Ariel University Observatory

Planner and Scheduler

Software Design Document

23.12.20

Table of Contents

Table of Contents **2**

Introduction **3**

Purpose 3

Scope 3

Overview 3

Reference Material 3

Definitions and Acronyms 3

System Overview **4**

System Architecture **4**

Architectural Design 4

Decomposition Description 6

Design Rationale 7

Data Design **8**

Data Description 8

Data Dictionary 9

Component Design **10**

Human Interface Design **13**

Overview of User Interface 13

Screen Images 14

Screen Objects and Actions 15

Requirements Matrix 16

Introduction

Purpose

This software design document describes the architecture and the system design of the Planner and Scheduler modules of the robotic observatory of Ariel University.

NOTE: The designs and the concepts within this document may change or cancel as the project progresses.

Scope

This project engaged in two modules - Planner and Scheduler. Those components' purpose is to make a way to conduct an observation for research purposes and decide which observing should run based on data, priority and defined conditions. Those modules separate the robotic observatory from the traditional observatory. The robotic observatory is valuable because it enables the university researchers to set much more research thanks to its efficient use of the facility.

Overview

The architecture of the Planner and Scheduler uses two web servers for each and a web user interface. We want to achieve a great user experience, usage of already existing tools, and an efficient decisioning process. In this file we describe the system's design, processes, components , data structures and entitis. For now we are integrating with the ACP program as the control software but it will change in the future.

Reference Material

DC-3 Dreams - Dispatch Scheduling of Automated Telescopes
http://articles.adsabs.harvard.edu/pdf/2004SASS...23...35D

Definitions and Acronyms

ACP - Astronomer Control Panel
SIMBAD - the Set of Identifications, Measurements and Bibliography for Astronomical Data.

System Overview

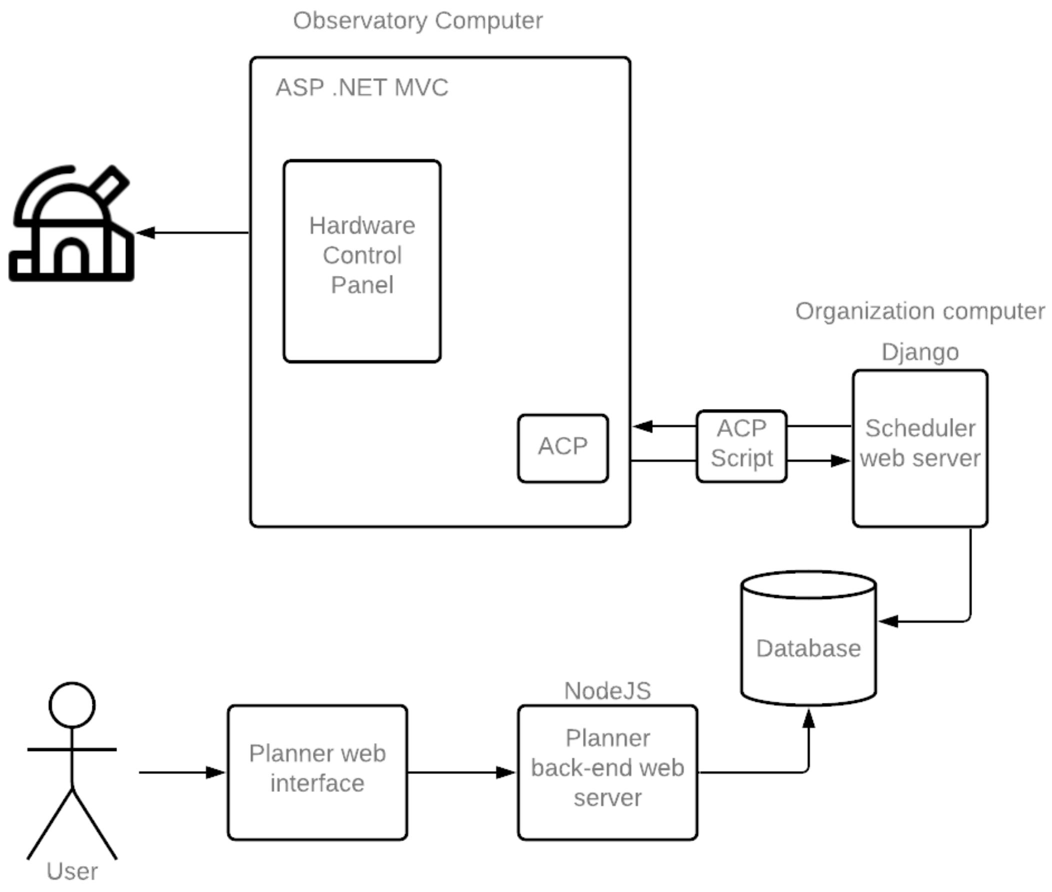
The whole system is built from several units: observatory facility which includes the system's hardware components , control software who operates the observatory peripherals, planner that add new observing plans into the mission bank , scheduler which decide from the missions bank which one should be send to the control for execution, database which help the scheduler and the planner make decisions based on reliable data.
The Planner and the Scheduler need to be connected between themselves through the database. The Scheduler also needs to be connected to the control system.

System Architecture

Architectural Design

In this section we will discuss at a high level about the relationships between the different subsystems with The planner and/or the scheduler.
Planner - Database : The planner's tools which help the user set up a plan will use an external database (for instance , Celestial coordinate planner tool will be based on star catalog from the database). Also, rejection or approval of a new plan method will be based on data from those databases. After the plan was checked, planner will also store the plans in the system local database.
Scheduler - Database: The scheduler will be using external databases in real time in order to prioritize observations by updating their constraints and the current environmental conditions. It will also receive new plans every scheduling cycle from the local database.
Planner - Scheduler: After the approval of a plan being done by the planner, the plan goes to a missions bank which is located in the database. The scheduler uses this bank in order to choose the best one.
Scheduler - Control: The scheduler sends the chosen plan to the control for execution. While the plan runs the scheduler will wait for response from the control which reports back if the plan succeeded or failed.

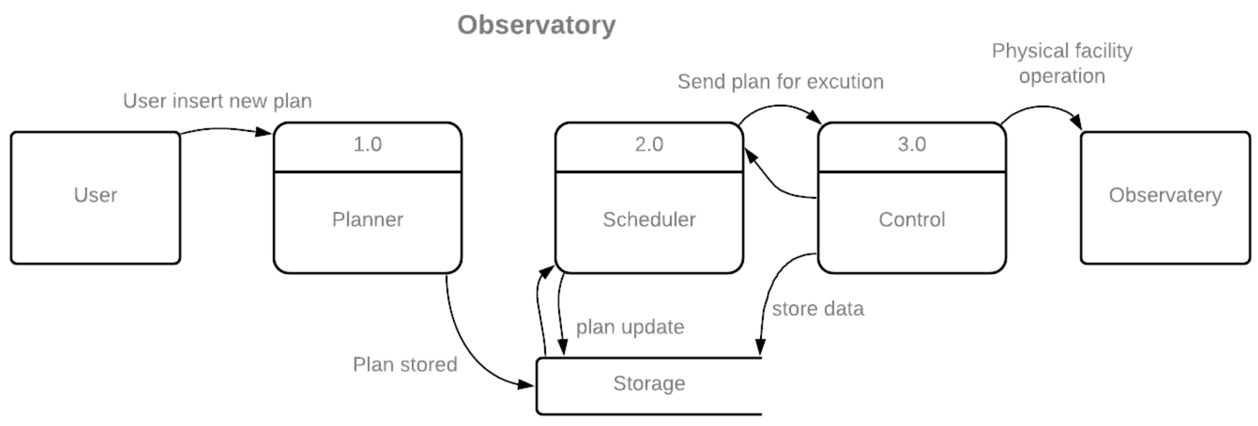
We will present a sketch describing the architecture of the system with emphasis of the Planner and Scheduler modules. This sketch does not include external interfaces.

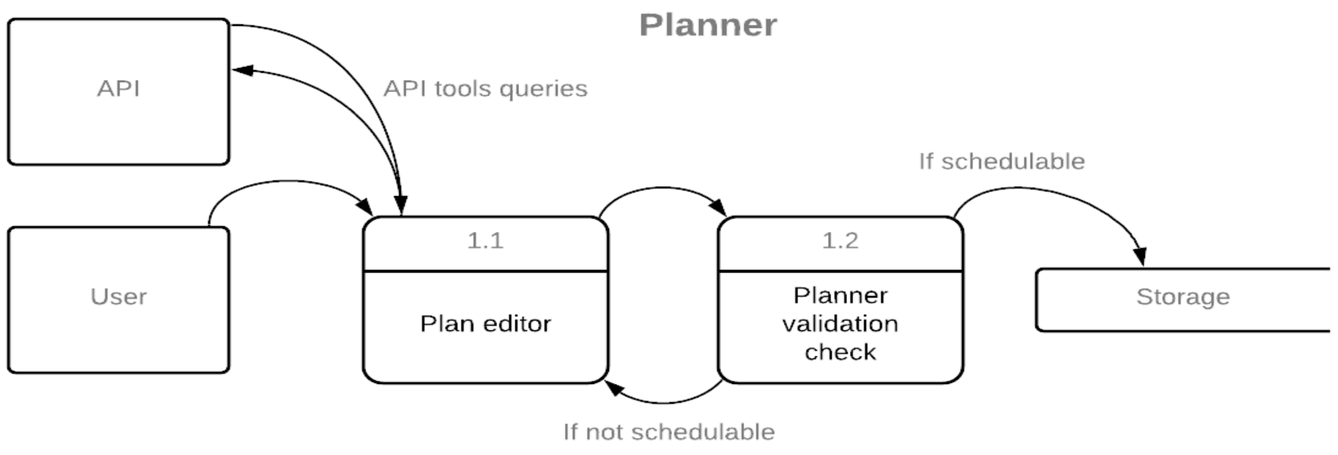


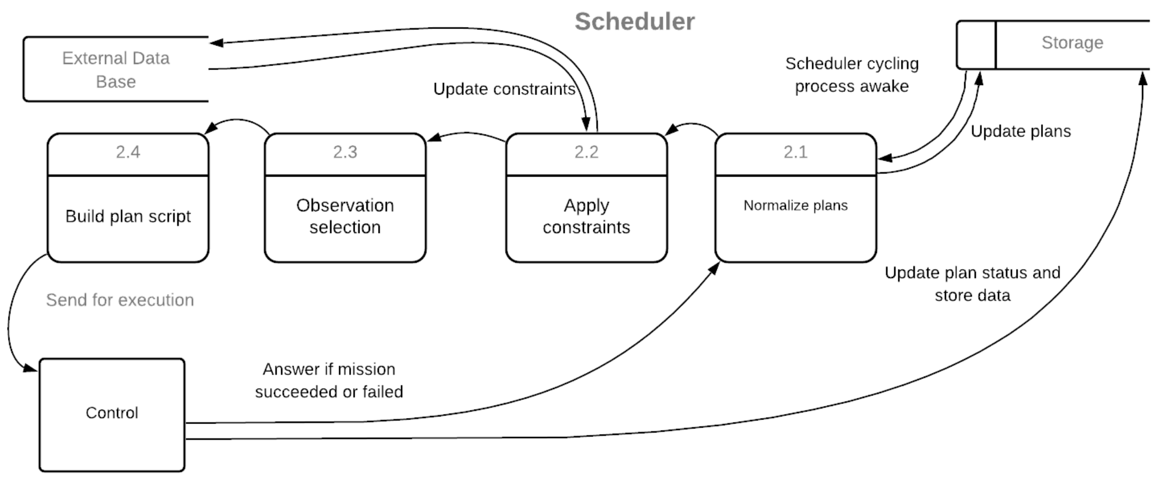
Decomposition Description

In this section we will provide a functional description of the Planner and the Scheduler modules by presenting data flow diagram and structural decomposition diagram.

Data Flow Diagrams:







Design Rationale

In order to construct this architecture we need to develop a web user interface and two web server applications for each Planner and Scheduler. We have decided to use Django for the Scheduler and NodeJS for the Planner server. We want to make each module independent and replaceable so two separate servers are needed. We may need in the future use of multi threads in the Scheduler server so we wanted to avoid the single thread limitation of JavaScript. Python is relatively user friendly and familiar with physicists, so if the scientists from the department of astrophysics would ever want to change the logic of the system , they won't necessarily be dependent on the development team. Also, we will constantly use API services that best work with Python. For the Planner side we prefer NodeJS server which is faster. That is to maximize user experience.
Because we are integrating with ACP software as the current control program,we will create a similar process. So the output of the Scheuler is a text script which contains target, coordinate, configurations and exposure setting according to ACP syntax.

Data Design

Data Description

In order to serve the astrophysics researchers efficiently the data entities need to represent the research accurately. The data structures that needed are : User , Project , Plan , Observation and ImageSet.
User - Creating an entity to the researcher that represents him/herself. Functioning as a top level node, contains a list of Project and account information.
Project - Child of User, describes a single project which is being conducted by its parent(User). Contains a list of plans due to the need for continuous observation over a couple of nights.
Plan - Child of Project, represents a set of at least one observing operation on one night. It is represented as a list of Observations and also carries several constraints in order to determine its profitability. This entity is executable.
Observation - Child of Plan, represents a single target at a single time. Users can set multiple exposures for an observation which makes a need to unite them into one group. This entity similarly to Plan is executable and has constraints. This unit is linked to other Observation entities on the same plan.
ImageSet - Child of Observation, represents at least one image that is acquired from the observation process. This entity will be stored in the database as it functions as the product of the whole process.

Besides the data structures mentioned above, we constructed data structures that represent relevant astronomical related data. For instance, celestial coordinates. Object that contain two fields which are declination in degrees and right ascension in seconds.

Data Dictionary

We have concentrated special data and entities which will take an important role in this project. You will find here the entities from the prior section along with some of the constraints. We are planning to add and update our constraint list according to the astrophysics researched instructions.

Name	Type	Description
User	User	Entity Represent an observatory user.
Project	Project	Entity At least plan, represent research.
Plan	Plan	Entity One or more observations over one night.
Observation	Observation	Entity Observation at one target at a one time.
ImageSet	ImageSet	Entity Set of images - the product of the observations.
AirMass	AirMass	Constraint Represent the air mass in the atmosphere in the target’s direction.
Angle	Double	Constraint The angle which the telescope points to.
MoonDistance	Double	Constraint
CelestialCoordinates	CelestialCoordinates	Constraint Built from right ascension and declination. This is a coordinate for a space object.

Component Design

In the section we will explain each function mentioned in the Decomposition Description individually. For that, we will provide a pseudocode which describes the basic structure of the component. We will mention once again that this is an initial direction for those methods and will be changed in a more efficient way.

Planner

1.1 Plan Editor - This process creates the observatory mission. The user inserts the fields that construct the plan object and send it to the plan verification process. We use tools to ease the process by using API services. Further explanations about this module can be found in the Human Interface Design section.

Function explanation: This process is simply a construction of a plan object. Therefore we will use the plan's constructor method and API tool like target finder etc.

1.2 Planner validation check - Function that checks whether or not the plan is schedulable or not, and if the plan is likely to be scheduled. If the plan is illogical the plan will be rejected. If the plan is legal and is not likely to be scheduled the user will be notified and be asked if the plan should be scheduled anyway.
Pseudocode:

```
   ValidationChecker(Plan plan)
1. ok = runChecks(plan)
2. if ok is false
   2.1 return false
3. else
   3.1 efficient = isEfficient(plan)
   3.2 if efficient false
     3.2.1 answer = askUserPremission()
     3.2.2 if answer true
       3.2.2.1 return acceptPlan(plan)
   3.3 else
     3.3.1 return acceptPlan(plan)
4. return false
```

This pseudocode describes the structure of this component. We have yet to define all of the needed checks to assure validation but the runChecks function checks the plan's parameters and returns boolean value which determine if the plan is legal and logical. Next the Planner checks the probability of the plan to be scheduled. If the chances are low, the user will have to approve this in order to add it to the waiting list. If it has a fair chance - the plan will be added to the waiting list which is stored in the database.

Scheduler

2.1 Normalize plans - This method will set every plan to 0.5 priority in order to update it's priority due to environment changes and new plans.
We will provide pseudocode to demonstrate it.
Pseudocode:

```
    NormalizePlans(List<Plan> plans)
1.  plans.AddNewPlanFromDB
2.  For plan in plans
    2.1 plan.SetPriority(0.5)
3.  return plans
```

2.2 Apply constraints - Update the constraint of the plan in order to revalue the priorities. The scheduler will look for any data changes from the last scheduling cycle, and if the observation can be launched - the Scheduler assigns it with a number between (0,1) which represent its priority. Also, according to the current data the Scheduler will throw every plan that cannot be scheduled either from timing clash or one of those constraints preventing it to be schedulable.
Pseudocode:

```
    ApplyConstraints(List<Plan> plans)
1.  plans.UpdateConstraints()
2.  schedulablePlans = ClearPlanWithIllegalConstraint(plans)
3.  schedulablePlansWithNoClashes = ClearClashingPlans(schedulablePlans)
4.  return schedulablePlansWithNoClashes
```

In order to revalue the current plans we need to query the database for every constraint for each plan. Then to eliminate irrelevant plans we will have to check which of whom has an allowed constraint and is not clashing with another timeline. This way, we have created a relevant Plan list which is transferred to the decision algorithm for further process.

2.3 Observation selection- This phase is the decision maker component which selects the most suitable plan among the current plan list. We will use an efficiency function which determines the priority index for each plan. The function taking the plan’s constraints into account by measuring their efficiency according to the ideal environmental conditions. This way the Scheduler estimates which plan is the most worthwhile.
Pseudocode:

```
      observationSelection (List<Plan> plans)
  1.  FOR plan IN plans
      1.2    plan.serPriority(efficiency(plan))
  2.  return maxPriority(plans)
```

In this pseudocode we show that the scheduler will iterate between the plans and set priority for each by using the efficiency function. At last, we return the highest priority plan.
The efficiency will be a sum of the measures that will be determined by the constraints.

2.4 Build plan script - This function is temporary and will be replaced in the future. The goal of this component is to make an executable script that runs on ACP. Afterwards the files would be transferred to the control for execution.
Pseudocode:

```
      ScriptBuilder (Plan plan)
  1.  Text ACP_Script =PlanToText(plan)
  2.  sendForExecution(ACP_script)
```

Human Interface Design

Overview of User Interface

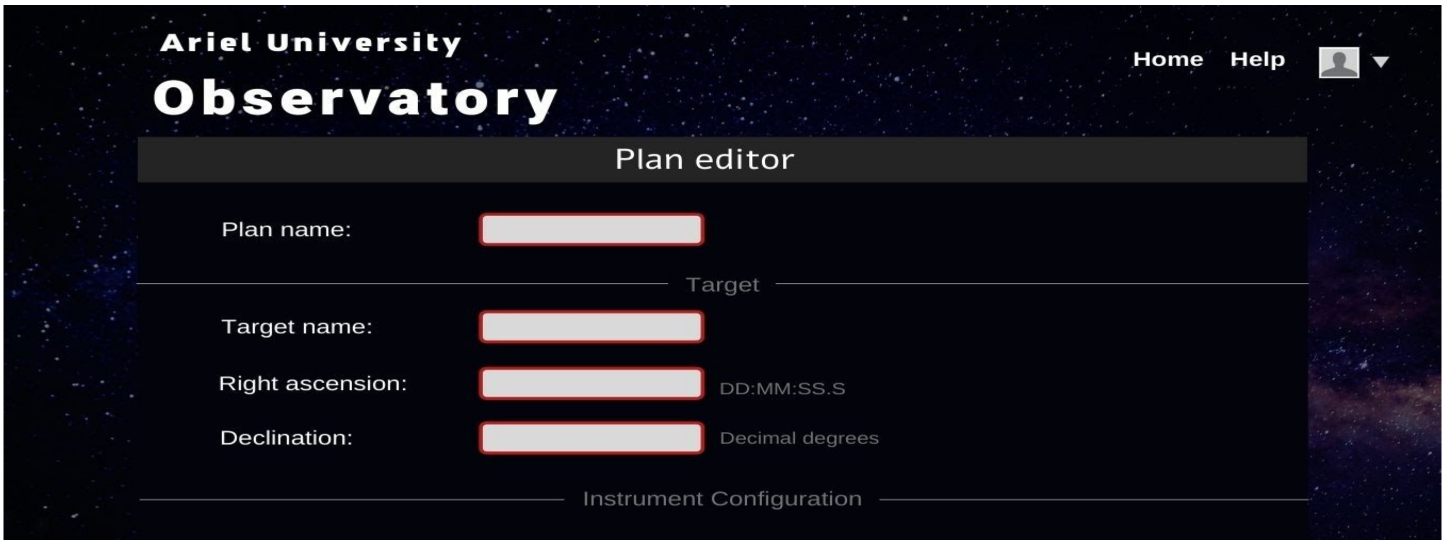
The user interface will be a web site designed to conduct new observing plans and follow the observatory's actions. The interface will include a page destined for adding a new observatory mission which we call “Plan Editor”.
The user will set the following:

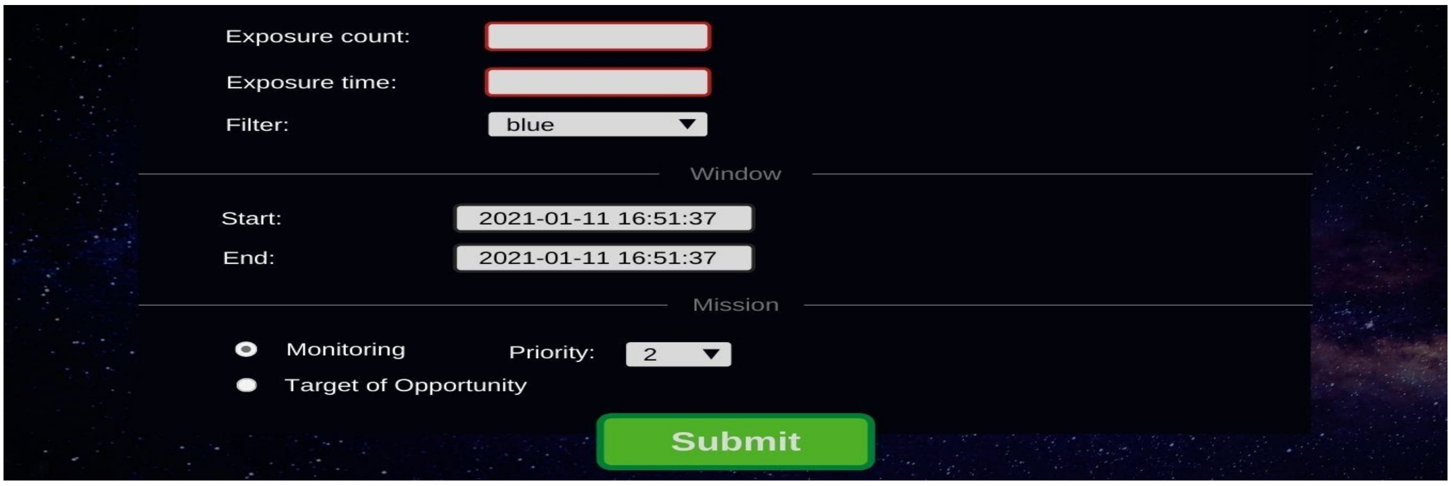
- Plan's name : Every plan will have a name in order to recognize it later on.
- Dates range : The user will have to set the starting date and the ending date of a plan so the scheduler will know when to run it.
- Target : In order to know where to calibrate the telescope's angle, the user will have to either enter the target's name or its celestial coordinates. If the user enters a name the planner will automatically complete the celestial coordinates fields.
- Configurations : For each plan the user will be able to decide how many exposures and how long one exposure will be when the observatory takes the shot. The filter of the exposure will be set in this section as well.
- Prioritization : Plan has many constraints who determine their likelihood to be scheduled. We have dedicated one constraint which is set by the user. That way the user can influence the scheduler decisions in order to run more urgent missions. Also, there is another field that determines this constraint and it is the kind of the mission. Target of opportunity are rare and therefore always prioritize over monitoring missions.

Another side of the user interface is the home page - a place where users can watch the facility doings over the night (and past nights). This page will present the mission status. For instance, if a plan has been completed , on a continuous mission failed to run or hasn't been executed yet. The user will recognize his plan by checking the name he/she gave to it and by attribution to the username.

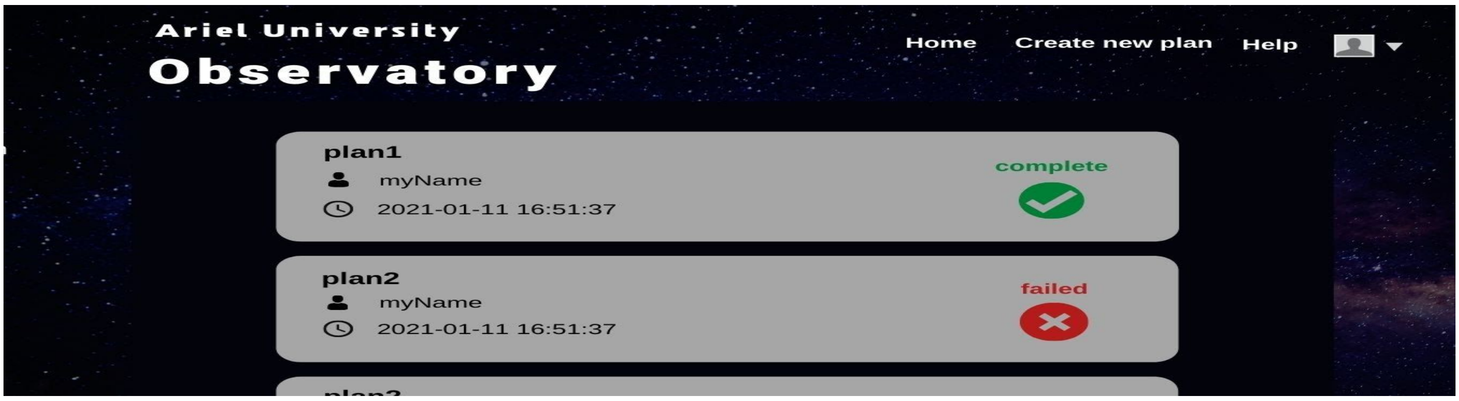
Screen Images

We have designed a sketch for the web interface which includes a plan editor and missions stock.
The next two screenshots display the Plan Editor - a space where users can add new missions to the observatory.





Stock - this page will present the observatory missions. Each tab will contain the name of the user, the name of the plan, dates and status.



Screen Objects and Actions

In this section we will discuss the functionality of the web interface, or in its other name - the Planner. This component has two main goals, the first one is to make a comfortable user experience. It is being achieved by :

- Plan form - Operating the observatory facility can be a complicated task for most people. The plan form is a way to simplify the process of creating tasks. The only thing that the user has to do is to instruct the facility when and where to observe and for how long. Visibility plays an important role in the user experience, so we will make great looking web pages with clear buttons and short and to the point texts.
- Tools - We will include tools which help the user to fill the form. One of the tools will be a method that will receive a string representing a target name, the function will return its celestial coordinates. This functionality saves the user time which improves the user experience. Tools that may be added are an exposure time suggester and popular sites recommender.

The second goal of the planner is to transfer an executable ,schedulable with high probability to collect quality data. For that the planner will do the following:

- Reject illogical plans - The planner will check whether the plan is possible considering the facility location on Earth, the hardware limitations or any other illogical reason which make the plan not executable. If the cause of this kind is detected - the scheduler will prevent submitting it.
- Inefficient plans warnings - The planner will check if the plan has a high reasonable chance to be launched successfully and collect on standard quality data. If not, the Planner will let the user know that the plan may not be scheduled due to low probability of getting picked. But if the user decides to run it anyways - the Planner sends the plan to the scheduler.

The user interface in the main input of the whole system, the plan editor page creates a Plan entity which is discussed in the data sections. This is an executable entity which we can examine and make decisions with. So the user creates a plan through the plan editor by entering the entity's fields with help of tools. The Planner analyses the plan and determines if it executable and if it is, sends them to the scheduler. The user can see the status of the plan on the home page.

Requirements Matrix

To assure that the design and concept of the Planner and Scheduler modules accomplish the system requirements that we have set. We will check every functional requirement and point to the component that solves it.
Click here to see the modules' requirements (3.1 section).

Planner

- Web Interface - on the Human Interface Design section there is an explanation of this interface including screenshots. The first five and the seventh requirements are solved via the web interface and the Plan Editor component which constructs Plan object with help of databases and other tools.
- The sixth requirement is being done by the scheduler ACP script builder component.
- The Eighth requirement is accomplished by the Planner validation check component.

Scheduler

- Decision algorithm demand is being done by the ObservationSelection component.
- Second and fourth demand is being achieved by the ApplyConstraints component by updating the constraints and revalue their priority.
- Fifth requirement is a derivative of the last two components we have mentioned. Every schedule cycle ,plans revalued .So the plan won't be launched if its environment conditions are not the best.
- The other requirements are definitions of what constraints value are optimal or what consider as a value we should avoid. We have yet to implement those functions but they will be taken into account when we begin the implementation phase.