Operation Dark Sky

Software Design Doc

September 30th, 2022

Team: Justin Ceccarelli, Jordan Tatum, and Luke Thompson

Sponsors: Jim Clark, Peter Kurtz, and Henrique Schmitt

Mentor: Daniel Kramer

Capstone Director: Michael E Leverington

Version# 1.2

Table of Contents:

1 - Introduction:

The Navy Precision Optical Interferometer (NPOI) is an astronomical observatory operated through a partnership between the Lowell Observatory and the U.S. Naval Research Laboratory (NRL). The NPOI conducts research on unique astronomical phenomena like binary stars as well as carrying out a range of practical functions that support the U.S. Navy's navigational and communication capabilities, such as monitoring the satellites used for GPS navigation. The Navy still relies on star charts provided by the NPOI to serve as a form of navigation that is entirely independent of technology.

The NPOI is primarily dedicated to astronomical research with the aim of improving our understanding of the stars and exoplanets in our galaxy. The NPOI is an interferometer, which means that it uses a network of mirrors to collect and redirect starlight from up to six locations before merging the light together to create an interference pattern. Interference patterns capture how the light from two or more sources interfere with each other, and is especially useful when observing binary stars. The NPOI is the world's largest baseline interferometer, making it especially well suited to studying binary stars, or pairs of stars that orbit each other. Like many observatories, the NPOI can only operate at night, during fair weather, and beneath a cloudless sky. Since these circumstances are out of the observatory's control, they have to take advantage of every opportunity to collect data. This means planning each night of observation well in advance to make sure that the equipment is positioned and then continuously re-positioned to capture the appropriate stars at precise times over the course of an evening. Properly planning an event of this scale poses a significant challenge, especially considering the constant movement of the stars in the sky relative to the earth. The astronomy team at the NPOI initially needed to manually perform an exhausting amount of technical work to determine the optimal timing and ordering of their observations to ensure they could capture the necessary data. This planning process requires an experienced professional to dedicate a significant amount of time every night the NPOI observes the stars.

In order to handle this organizational challenge, the NPOI created a software application they called "obsprep", short for observation preparation. Obsprep consisted of two complementary components: the backend and the frontend. The backend contains a catalog of stars, including their location in the sky, as well as a library of astronomical functions called the Naval Observatory Vector Astrometry Subroutines (NOVAS). The frontend of obsprep was a Graphical User Interface (GUI) that allowed a user to access the data in the star catalog and to use the functions in NOVAS to perform calculations on that data. Together, the two parts of obsprep allowed astronomers to create a list of the stars they wish to observe before calculating the optimal time and order to observe those stars to maximize the observatory's uptime.

When obsprep was originally written in the late 90s, it met the NPOI's needs for planning and organizing the observatory's nightly work. Unfortunately, in the decades since then the software's frontend has become outdated to the point where it requires frequent maintenance simply to avoid crashes and unplanned downtime. The NPOI has estimated that a night of observations costs the observatory approximately $12,000. Hence, they have decided to retire the original implementation, and have hired team Dark Sky to create a new frontend application for the obsprep software. The NPOI envisions a solution that can be easily installed on a user's work machine regardless of operating system, that allows the user to select multiple desired stars for a specific night, as well as generate and save graphs of the stars' progress through the sky to better visualize their location as time passes.

Our solution must meet certain requirements set by our client. First, our solution must feature an intuitive user interface that our client can use with minimal training. Second, it needs to present astronomy data to the user in a specific graphical format. Finally, our solution needs to address several issues that were present in the previous iteration of obsprep, primarily issues of installation and maintenance. In terms of performance, our client expects our solution to be fast, retrieving and processing data in a matter of seconds. It should require only a single, brief training session for an employee to become familiar with the program, and tooltips and help information should be readily available and easy to access.

In this document we will describe our project's overall architectural design, as well as the individual design of each major module. We will outline and describe the modules and interface elements essential to our project, as well as the relationships between them. Finally we will describe a design-centric implementation plan that provides a schedule for our project.

2 - Implementation Overview:

Team Dark Sky's objective is to build a new GUI for the NPOI's pre-existing obsprep software. The new interface is being written from the ground up with new code, but will still connect to the same NOVAS backend that the original software used. This frontend program will carry out several key functions. It will:

- Collect user input into a simple, compact GUI based on the clients specifications
- Access a catalog of star data and astrometry functions
- Process data according the the user input
- Provide output in a specific format, including text and graphical options

This project is entirely software, and will not include any hardware components. It will be installed on the client's workstations, which are running either Linux or MacOS based systems. Once the software has been created, our client expects to be able to update and maintain the software on their own, without input from the design team. This means that documentation and good coding practices are critical to the long term success of this project.

This project will be written primarily in the Python Language, as it is familiar to our design team and compatible with the NOVAS software backend[1]. The team will be building the GUI using Tkinter[2], a graphical python framework available on all major platforms. The tkinter GUI will call functions from several python libraries stored in the backend, primarily Compat.py and Astro.py.

3 - Architectural Overview:

While designing the architecture of our software, we tried to establish longevity and ease of use as our top priorities. The overall scope of our project is fairly narrow, and we wanted to reflect that in our architecture. The software consists of four major components: First the GUI, which allows the user to input data and specify the type and format of the output. Second, the ephemeris file, which serves as a catalog containing all the relevant positional star data that the program will need. Third, the NOVAS backend, which includes several python libraries containing all of the astronomical functions that we will need to process data. Finally, the core obsprep file, which collects user input from the GUI and star data from the ephemeris, then processes those into the requested output using the functions provided by NOVAS.

[1] https://pypi.org/project/novas/
[2] https://docs.python.org/3/library/tkinter.html

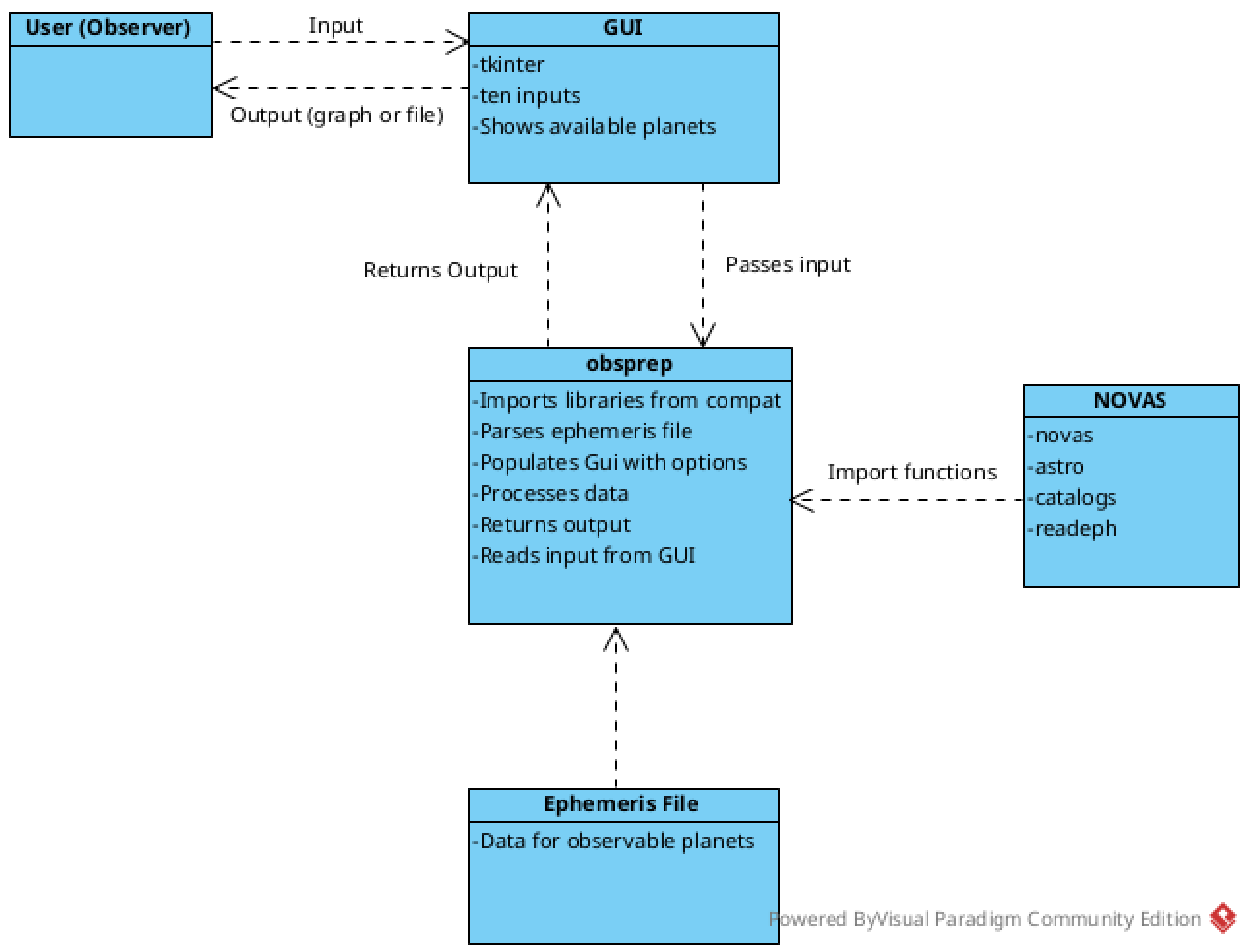


Figure 3.1 - Architecture Diagram

NOVAS is a large and complex assortment of files that serve a variety of purposes, but we are primarily concerned with complex astronomical functions contained within novas.py, astro.py, and catalogs.py. These libraries hold functions that can be used to calculate advanced astrometric values and calibrate equipment. They will be imported into the obsprep file so that their functions can be called as needed to process data. These include calculating the apparent position of a star, the local time when it will rise and set, and other functions essential to the operation of obsprep.

Obsprep serves as a central, coordinating element of the system. It accepts input from the GUI to determine what information the user needs and what format the output should have. It then parses input from the ephemeris file for the star data necessary to fulfill the user request. Finally it calls functions from NOVAS to calculate the necessary parameters and create the output.

While designing our architecture we drew inspiration from the model-view-controller pattern. We believe that this is a strong choice for building something that is easy for our users to learn and maintain over the long term. The view portion of our architecture is our GUI. The controller is the obsprep python file that calls the different backend functions to be able to produce the calibration. As for the model we don't have one single model we manipulate because we have to create a unique configuration based on the specified input.

4 - Module and Interface Descriptions:

4.1 - The GUI

The GUI component will be responsible for accepting user input and displaying a processed output to the user. The interface has ten mandatory user input values that it requires in order to function. These values are then used to calculate a calibration for observing the star at the provided time. These inputs are:

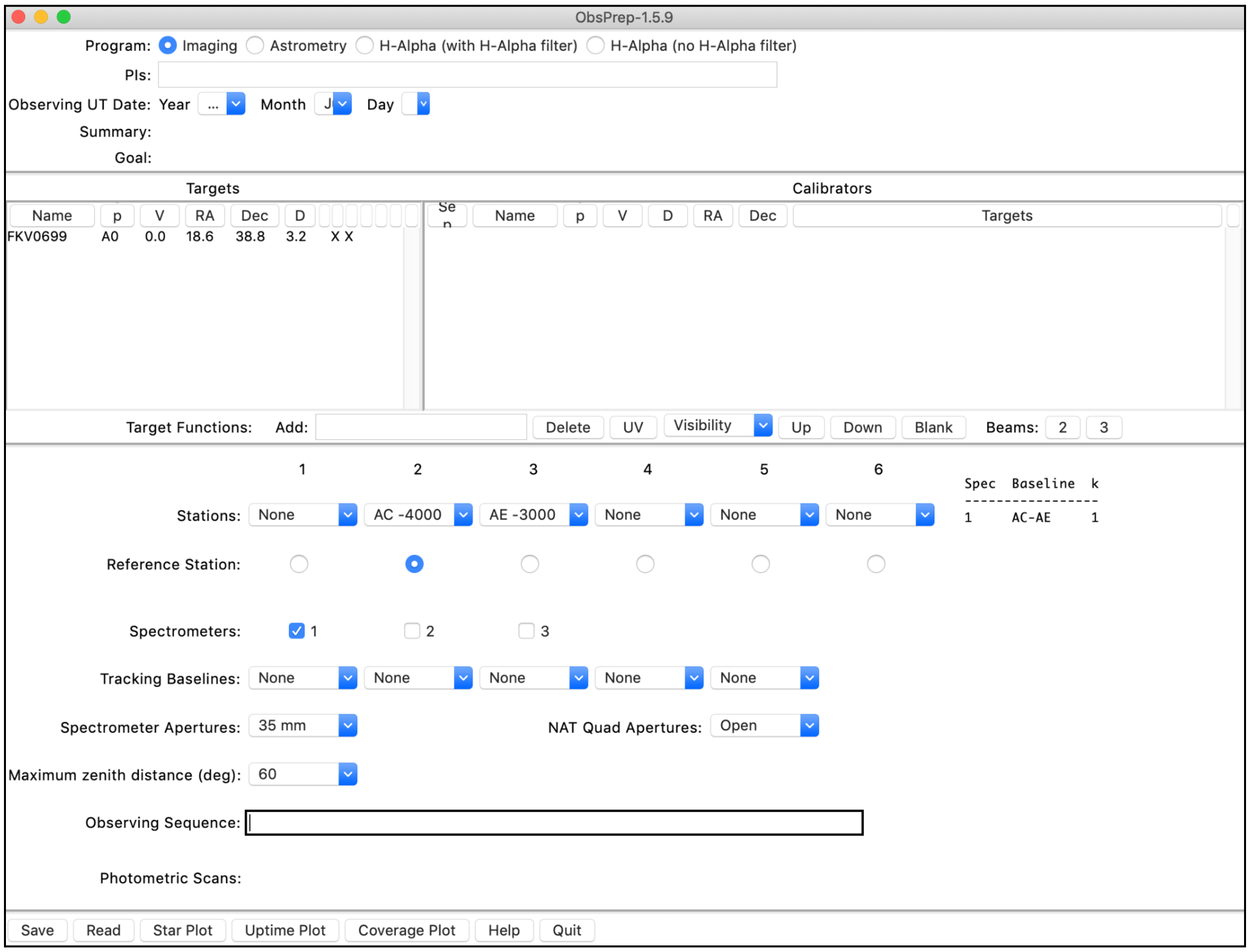
- The precise time and date at which the star will be observed.
- Which program to run, choosing between imaging, astrometry, h-α with filter and h-α without the filter.
- Which stars they need to observe.
- Which stations they want to include in the observation. The user must choose at least two stations out of the six available, which will be provided within a drop-down menu.
- A reference station for their observation.
- At least one spectrometer, although they have the option to choose up to three at a time.
- The tracking baseline, which has four options but can be left blank.
- A spectrometer aperture
- A NAT quad aperture.
- Max zenith distance.

There is an array of buttons at the bottom of the GUI that allow the chooser to perform several additional actions:

- Read a file that provides all the required input.
- Save their current input as a new readable file.
- Read in a binary file of star information which will then populate the star table so that the user can choose a star.

There are three additional buttons, each corresponding to a particular graph. The user chooses from star plot, uptime plot, and coverage plot. There's also a help button that the user can click to access documentation and general instructions on how to use the interface. Finally we have a quit button that closes the program.

As described earlier in this paper, our client has previously built and used a form of obsprep that has become unusable over the past few decades due to aging hardware and reliance on aging, un-maintained software. Because of their experience using the original software tool, our client has a clear understanding of how the GUI should work, and how they would like it to look. They provided our team with this screenshot of the original interface:



ObsPrep-1.5.9

Program: [✓ Imaging] Astrometry H-Alpha (with H-Alpha filter) H-Alpha (no H-Alpha filter)

PIs:

Observing UT Date: Year ... Month J Day

Summary:

Goal:

Targets

Name	p	V	RA	Dec	D	
FKV0699	A0	0.0	18.6	38.8	3.2	X X

Calibrators

Se n	Name	p	V	D	RA	Dec	Targets

Target Functions: Add: Delete UV Visibility Up Down Blank Beams: 2 3

	1	2	3	4	5	6
Stations:	None	AC -4000	AE -3000	None	None	None
Reference Station:		✓				
Spectrometers:	[✓ 1]	2	3			
Tracking Baselines:	None	None	None	None	None	

```
Spec  Baseline  k
-----------------
1     AC-AE     1
```

Spectrometer Apertures: 35 mm

NAT Quad Apertures: Open

Maximum zenith distance (deg): 60

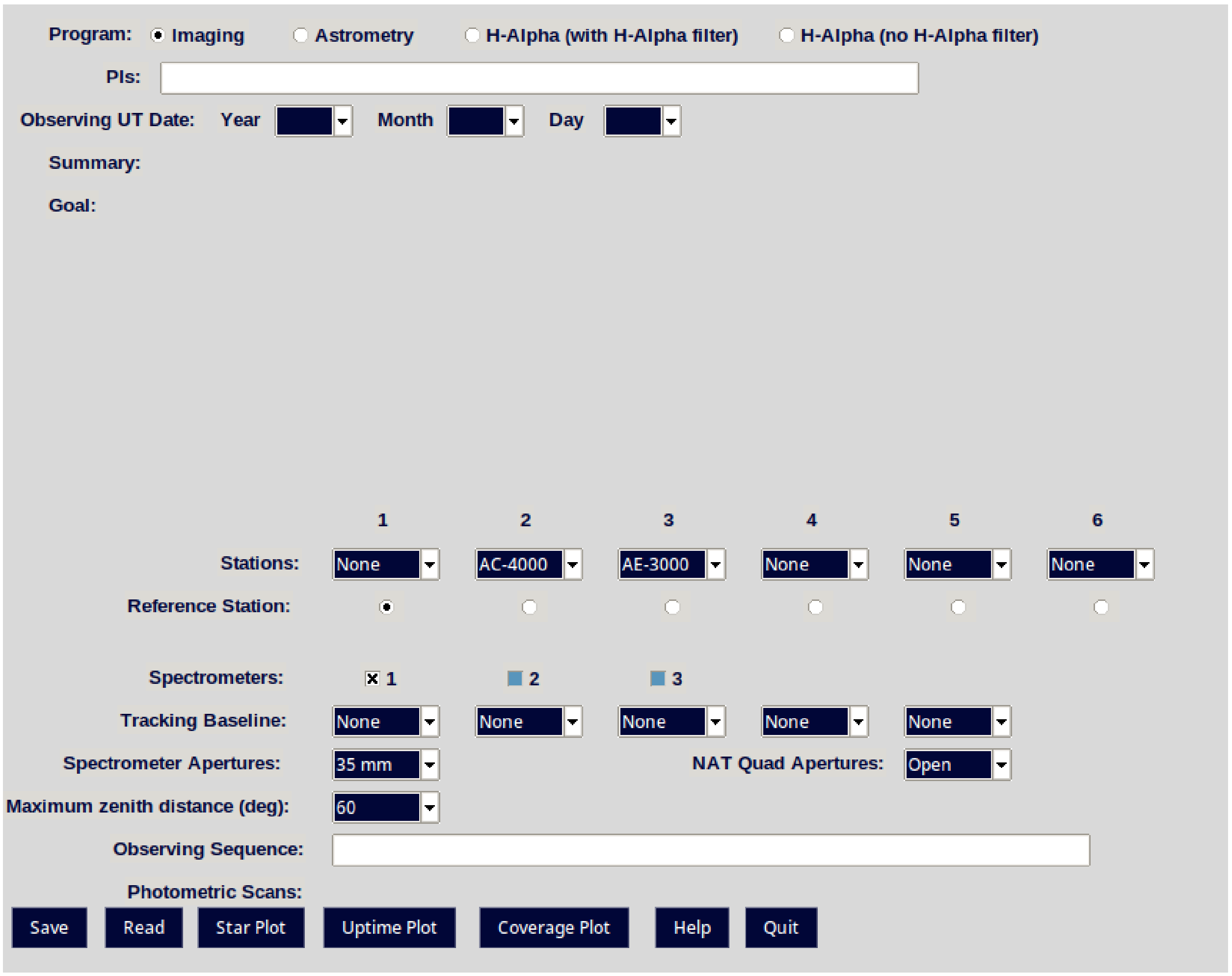
Observing Sequence:

Photometric Scans:

Save Read Star Plot Uptime Plot Coverage Plot Help Quit

Figure 4.1 - Original obsprep GUI

Due to their familiarity with this design and confidence in its ease of use, the client requested that our project imitate the original layout as closely as possible. We have already created a prototype interface using tkinter:



Program: [✓ Imaging] Astrometry H-Alpha (with H-Alpha filter) H-Alpha (no H-Alpha filter)

PIs:

Observing UT Date: Year Month Day

Summary:

Goal:

	1	2	3	4	5	6
Stations:	None	AC-4000	AE-3000	None	None	None
Reference Station:	●	○	○	○	○	○
Spectrometers:	☒ 1	■ 2	■ 3			
Tracking Baseline:	None	None	None	None	None	

Spectrometer Apertures: 35 mm NAT Quad Apertures: Open

Maximum zenith distance (deg): 60

Observing Sequence:

Photometric Scans:

Figure 4.2 - New prototype GUI

This prototype currently does not include the table of target stars, as that is one of our current coding challenges. Once that has been implemented, the GUI will see additional aesthetic changes to elements such as color and button shape in an effort to make the GUI as appealing and intuitive to use as possible. We have been, and will continue to be in constant communication with our client to ensure that our software meets their particular technical and design requirements. This includes providing a functional prototype so that the users can test the software for themselves and provide feedback.

4.2 - Setup.py and Install instructions

Contains the scripts and instructions for setting up obsprep and NOVAS according to previous versions specifications. This part of the software is likely to change radically over the course of this project. This module is executed with a series of terminal commands that establish the architecture that obsprep and NOVAS require in order to interact with each other in their current state.

- Intermediate step in establishing OBSPREPDIR environment variable
- Establishes an assumed file architecture for obsprep (where to find catalog and ephemeris files)
- Establishes obsprep version

This module is intended to rectify a major issue our client had with the original obsprep software. The original software required extensive use of the command line to install and operate, which was challenging and confusing for basic users. The version number was assigned rather than programmatically determined, which could potentially cause errors. Furthermore the use of environment variables to determine the architecture of a program is a clunky and complicated procedure that is typically considered a bad programming practice.

Our client wants our project to implement an installation method that does not require extensive command line interactions so that it can be installed and operated by users without a programming background. Our solution will accomplish this by requiring just a single command line operation. The project will also include simple yet comprehensive instructions, including graphical directions that ensure a user can set up the program without issue or frustration. These instructions will be contained within a word processor file that can be easily shared between users.

4.3 - Ephemeris Binary File

The ephemeris file is a binary file containing astrometrical data about the position and rotation of the earth that must be known in order to calculate the relative position of distant stars. The file is generated by the NPOI, and needs to be parsed by our program. Fortunately, there are several utility functions within NOVAS, including `run_eph.py`, that provide tools for reading and parsing the ephemeris file. While some observatories must update their ephemeris file regularly to account for the movement of the bodies they observe, the stars that the NPOI studies are so distant that any change is functionally irrelevant to our calculations. As such, the ephemeris file only needs to be updated once every few decades, and that is managed entirely by our client. Our software will still be designed to handle any provided ephemeris file, so changing this should not cause issues with the operation of our software.

4.4 - NovasC + Novaspy

This is the functional core of NOVAS. It contains the actual functions that Novaspy provides to the interface. These include:

- AppStar and AppPlanet
- TopoStar and TopoPlanet
- VirtualStar and VirtualPlanet
- LocalStar and LocalPlanet
- AstroStar and AstroPLanet
- MeanStar
- TransformHip and TransformCat
- SetEphemeris
- Supporting methods for the above

NovasC is responsible for the actual computations involved in evaluating target and calibrator information while Novaspy provides access to these functions for python programs. With the information provided in catalogs.py and the ephemeris files these objects and functions can be used to determine when a star will be visible and its position in the sky at that time to help astronomers plan their observations. These files are already accessible by all of our clients' workstations, and do not need to be installed or set up in any way.

4.5 - catalog.py

The catalog.py module maintains a library of stars through two methods. The first is by parsing a pickled file currently simply called "catalog" to access an already acquired set of stars. The second is by querying the SIMBAD database for new information and creating cat objects out of that information. These cat objects can then be stored in a new pickle file for future use. This module also contains the functionality to acquire known diameters of stars, and determine their suitability as calibrators. This includes:

- getKnownDiameters - Returns a dictionary of stars using the number of the star as the key. Dictionary also includes suitability as a calibrator for that star
- simbadQuery - Returns a dictionary containing the SIMBAD information and input information about the star
- Star Object class - Class containing the many fields needed to describe a star
- StarCatalog object class - a class for more easily managing the star objects

- VizierStarTable and VizierDictTable classes for managing Vizier queries

4.6 - astro.py

The astro.py file is the main file that does astronomical calculations for our program to pass to the novas.py file. Our clients gave us the file and the only changes we had to do to it were to port it from python 2.4 to python 3.10. The first major responsibility of the astro.py file is to call the setEphemeris function and have the ephemeris set to the JPL (Nasa's Jet Propulsion Laboratory) ephemeris file location. This allows the astronomers to have access to the file that lets us know earth's location relative to the object they're trying to observe. Then we set the npoi site location by passing in the given coordinates to the site_location class. From there the file performs different astronomical functions based on the location we created for the site. These functions include:

- JulianDate2ModifiedJulianDate and ModifiedJuliandate2JulianDate
- EquationOfEquinox
- GreenwichApparentSiderealTime and GreenwichMeanSidereelTime
- ObservedPlanet and ObservedStar
- ApparentStar
- Airmass2ZenithDistance
- HourAngle
- RiseSet
- Twilight
- ZenithDistance

4.7 - Obsprep

The central element of the project is the obsprep file, which essentially exists to coordinate the data and inputs from the other elements in order to generate the necessary output. Obsprep acts like an API that will be calling the backend files and returning the output to the interface. It will accept input from the GUI, parse data from the ephemeris file, and call the libraries within NOVAS for the astrometric functions required to create the desired output. The GUI will send the user input information as an array to the obsprep file. This array will then be used to get the necessary astrometric calculations to return the calibration for the observers. Obsprep will parse data from the ephemeris and catalog files based on the star(s) chosen by user input. Once obsprep has all the necessary data, it will call functions from astro and compat as necessary to process it until it has the appropriate output. That output will be returned to the GUI.

<u>4.8 - UML Class Diagram</u>

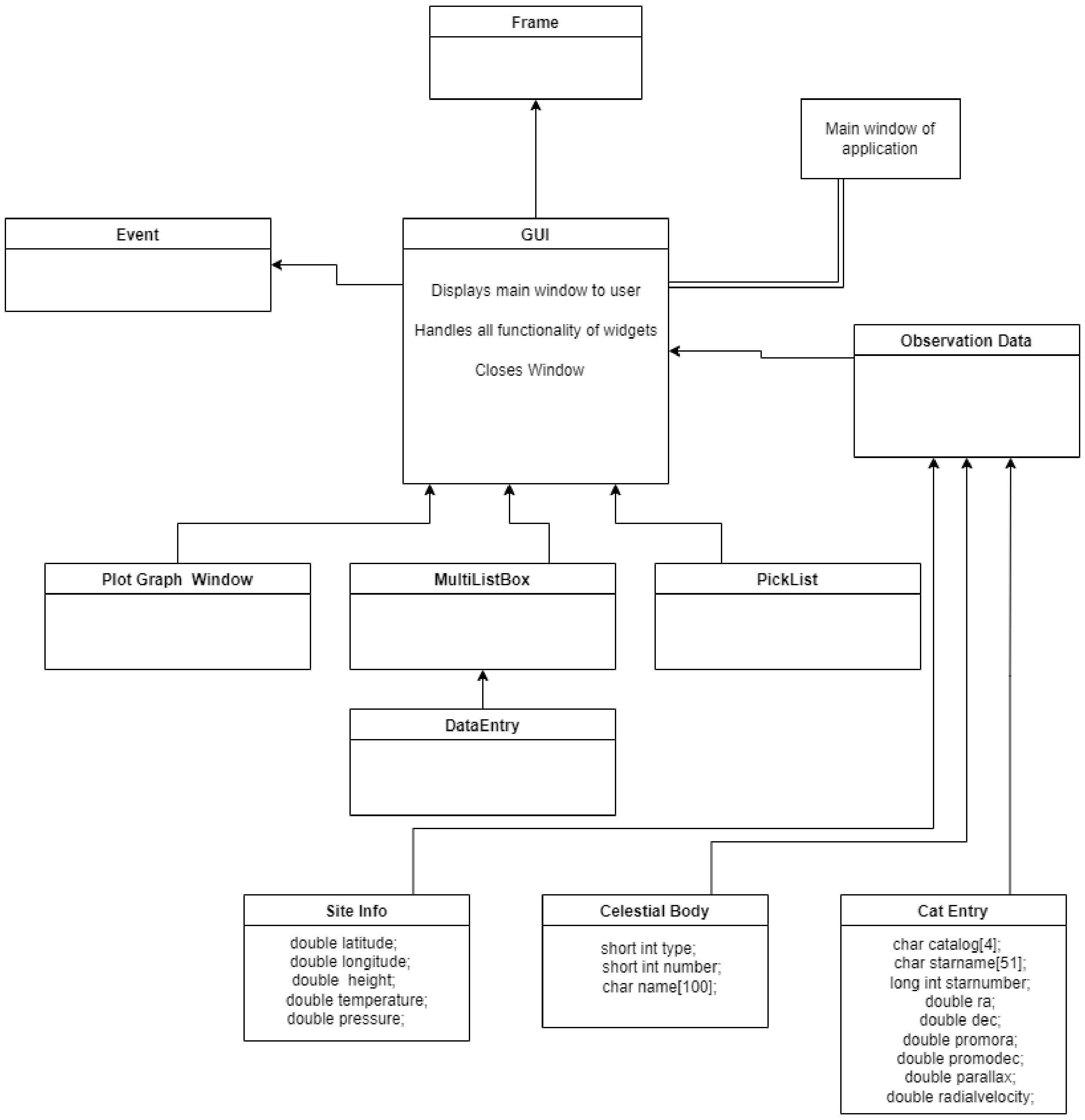


Figure 4.8 - UML Class Diagram

5 - Implementation Plan:

In this section we will outline our plans for implementing this software design. Our plan revolves around building functionalities so that their capabilities can be supported and tested. Before diving into design we need to define two terms: Technical input and astronomical input. Technical input and output are the files, both text and binary, that the program will use to generate the data on the screen and then later output into similar files. Astronomical input and output are principally determined by the user or derived from the technical inputs. The key difference is that technical outputs can be managed without the need for user input, while astronomical outputs are based on selections made by the user. (GUI input + technical input = astronomical output)

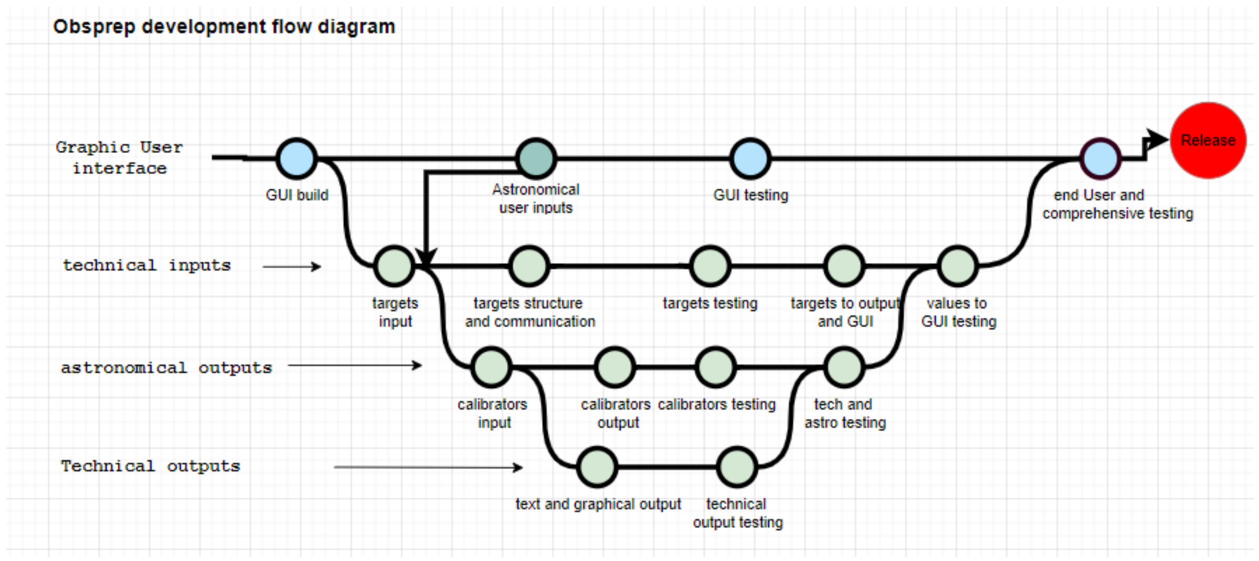


Figure 5.1 - Development Flow Diagram

Technical inputs and GUI input are therefore a prerequisite to the wider functionality of obsprep, and are the first item on our implementation plan. The top row of our plan involves the creation of the frame of the GUI (buttons, menus, window), then creating its ability to accept input from the user where needed. Then we can begin to process technical input. This comprises the first two nodes of the top row and the first node of the second row. At this stage the target's input can be read into memory, and a visual representation of the program is displayed.

From here, three parallel lines of development open up. The topmost row involves GUI testing, making sure all the intended widgets provide the expected functionality and associate with their correct functions. No data is being manipulated by the GUI at this stage, just ensuring that the window functions in an intuitive and expected way.

The second line represents the implementation of technical inputs into a usable form by other modules (mostly the GUI) and testing these inputs for correctness and file formatting. As well as ensuring that file reading errors are avoided.

The third row is calibrator testing. The bulk of calibrators will be read into memory and stored such that it can be manipulated based on GUI input.

From here the development plan is mostly integration. Bringing the data derived from the GUI inputs and technical inputs back up into the GUI so the user can see them, and then displaying those outputs as either a text file or a graph, depending on user input. The last step is user testing, which will involve shipping a version of the software to our sponsor so their users can attempt to install and use the software as they would a final product. At this stage we will still be able to make functional changes to suit our clients needs and aid in their ability to use, understand, and maintain the software. We can also make stylistic changes to the appearance and feel of the GUI to make the end users more comfortable with its operation. This is also a final check to ensure we are providing our client with everything they need.

6 - Conclusion

The NPOI is currently operating the world’s largest baseline interferometer and using the information it generates to inform and improve our understanding of unique astronomical phenomena. In particular, their equipment allows them to observe binary stars with a precision far beyond that of traditional telescopes. In addition, their work contributes to the Navy’s management of essential functions like the GPS system and navigational star charts. The observatory is expensive to operate, and requires very specific weather and lighting conditions to function, so the NPOI needs to make the most of every opportunity they have.

The software solution outlined in this document is designed to enable the NPOI to make the most of their valuable time by simplifying the organizational challenge of planning out a night of observations. It is intended to be simple to understand, and equally simple to update and maintain. Based on the model-view-controller architecture pattern, our solution is intended to intuitively access the large amount of complex data and astronomical functions necessary for the observatory to operate, while remaining easy to install, operate, and maintain. Our client has specific requirements for the GUI, since the observers are accustomed to a certain layout provided by a previous iteration of this program, and our architecture makes it easy to customize the GUI to accommodate their needs without changing the logic or function of the underlying program.

While we are happy with our design and confident that it will be sufficient to create our final product, it is still a work in progress. What we have so far is a prototype of the GUI and an initial version of obsprep with basic functionality. We have access to all the files on the NOVAS backend, which we are studying and in some cases modifying to work with our intended architecture. As we implement more of the modules and work on our solution's functionality we will remain open to changing and modifying our designs. We strive to keep our client actively involved in the design process, particularly in regards to the GUI. We will be providing them with multiple prototypes so that their users will have the chance to provide feedback.