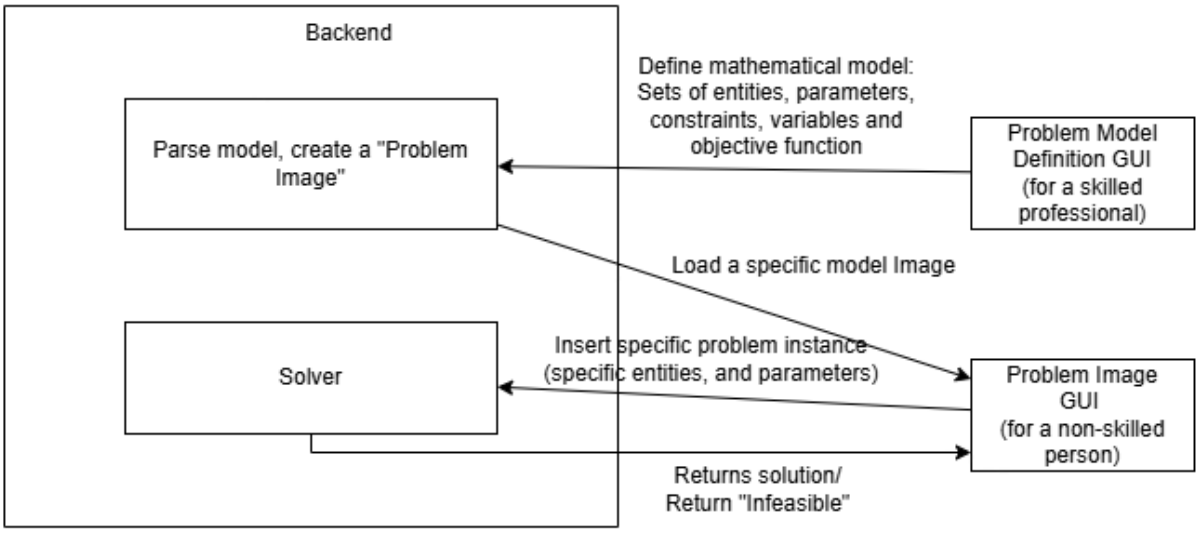
1. FIrst the user defines the space of the problem, be it a predefined set such as a week or month or a custom domain such as a series of custom variables, (e.g. locations, times, people).
2. The user defines the basics of what the program needs to solve- In the previous example it could be inserting people into shift slots such that each person has a minimal amount of shifts per week (thus also equalizing working hours among workers).

3. The user defines any constraints that must be upheld, these can be as basic or complex as the user wants, such as “No worker can be slotted more than once at the same shift”, “No worker works more than 12 hours a day or 2 shifts in a row” Or more case specific such as “No religious worker will be slotted in a weekend shift”. The engine will never ignore constraints, and if there is no feasible solution with them, such will be its answer.
4. Any additional non critical preferences are defined, these are ordered from most to least important, all preferences should be defined with the assumption that they may be ignored, if necessary. The project will try to find a solution while upholding as many preferences as possible, with the logic of ignoring preferences defined on a case by case basis. Examples for preferences are: “Worker A has a party on Monday noon” or “Worker B prefers no to work night shifts” etc.

   All variables, such as preferences and constraints come with a cost function- representing its weight compared to other considerations. The cost is defined by a number, meaningless by itself, that gets its value when considered across multiple other costs.
   Initially, that problem will have to be defined directly while editing the model code, or derived from a pre-existing template of a similar problem. After the initial setup of the problem and its different variables, other users with no technical knowledge can freely edit it to suit their specific day to day.

1.3 Software Context

The user inputs his problem domain in the specialized GUI,



1.4 Stakeholders

The stakeholders of the system could be anyone who takes place in the process of assigning objects into slots, for example - assign workers into shifts :

- Employees:
  Employees are the ones who get affected the most by the system. The result of the system in their case is the shift schedule that they will work by, and obviously their life will be affected by the way we assign their shifts.
  At the end of the day, workers who are satisfied with their shift schedule will probably stay more at their workplace, which is the companies' interest.

- Managers/Supervisors:
  Managers are the main users of the system. Usually, they are the ones to do the work that our system is supposed to do, but they have a lot to do other than that. Letting our system assign the shifts for them will save a lot of time for them and will allow them to be a lot more productive in their other tasks.

- HR/Administrative teams/Inspectors:
  Any kind of stakeholder who needs to verify that everything goes by the rules and laws, will be happy to use our system. Since the user supplies the constraints (the rules) to the system, the stakeholders will be able to see the

rules and the outcome, verifying that everything aligns with contracts, labor laws and organization policies.

Another examples could be :

- University student who needs to build his own class schedule, selecting classes/lectures according to the options he is given.
- Cinemas' managers assign movies into theaters based on movie lengths and movie types.
- Military commanders assign tasks to their soldiers based on their roles and the tasks' requirements.

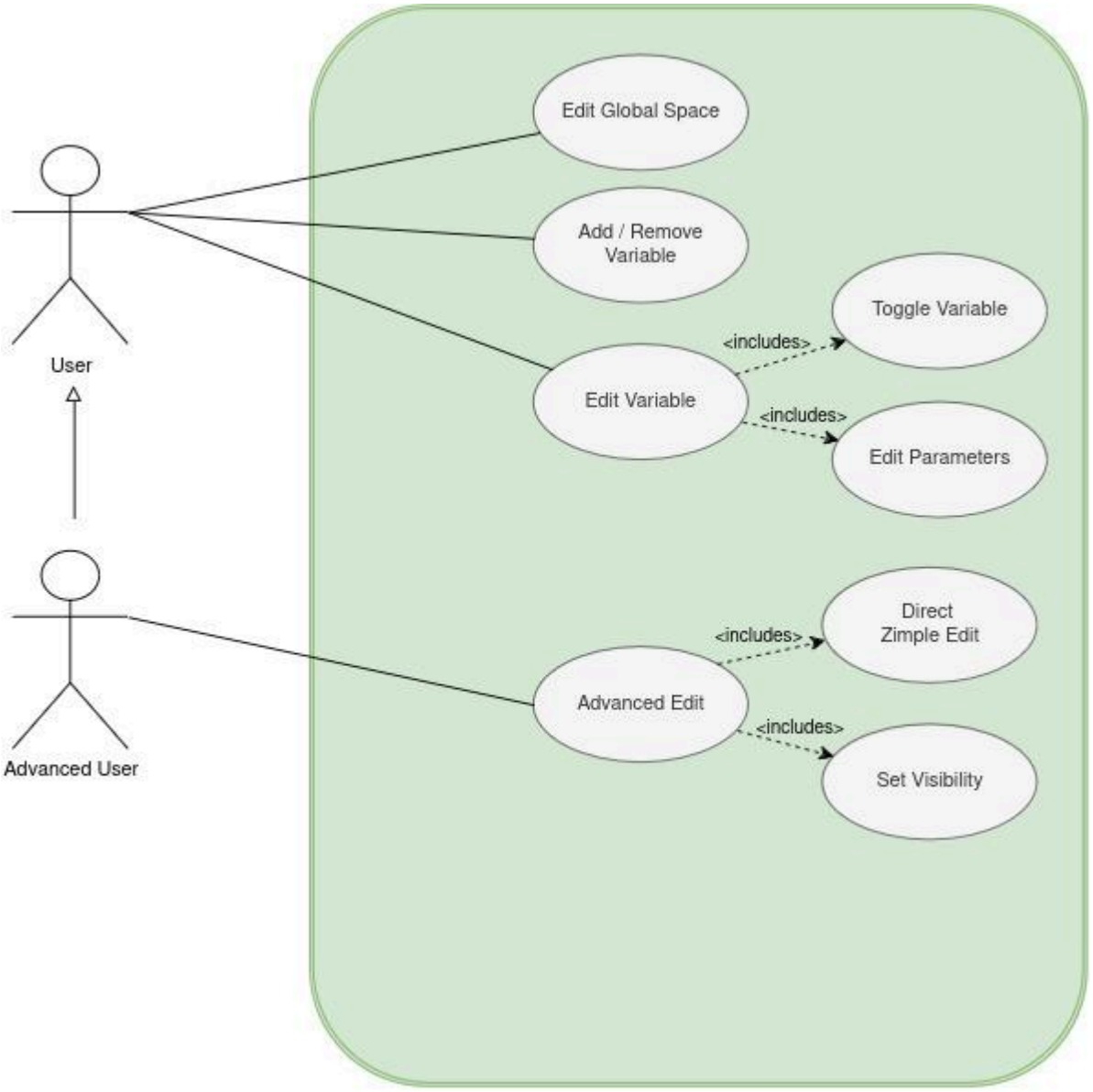
CHAPTER 2: Usage Scenarios

End Users

We separate the user base into 2 categories: A normal user and an advanced user. These user types are freely interchangeable at will, and the advanced user is an extension of the basic user, with the added capability of directly editing Zimpl code and additional power options.
Stemming from the generic nature of the software, the initial settings describing the logic of the space the engine works in, and the optimization functions are impractical to be implemented in a GUI, and thus will have to be implemented in Advanced User mode. An additional future option is to download a template for a space available in an external database domain.

- Basic User: WIll be able to add, remove and edit pre existing template variables. Editing a variable includes toggling it on/off and editing parameters in the variable. In the example that could be a worker, a shift, a variable comprising a time frame and a worker, signifying a time where a worker can or can't work etc.
- Advanced User: Has all the functionality of the Basic User, with the addition of being able to directly interact with the usually transparent .zpl files. Is required when first initializing the program, defining the complex constraints and setting templates for the basic user for the dynamic ones. It is possible to skip this step by using a predefined template, if applicable.

Use Case Diagram:


Edit Global Space
Add / Remove Variable
Toggle Variable
<includes>
Edit Variable
<includes>
Edit Parameters
User
Direct Zimple Edit
<includes>
Advanced Edit
<includes>
Set Visibility
Advanced User

CHAPTER 3 : Functional Requirements

3.1 - Users

- Users can create a personal account
- There are two types of users : basic user and advanced user.
- By creating accounts, users can share their spaces.

3.2 - Space Settings

An advanced user can set up a space for other users to use :

- Define/Edit Global Space: The user will be able to set a range for the engine to work it, ranging from predefined templates such as time ranges (day, week, month etc.) to more custom range, up to the user.
- Variable Editing: A variable is a generic term for all configurable inputs, including constraints or group of constraints, preferences, and sets of entities. A variable is either chosen from a template or defined by the power user, and any user can toggle it and edit its parameters.
- Optimization Equation Definition: An advanced user can (and initially, needs) define an optimization function and basic data groups.

3.3 - Variable and solution

- Users can set constraints in the space they use.
- Users can set preferences in the space they use.
- Users can set the size of the solution (for example, could be a weekly time table, or monthly work arrangement).
- The system will be able to build a solution based on the constraints and space that the user used.
- The given solution will always satisfy all the constraints.

3.4 - Cloud

1. The system will allow an advanced user to upload the environment they created to the cloud where the environment will be saved.
2. The system will offer a search engine for searching for environments in the cloud.
3. The system will allow searching for existing environments according to filters.
4. The system will allow advanced users to provide tags, descriptions, and categories for environments.
5. The system will provide default environments for common uses.

CHAPTER 4 : Non-Functional Requirements

- The user will be presented with a list of constraints and preferences, based purely on predefined templates:
    - Each Template will have a checkbox near it for toggling it on or off.
    - Each template will be comprised of read only informative text describing it, with editable sections of the template's actual values
    - The constraints and preferences will be divided into two groups, the constraints will be an unsorted group while the preferences will be sorted, with the user able to drag them or edit their placement number to change their order.
- An advanced user will have direct access to the optimization function cost values:
    - The user will be able to toggle each function to be considered or ignores
    - The user will be presented with an unnumbered bar describing the relation between optimization considerations. The higher up the bar a function, the higher its cost relative to others behind it and coincidentally it will be prioritized more by the engine.
        - The bar will be discrete and not continuous, to allow setting different costs to be equal.
        - The amount of snapple spots on the bar will be editable, represented by a simple numeric value.
        - The bar will not feature numeric values, instead representing purly the relationship between different considerations. Some trial and error may be required due to the abstract and hard to predict relation between the values and how it affects the engine.

CHAPTER 5 : Risk Assessments

- Linear Programming is an NP-Complete problem, meaning at the worst case (which we cannot predict or measure), any engine can run in exponential time, even at average input sizes will practically never end.
  This problem is unfixable (unless someone is to prove P = NP), but a potential workarounds are:

  - A timeout, set by the user for how long he's willing to wait.
  - A manual cancel switch, when the user notices the engine takes too long.
  - Runtime analytics, to automatically kill the engine if it uses too many resources comparable to what's available.

  Linear Programming solving is an old and well researched problem, so most engines, including scip, are very optimized- A reasonable problem taking an unreasonable amount of time to solve is expected to be very rare, but inevitably still possible and must be considered.
- Linear Programming is naturally hard to predict, thus it's hard to predict what even small changes in weights will do outside of simply running the engine and analyzing its output. No real solution is feasible within the scope of our project but possible aids are:
  - Limiting direct value manipulation to advanced users, and having the values be managed by the backend while transparent from the end user, unless he's chosen to use the advanced GUI
  - Warning and explaining to the user this behaviour.

Appendices

Glossary

- Advanced User- A user that, by choice, has been granted access to advanced utility such as direct r/w to Zimpl code and cost manipulation. A user who doesn't know what he's doing may break functionality, and will so be warned.
- Basic User- WIll only be able to manipulate preset templates, including adding,removing and changing their parameters.

Linear Programming (LP)

An optimization or feasibility problem defined by a set of
all its variables being restricted to integers. A subset of such a problem is Mixed Integer Programming (MIP), which differs from LP by having some or all its variables be non discrete values.
Our project creates models written in the ZImpl language, who are then solved by the Scip solver, detailed below:

Zimpl

Zimpl (Zuse Institut Mathematical Programming Language) is a high level, lightweight programming language designed to model mathematical problems into a linear or nonlinear mixed integer mathematical program expressed in .lp or .mps file format which can be read and (maybe) solved by a LP or MIP solver. Zimpl supports multiple solvers and is well integrated with the Scip solver, who's designed by the same German organization.

Scip

Scip (Solving Constraint Integer Programs) is a solver for LP, MIP and mixed integer nonlinear programming (MINLP), and is one of the quickest FOSS solver frameworks available, designed for absolute control of the solution process and access detailed information deep inside the solution.