Großer Beleg

ILR-RFS PP 20-07

Erweiterung der Bodenstationssoftware um Elemente für die Missionsplanung und Visualisierung der Lageparameter des Nanosatelliten SOMP2

Hans Nicolas Blättermann

Betreuer:

Dipl.-Ing. Yves Bärtling, ILR, TU Dresden

Dipl.-Ing. Georg Langer, ILR, TU Dresden

Juni 2021

TECHNISCHE UNIVERSITÄT DRESDEN

Fakultät Maschinenwesen Institut für Luft- und Raumfahrttechnik
Professur für Raumfahrtsysteme

Aufgabenstellung für Projektarbeit im Forschungspraktikum

ILR-RFS PP 20-07

Studiengang:	Maschinenwesen
Studienrichtung:	Luft- und Raumfahrttechnik
Name des Studierenden:	**Hans Nicolas Blättermann**
Matrikelnummer:	4514066

Thema: Erweiterung der Bodenstationssoftware um Elemente für die Missionsplanung und Visualisierung der Lageparameter des Nanosatelliten SOMP2

Subject: Enhancement of the Ground Segment Software for Mission Planning and Visualisation of the Attitude Parameters of the Nanosatellite SOMP2

Motivation:
Für den Betrieb des Nanosatelliten SOMP2 ist die Kommunikation zum Satelliten vom Boden aus von zentraler Bedeutung. Der Satellit sendet Informationen über den Zustand der verschiedenen Subsysteme sowie Daten der installierten Nutzlasten. Für den Erfolg der Mission ist dabei das Lagebestimmungs- und -regelungssystem von großer Bedeutung, da dies für die optimale Ausrichtung der Nutzlastexperimente sorgt und Beurteilung gewonnener Experimentdaten vereinfacht. Der Satellit sendet dabei Informationen bezüglich der von ihm bestimmten Lage sowohl in festen Intervallen (Baken) als auch auf Anfrage vom Boden aus (Telemetriedaten). Für die Auswertung der empfangenen Daten als auch die Optimierung des Lagebestimmungs- und regelungssystem ist die Visualisierung der empfangenen Daten in intuitiver Form notwendig.
Ziel dieser Arbeit ist die Erweiterung der bestehenden Bodenstationssoftware um ein Modul zur Aufbereitung und Darstellung von Daten des Lagebestimmungs- und regelungssystems, wobei es möglich sein soll, die Daten von verschiedenen Schnittstellen einzulesen. Das Modul soll zudem auf Grundlage aktueller Satellitenbahnelemente die Missions- und Experimentplanung durch z.B. Prognose kommender Überflugzeiten oder Übergänge in die Schattenphase vereinfachen.

Aufgaben:
- Ableiten der konkreten Anforderungen an die Software
- Schaffung von Datenschnittstellen für den Import der Daten des Lagebestimmungssystems aus gespeicherten Telemetrie- und Echtzeitdaten
- Optisch ansprechende Darstellung der vom Satelliten empfangenen Lagedaten in Bezug zu seiner Position unter Nutzung aktueller Satellitenbahnelemente bzw. der vom Satelliten berechneten Positionen
- Darstellung auswählbarer Parameter des ADCS in Diagrammform
- Entwicklung eines Moduls zur Vereinfachung der Missionsplanung sowie der zeitlichen Planung der Experimente
- Nachweis der Funktionsfähigkeit der entwickelten Software
- Ausführliche Dokumentation der Arbeit.

Rechtliche Bestimmungen:

Betreuer:	Dipl.-Ing. Yves Bärtling, ILR, TU Dresden Dipl.-Ing. Georg Langer, ILR, TU Dresden
Umfang:	390 Stunden
Ausgabe:	31.08.2020
Abgabe:	28.02.2021

Empfangsbestätigung des Studenten:
Ich bestätige hiermit, dass ich die Aufgabenstellung sowie die rechtlichen Bestimmungen und die Studien- und Prüfungsordnung gelesen und verstanden habe.

Prof. Dr. Martin Tajmar
Verantwortlicher Hochschullehrer

Unterschrift des Studierenden

Selbstständigkeitserklärung

Hiermit erkläre ich, dass ich die von mir dem Institut für Luft- und Raumfahrttechnik der Fakultät Maschinenwesen eingereichte Projektarbeit zum Thema "Erweiterung einer Bodenstations-software um Elemente für die Missionsplanung und Visualisierung der Lageparameter des Nanosatelliten SOMP2" selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie Zitate kenntlich gemacht habe.

Name:	Blättermann
Vorname:	Hans
Matrikel-Nummer:	4514066

Dresden, 15.06.2021 . . . . . .

Hans Blättermann

Abstract

For the operation of the nanosatellite SOMP 2b, and the optimal execution of its experiments, an extension of the ground station software is required to visualise its position and orientation based on telemetry data. To provide planning capabilities for some specialised experiments, it is further necessary to predict the satellites entry and exit into the earth shadow as well as its overflight times.
Since SOMP 2b registered at the SatNOGS network, its transmissions are globally acquired, and it was possible to access them using a proprietary REST API client. After receiving the transmitted information through the AX.25 protocol and the SOMP beacons, all their contained data is visualised using graphs and 3D rendering techniques. Through NASA NAIF's SPICE toolkit and the SGP 4 algorithms, models for the earth and the satellite's orbit were successfully integrated and verified. Subsequently, they were extended to predict eclipse and overflight events.
A software architecture driven development approach was conducted to maximise extensible and modifiable for a future-oriented design. Thereby the Model, View, Controller design pattern and Concurrency were used to create the application from the ground up.

Kurzfassung

Für den Betrieb des Nanosatelliten SOMP 2b und die optimale Durchführung seiner Experimente ist eine Erweiterung der Bodenstationssoftware erforderlich, welche seine Position und Orientierung auf der Basis von Telemetriedaten visualisiert. Um Planungsmöglichkeiten für einige spezialisierte Experimente zu schaffen, ist es außerdem notwendig, den Ein- und Austritt des Satelliten in den Erdschatten sowie seine Überflugzeiten vorherzusagen.
Da SOMP 2b im SatNOGS-Netzwerk registriert ist und dadurch seine Übertragungen global erfasst werden, ist es möglich auf diese durch einen proprietären REST API-Client zuzugreifen. Nach dem Entschlüsseln der übertragenen Informationen aus AX.25-Protokoll und den SOMP-Baken werden alle darin enthaltenen Daten mithilfe von Diagrammen und durch 3D-Rendering visualisiert. Durch NASA NAIFs SPICE Toolkit und den SGP 4 Algorithmus wurden Modelle für die Erde und die Satellitenbahn erfolgreich integriert und verifiziert. Anschließend wurden sie zur Vorhersage von Schatten- und Überflugevents erweitert.
Um die Erweiterbarkeit und Modifizierbarkeit des Software Designs zu maximieren, wurde ein von Softwarearchitektur getriebener Entwicklungsansatz verfolgt. Dabei wurden das Model, View, Controller Design Pattern und Multithreading verwendet, um die Anwendung von Grund auf zu entwickeln.

Table of Contents

Appendix

List of Figures

List of Tables

Index of Symbols

Symbol	Unit	Description
t	s	Time
$\vec{r}$	m	Position
v	$\frac{m}{s}$	Velocity
γ	°	Rotation around x-axis
β	°	Rotation around y-axis
Ψ	°	Rotation around z-axis
μ	$\frac{m^3}{s^2}$	Central body constant
RA	°	Right ascension
Dec	°	Declination
r	m	Radius
h	m	Height above earth surface
val_{out}		Generic output value of the SOMP 2b beacon
val_{packed}		Generic value packed inside a SOMP 2b beacon
$val_{divider}$		Divider used by the SOMP 2b beacon
val_{offset}		Offset used by the SOMP 2b beacon
$\alpha_{observer}$	°	Elevation of spacecraft above observer
$\alpha_{observer;min}$	°	Minimum elevation of spacecraft above observer required for pass

List of Indices

Definition of Indices

$\vec{r}_{j,k}$	Position vector $\vec{r}$ of body j relative to frame origin k
$\vec{r}_{i,j,k}$	Position vector $\vec{r}$ of body j relative to body i in frame k
x_k, y_k, z_k	X,Y,Z-axis of frame k
v_k	Velocity v of body j
μ_k	Central body constant μ of body j
r_k	Radius r of body j

List of Body Indices j / i

earth	Earth
satellite	Satellite
earth-sun-direction	The intersection of the perpendicular line of the related body to the sun-earth-direction

List of Frame Indices k

ECEF	Earth-centered, Earth-fixed
ECI	Earth-centered inertial
ICRF	International Celestial Reference Frame

List of Abbreviations

2D Two Dimensional . . . 16
3D Three Dimensional . . . 16
ADCS Attitude Determination And Control System . . . 6
API Application Programming Interfaces . . . 3
API Application Programming Interface . . . 3
AX.25 Amateur X.25 . . . 5
CEO chief executive officer . . . 26
CK C-Matrix Kernel . . . 34
CNES Centre national d'études spatiales . . . 13
CPU Central Processing Unit . . . 25
CPU Central Processing Unit . . . 25
CSV comma-separated values . . . 42
DLR Deutsches Zentrum für Luft- und Raumfahrt . . . 7
DSK Digital Shape Model Kernel . . . 35
ECEF Earth-centered, Earth-fixed . . . 3
ECI Earth-centered inertial . . . 2
EK Event Kernel . . . 34
ESA European Space Agency . . . 13
ESOC European Space Operations Centre . . . 13
FCS Frame-Check Sequence . . . 5
FK Frames Kernel . . . 34
GEO Geostationary Earth Orbit . . . 7
GMSK Gaussian Minimum-Shift Keying . . . 5
GPS Global Positioning System . . . 7
GUI Graphical User Interface . . . 21
HTML HyperText Markup Language . . . 4
HTTP Hypertext Transfer Protocol . . . 4
ICRF International Celestial Reference Frame . . . 2
ICS Integrated Computer Solutions . . . 26
IDE integrated development environment . . . 33
ID identifier . . . 3

1 Introduction

1.1 General

This work is part of the Students-on-Orbit-Measurements Project (SOMP) of the Technical University of Dresden, during which CubeSats were developed to measure oxygen in the upper atmosphere of the earth. Their latest satellite SOMP 2b (orbit information are displayed in table 1.1) was launched on 24.1.2021 and offers advanced capabilities for the entire measurement of the residual atmosphere. It further contains experiments to study carbon nanotubes under space conditions and test the conversion of solar heat into electrical power.
To maintain the satellite and schedule its experiments, predictions and visualisations of the satellite's orbit, solar illumination state, and orientation are required. During this thesis, existing software solutions are investigated and compared. Afterwards, a software tool, which is adapted to the specific requirements of the SOMP, is implemented and verified.

SOMP 2b	Orbit Attributes
NORAD ID	47445
Perigee	526.7 km
Apogee	542.3 km
Inclination	97.5 °
Period	95.2 minutes
Semi major axis	6905 km
Launch date	January 24, 2021

Table 1.1: Overview of an Extract of the Orbit Attributes for SOMP 2b [53]

Since the ground station of the TU Dresden has only short intervals in which telemetry data of SOMP 2b can be directly received, a software tool is required to access the database of SatNOGS worldwide ground station network. The SatNOGS network is further introduced in section 1.1.2. The information thereby obtained must be visualised to analyse the satellite and should be used to predict overflights overground station.

1.1.1 Reference Frames

In this section, the reference frames used during this work are introduced. Its definition and differentiation to a coordinate system are adopted from National Aeronautics and Space Administration (NASA)'s Navigation and Ancillary Information Facility (NAIF) [4].
A **reference frames** is represented by a set of three orthogonal, time-dependent unit-length direction vectors that are right-handed. In comparison, a **coordinate system** is a mechanism for locating points within the former. Therefore to describe the state (position and velocity) of an object, both the reference frame and a coordinate system are used. [4]

For reference frames, there is a further type of separation between inertial and non-inertial. Inertial frames have no rotation relative to the stars, and the acceleration of their origin is negligible. An example is the J2000 frame (described in the following sub-section). In contrast, these restrictions do not apply for non-inertial frames like the local vertical, local horizontal (LVLH) frame (see section 1.1.1). An overview of the reference frames is displayed in fig. 1.1.

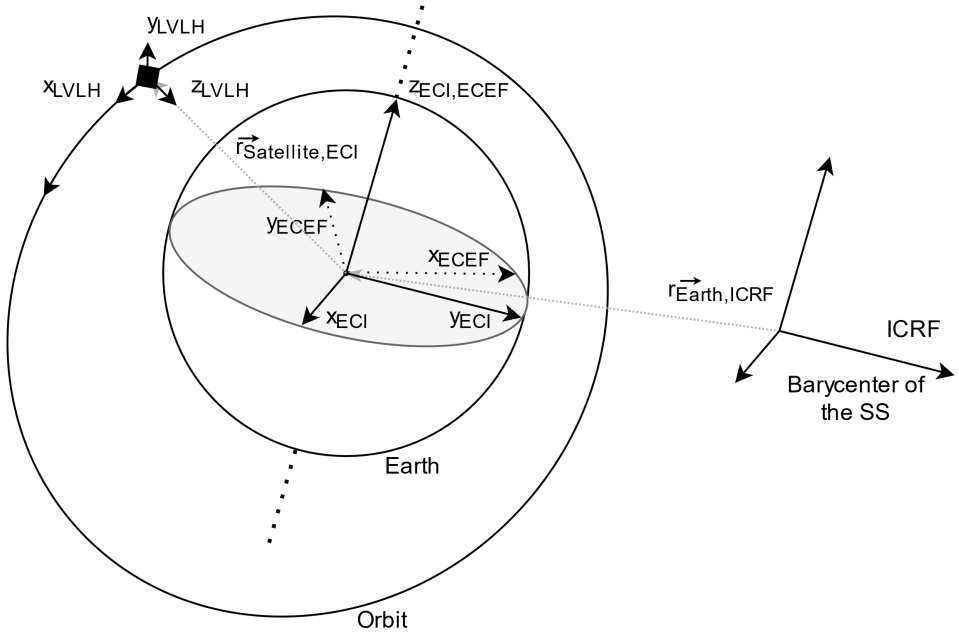

Figure 1.1: Overview of the ECI, ECEF, ICRF Reference Frames and their Axis

J2000

The J2000 or EME2000 is an inertial frame defined at the J2000 epoch (1.1.2000 12:00:00 Barycentric Dynamical Time (TDB)). Its x-axis is collinear to the earth's vernal equinox (intersection of the equatorial and the ecliptic plane). The z-axis is normal to the mean equator and, therefore, approximately the earth's spin axis. The y-axis is defined as orthogonal to the x,z-plane. The J2000 frame is used as Earth-centered inertial (ECI) frame throughout this work. [4]

As a side note, the axis of the International Celestial Reference Frame (ICRF), which is centred at the barycenter of the Solar System (SS), almost exactly coincides with the J2000 frame. Both are displayed in fig. 1.1. [4]

ECEF

The Earth-centered, Earth-fixed (ECEF) frame is contrary to the ECI non-inertial and follows the earth's rotation. Its x-axis aligns with the intersection of the equator and the 0° meridian. Its y-axis extends through the true north. Therefore the y-axis aligns with the 90° meridian and the equator. Since this z-axis is not equal to the earth's rotation axis, a nutation and precession movement can be observed.[56]

LVLH

The LVLH can be considered as an "orbit-fixed" frame. Therefore its centre is the spacecraft's mass centre, and its x-axis is pointing to the centre of the earth. Its y-axis is defined as the negative normal of the orbit plane, and the x-axis is perpendicular to both (considering the right-hand rule). [19]

1.1.2 SatNOGS

In this section the Satellite Networked Open Ground Station (SatNOGS) is introduced. It combines an open-source hardware platform, the SatNOGS Ground Station and free operating Software. The latter contains [47]

- the SatNOGS Client, which is connected to the ground station to record and schedule its observations.
- This data is then transmitted and stored in the online SatNOGS Network [48].
- Additionally the SatNOGS Database provides information about all observed satellites and transmitters and gives easy access to their observations. [46]

Through its users and their ground stations, SatNOGS provides Low Earth Orbit (LEO) satellite operators with the possibility to receive the transmissions of their satellites from all around the world. This is particularly useful for small projects with limited budgets that do not have their own or limited ground infrastructure.
Each observation can be viewed directly on the SatNOGS website. An example observation for the SOMP 2b on the 26.04.2021 is provided in the bibliography [50]. These observations contain general information like North American Aerospace Defense Command (NORAD) identifier (ID), observer, a time frame and metadata. The transmitted data is provided in the form of [50]

- an audio file,
- each individual package as binary data in hexadecimal representation
- and as visual waterfall (see fig. 1.2).

Besides their normal website, the observation data can also be obtained through a REST Application Programming Interface (API) [49].

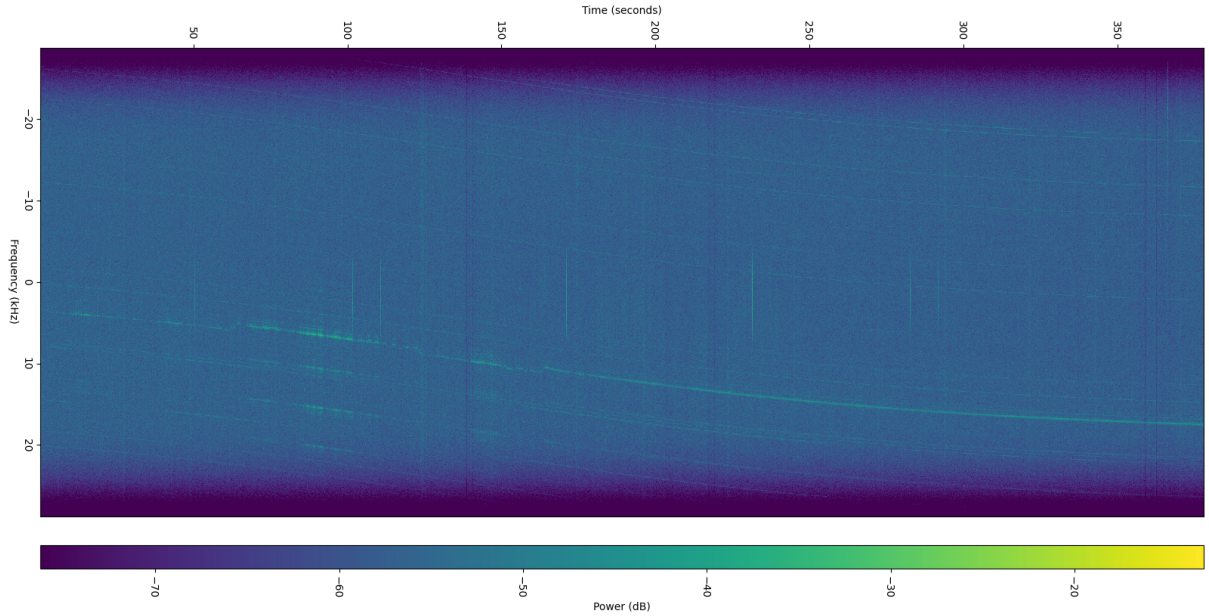

Figure 1.2: Waterfall of SOMP 2b Observation 4007887 provided by SatNOGS [50]

REST API

The representational state transfer (REST) API is a commonly used software architectural style to interact with a web service over Hypertext Transfer Protocol (HTTP). It generally provides the operations [44]

- GET,
- POST,
- PUT and
- DELETE.

By manipulation the Uniform Resource Locator (URL), one can, for example, query a web service for certain information with a REST API GET call. The following URL represents such an exemplary GET query. It requests the observations on the 29.4.2021 between 00:00 and 12:12:33, that contain the NORAD ID 47445 (SOMP 2b), have a good observation status and are on page one. [44]

```
https://network.satnogs.org/api/observations/?end=2021-4-29T20%3A12%3A33&
    page=1&satellite__norad_cat_id=47445&start=2021-04-28T00%3A00%3A00&
    status=good
```

The payload of such an REST API request can be formatted in HyperText Markup Language (HTML), Extensible Markup Language (XML) or JavaScript Object Notation (JSON). SatNOGS provides its answers in the latter format. The payload can also provide links to resources, like images or binary data.

1.1.3 Transmission Protocols

This section explains the data structure of a SOMP 2b transmission, which contains the telemetry and payload data. On the physical level (corresponding to the Open Systems Interconnection (OSI) model layer 1), the digital data is modulated by the satellite using the Gaussian Minimum-Shift Keying (GMSK) method with a baud rate of 9600. Since the demodulation takes place during the receiving process, it is not further described in this work. The user must then further decode the generated binary data to obtain its information. The two thereby used protocols are shown in fig. 1.3 and explained in the following subsections.

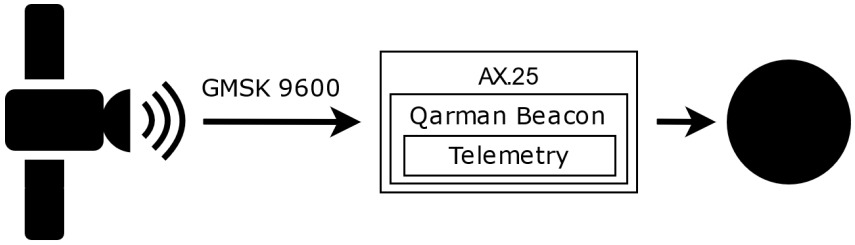

Figure 1.3: Overview of Transmission Protocols

AX.25

Amateur X.25 (AX.25) is a protocol occupying the data link layer (layer 2) in the OSI model. This means it creates a direct link between nodes. Therefore it contains address information, frame start and end bits, and a checksum to ensure a correct transmission by the physical layer. AX.25 is often used for package-based transmission and works equally well in half-or full-duplex environments. [1]

Flag	Address	Control	Info	FCS	Flag
01111110	112/224 Bits	8/16 Bits	N*8 Bits	16 Bits	01111110

Figure 1.4: AX.25 Frame Construct [1]

An unnumbered AX.25 frame contains (see fig. 1.4) [1]:

- Two **Flag Fields**, which delimits frames. It is important to note that a 01111110 sequence must not occur within the frame.
- An **Address Field** that contains source and destination information, as well as the command/response information.
- A **Control Field** containing an identifier of the frame type and attributes concerning the OSI layer 2.
- A **Data Payload field**, which is having a size that is a multiple of 8.
- A **Frame-Check Sequence (FCS)** to evaluate if the frame was corrupted.

For transmissions of SOMP 2b, the address field has a size of 112 bit and the control field a length of 8 bit. Therefore the first 16 bytes of each received package are the AX.25 header, and after that, the telemetry and payload data is placed.

SOMP 2b Beacon

The SOMP 2b Beacon finally describes how the binary payload data of the AX.25 frame can be converted to usable information. The beacon definition (see appendix B.1) contains two individual decoders, the Standard Beacon, which is sent every 60s and an Alternative Beacon which is sent every 180s.
The former contains important telemetry data for satellite sub-systems like

- the batteries,
- the solar panels,
- the electrical supply and their distribution,
- the communication systems,
- the On Board Computer (OBC),
- the payload
- and finally the Attitude Determination And Control System (ADCS).

The majority of these values are defined with a bit depth, a divider, and an offset. They can be unpacked using eq. (1.1).

$$val_{out} = \frac{val_{packed}}{val_{divider}} - val_{offset} \tag{1.1}$$

The ADCS-part of the standard beacon contains the values for the angular velocities and the estimated attitude (in the form of a quaternion from the ECI to the satellite frame). These values will be used for the satellite in-orbit visualisation.
The second beacon contains some redundant information and more details about the status of the batteries and the communication system. To unpack its values, no conversion formula is required.

1.1.4 Prediction and Model Scope

This section will discuss which models are in general required for the implementation of the described tool. Furthermore, their specific requirements and the expected accuracies are defined.

For the pass and illumination prediction, and the satellite visualisation, an orbit model is necessary. Since SatNOGS provides its orbital data in the form of the Two-line element set (TLE) which contains the orbital parameters used in the Simplified General Perturbations (SGP) and Simplified Deep Space Perturbations (SDP) models. These parameters could, in theory, be used as general orbit parameters and, therefore, for different orbit models. Nevertheless, because these are "mean" values and are directly adjusted for the SGP and SDP algorithms, the usage of different algorithms would result in degraded prediction accuracy. A full description of these algorithms can be found in the Spacetrack Report no. 3 from 1980 [15].
The major difference between the SGP and SDP models is that the SDP algorithms are specialised for deep-space satellites (period > 225 minutes) and therefore contain the gravitational effect on the sun and the moon. Since SOMP 2b's orbit does not fall into this category, this project is focused mainly on the SGP variant and specifically on the two most recent implementations, the SGP4 and SGP8. [15] These two models are based on the same gravitational and atmospheric models, [15] but the SGP8 provides better reentry predictions [55]. Since this thesis is not concerned with the reentry, these two models are completely interchangeable from the application's perspective, and the selection is made during the implementation of the orbit model.
In the Revisiting of Spacetrack Report no. 3 [15] from 2006, some changes to the original SGP and SDP algorithms are mentioned that have manifested themselves over the years. These changes concern the TLE data format, coordinate system, time systems, and more. Therefore the implementation of the most recent version of the respective algorithms should be used.
In terms of TLE/SGP4's accuracy, multiple resources were analysed, and their results are here summarised.
A statistical error analysis of more than 150 000 Medium Earth Orbit (MEO) and Geostationary Earth Orbit (GEO) TLEs [43] showed that the TLE error at epoch has significantly declined over the last 20 years and can be estimated for MEO object at $< 500m$. For GEO satellites, the error is about twice the size. It furthermore indicates that the along-track tack error for MEO dominates the cross-track and radial error. Their data points for Global Positioning System (GPS) MEO TLE sets are represented in fig. 1.5. [43]
In 2013 the Deutsches Zentrum für Luft- und Raumfahrt (DLR), has compared TLE/SGP4 predictions to high accuracy ephemerides of their LEO satellite GRACE-1 (at an altitude of $450km$). [2] At epoch, their results were similar to the pre-2015 results of the MEO GPS TLEs. Furthermore, they have investigated the propagation accuracy for periods of 2-3 and 6-7 days. Their results show that the radial and the cross-track error are dominated by fluctuation and that their error is relatively small compared to the along-track error. The results of the latter indicate [2]

- a mean error of around $6000 - 8000m$ with a standard deviation of around $1700m$ for a period of 2-3 days,

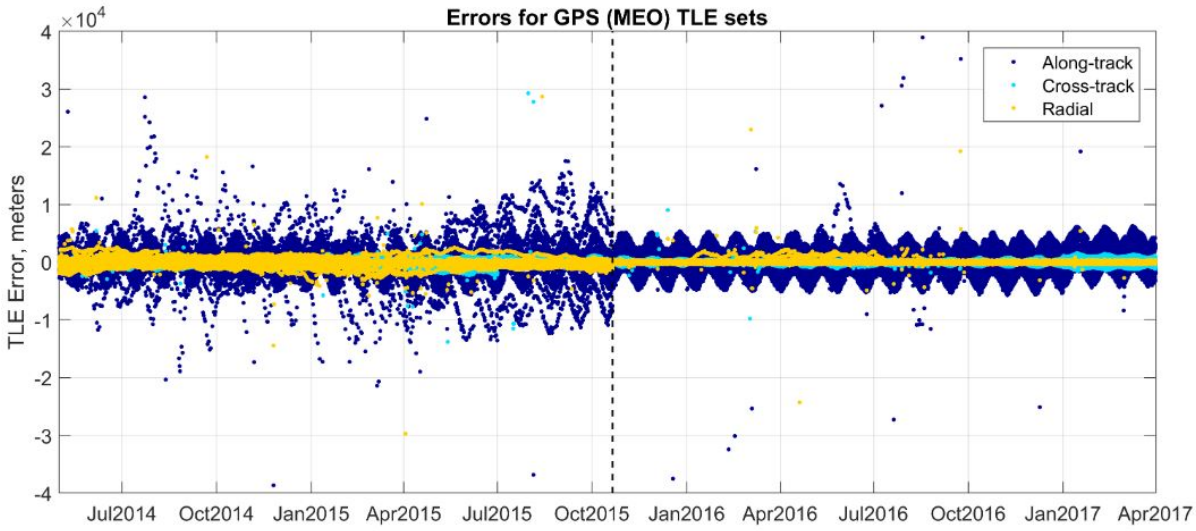

Figure 1.5: Errors of GPS MEO TLE Sets [43]

- and a mean error of around $30000m$ combined with a standard deviation of $3000m$ during a propagation period of 6-7 days.

A research by NASA [27] which again was focused on MEO and GEO satellites, shows that the TLE/SGP4 propagation error depends on the orbit of the satellite and therefore indicates the difficulties in its prediction. In this paper, an instantaneous range bias of $0.8 \pm 0.3km$ and an error growth of $1.5km/day$ is mentioned.
Lastly, the earth imaging company Planet Labs provide an overview of their satellites with an orbit height between 400 to $800km$ and the Root Mean Square (RMS) error between their position determination and the TLE position. Besides some outliers, this error is ranging from 0.3 to $1.5km$. [41]
The results of these different sources show that it is pretty complicated to give a definitive answer to the accuracy of the TLE/SGP4 model for LEO satellites. Mainly because their orbit is the most influenced by external disturbances like the earth atmosphere. Additionally, the significant along-track error directly impacts the prediction of future passes above-ground stations and the calculation of eclipse times. An error in this direction would delay or premature the estimated event based on the satellite's in-orbit velocity. For a circular orbit with SOMP 2b's minimum height, the velocity can be calculated via a simplified Vis-Viva equation. See eq. (1.3).

$$h_{min;SOMP2b} = 526.8km; \mu_{earth} = 3.986 * 10^{14} \frac{m^3}{s^2}; r_{earth} = 6378km \tag{1.2}$$

$$v_{satellite} = \sqrt{\frac{\mu_{earth}}{r_{earth} + h_{min;SOMP2b}}} = 7.6km/s \tag{1.3}$$

Therefore the prediction of passes and eclipse times for 2-3 days using the TLE/SGP4 model could provide an approximated accuracy below $1s$ if the errors are based on the DLR report

[2]. Meanwhile after 6-7 days a mean error of $\approx 4s$ can be expected. Although the study by NASA indicates a greater than linear increase in the in-track error (see fig. 1.6), for simplicity an estimated error increase of $\approx 5km/day$ for LEO TLE/SGP4 prediction would result in $\approx 0.66s/day$ of event epoch miscalculation.

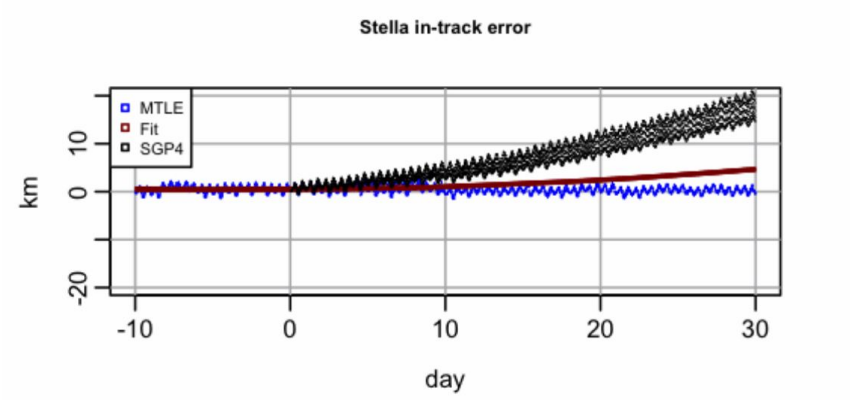

Figure 1.6: Growth of in-track Error on Example of the Spacecraft Stella [27]

The additional models required during orbit and illumination state prediction include an earth orbit model and a conversion from the ECI to the ECEF frame. Both should archive an accuracy of $< 1s$ for these predictions. The selection of these models is discussed in chapter 4.

1.1.5 Software Requirements

In general, the software needs to be easily modifiable and extendable to accommodate changes, which might be required during its period of use. An example of these changes could be a correction in the SOMP 2b beacon definition or the addition of a minor feature in the visualisation. This could make the tool a basis for future development by the SOMP team and might even open possibilities for different satellites.
From the technical point, the programming language C++ and the Qt framework [42] version 5.x should be used. This enables a possible integration of the developed software module into the existing ground command software of the SOMP 2b team. The Qt framework is an open open-source, cross-platform toolset for graphical user interface application development. Its version 5.15.2 combined with the MSVC2019 64 bit compiler is used during this work, and a focus was placed on the Windows 10 platform. For documentation purposes, this work and in-code commenting suitable for the Qt5 QDoc style are provided for the software.

1.1.6 Methodology

In this last section of the introduction, the methodology of this work is explained. During section 1.1 the problem was defined, and the necessary reference frames, transmission codings, and software requirements for this work were introduced. In section 1.1.4 the required models are named and their required/possible accuracy's are discussed. The second part of the

introduction represents a state-of-the-art overview of existing ground command, orbit, and ADCS visualisation tools.

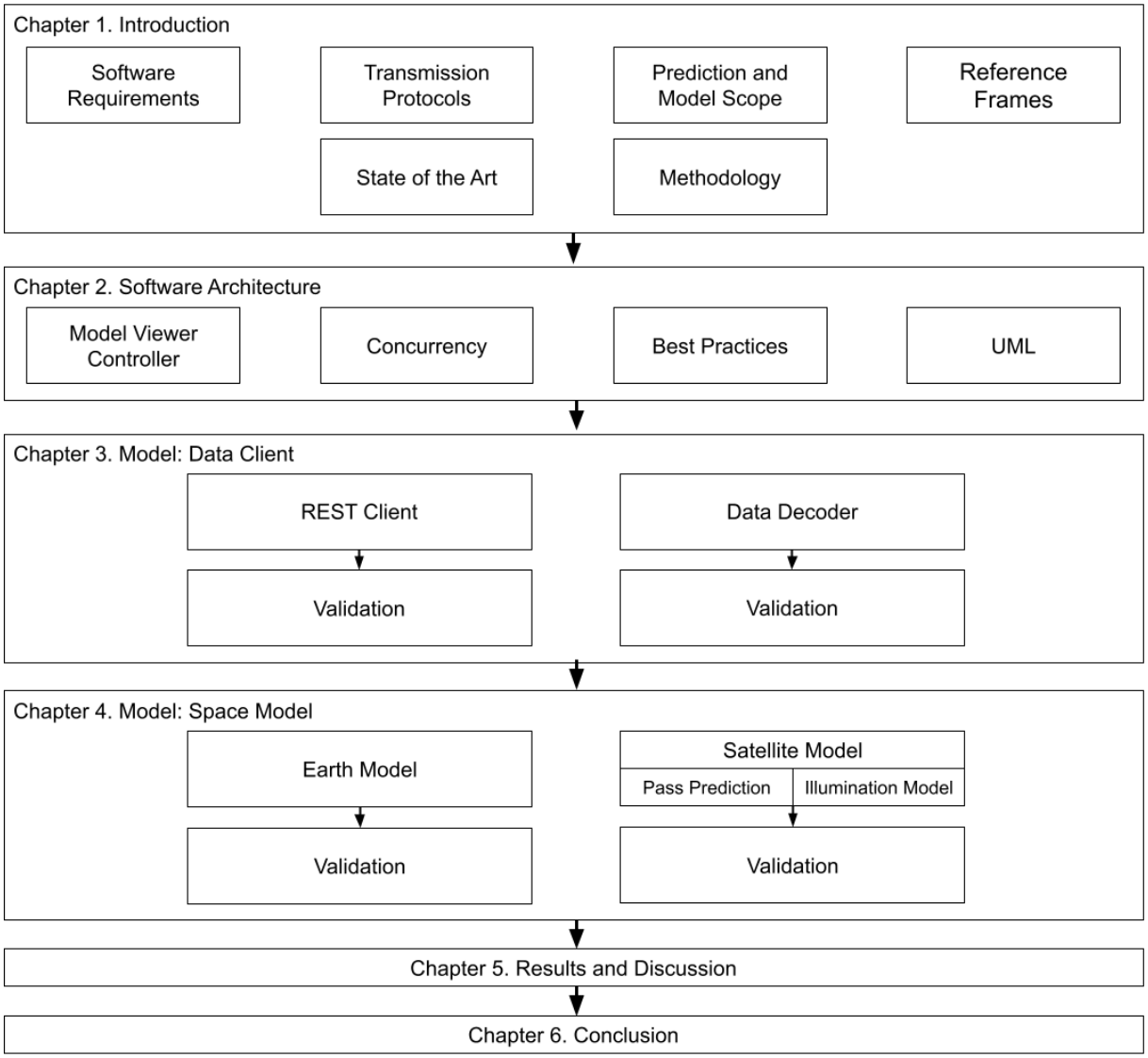

Figure 1.7: Methodology Overview

Afterwards in chapter 2 a general software architecture for the SOMP 2b orbit visualization tool is chosen. Thereby the MVC pattern and the usage of concurrency are discussed. During chapter 3 and chapter 4 the implementation and verification of the major software modules takes place. The former is about the implementation of the Data Client and therefore contains the integration of SatNOGS and the decoding of the satellite transmissions. The latter deals with the physical models of the satellite and earth orbit, the earth's rotation, as well as the pass and eclipse prediction.
Finally, the results of the developed tool are discussed and reviewed versus the initial requirements in chapter 5, and a conclusion is drawn in chapter 6. In the end, the possibilities for future work are presented in chapter 7. This methodology is additionally visualized in fig. 1.7.

1.2 State of Art

In this chapter, an in-depth look is taken at the solutions available on the market. Therefore, all software that can be vaguely described as base station software for small satellites is considered. As such, the software must fulfil one of the following tasks:

- Display the current satellite's position
- Communicate with the spacecraft
- Store or display received information of the satellite

Since many software applications meet these requirements, the results of this analysis can provide a broad overview of existing solutions. Nevertheless, these results are not intended to be an exhaustive list. In the following sections, a short overview is provided of every considered software. Afterwards, their primary attributes are compared in a table. As a starting point, for the search of existing software, the list of suppliers of the CubeSat organisation was chosen [28].

1.2.1 Open MCT

Open Mission Control Technologies (MCT) (see fig. 1.8) is an open-source framework developed by NASA and the Jet Propulsion Laboratory (JPL). It is mainly supposed to create applications for commanding and analysing telemetry data of spacecraft but could also be adapted for applications like Drones, Internet of Things (IoT) devices, Robotics, and more. From its core up, it is a web-based framework and therefore allows simple distribution of operations and even mobile access. As of flight heritage, it is currently used by NASA itself and in projects like Mars Cube One and Mars 2020 (a full list is displayed on their homepage [38]).
From a programming point of view, it is build using Node.js, a common JavaScrip runtime environment. To further extend the base functions of Open MCT a range of plugins is available, which for example, allow databank storage. [38]
NASA does provide an interactive preview [40].

1.2.2 Elveti MCT

Elveti (see fig. 1.9) is an on global aerospace standards-based flight-proven MCT. It is mainly focused on commanding and monitoring purposes for nano and small satellites. Solenix claims that their Elveti MCT supports multi-satellite and multi-ground-station capabilities as well as extensible and customisable features. However, it must be said that besides the official website, there is less information publicly available. [11]
Compared to all other competitors, Elveti MCT offers a unique feature. In fig. 1.8 one can see a 3D rendering of the satellite model, which shows the angle between the surface normal and the sun vector and therefore might work if extended as a visualisation of the ADCS.

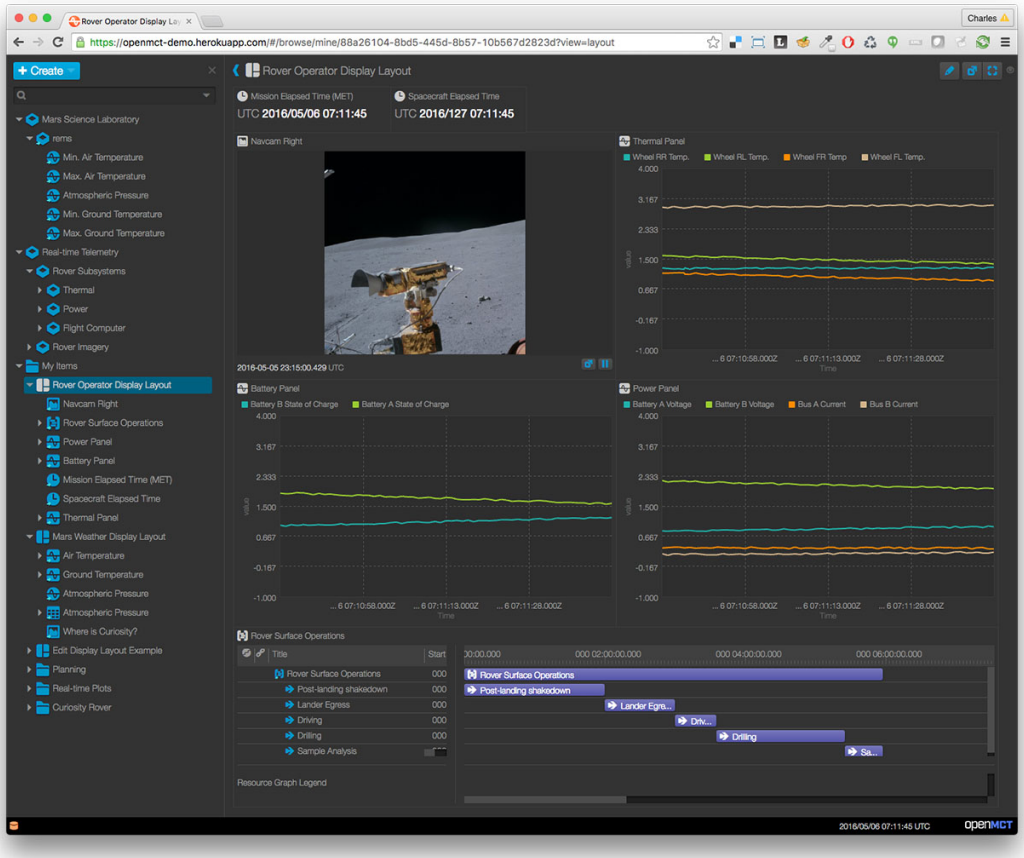

Figure 1.8: Open MCT Overview [38]

Figure 1.9: Elveti MCT Overview [12]

1.2.3 BOSS Dashboard

Additionally, Solenix provides the BOSS Dashboard (see fig. 1.10), a web-based monitoring system for satellites and other complex systems. BOSS present data in the form of multiple customisable widgets to show trends, thresholds and exceptions. Solenix claims that flexible connectors are available, which can gather data from multiple heterogeneous systems. [7] The dashboard is based on the Elastic Stack, which uses Kibana for data visualisation, Logstash for data processing and Elasticsearch as a database search engine. [7]

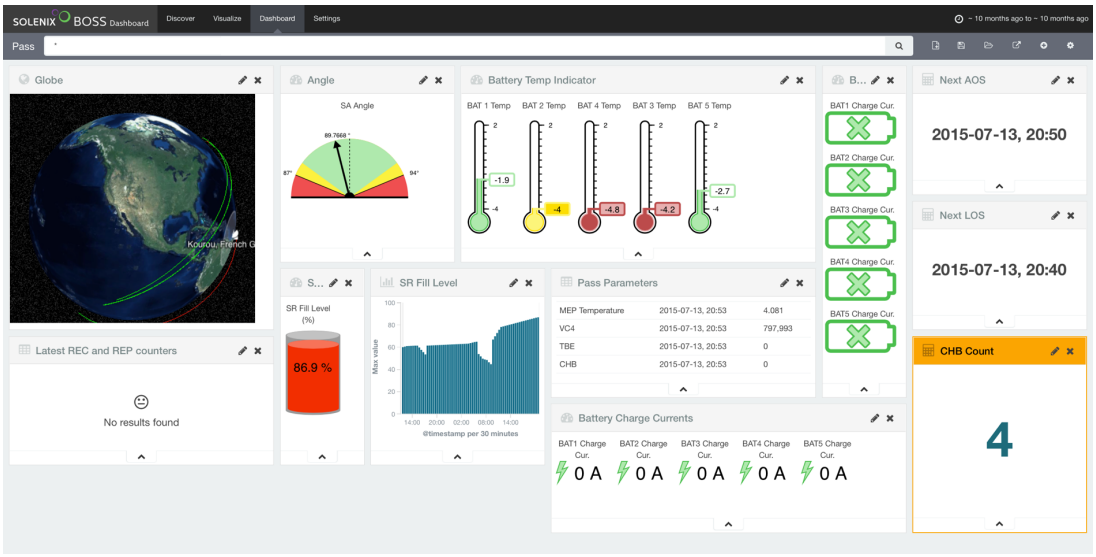

Figure 1.10: BOSS Dashboard Overview [7]

1.2.4 MUST: Mission Utility and Support Tools

Mission Utility and Support Tools (MUST) is the third tool in this comparison developed by Solenix. It is used for data analysis and is covering the following areas:

- Analysis
- Visualisation
- Storage

Initially, MUST (see fig. 1.11) was developed as an efficient telemetry data storage. Later it was greatly extended to store more data types (like telecommand history), a graphical user interface and advanced features like automatic anomaly detection. MUST's development took place under a contract between Solenix and the European Space Agency (ESA). Additionally, MUST 1.2 was created to accommodate the needs of European Space Operations Centre (ESOC), DLR and Centre national d'études spatiales (CNES). [30]

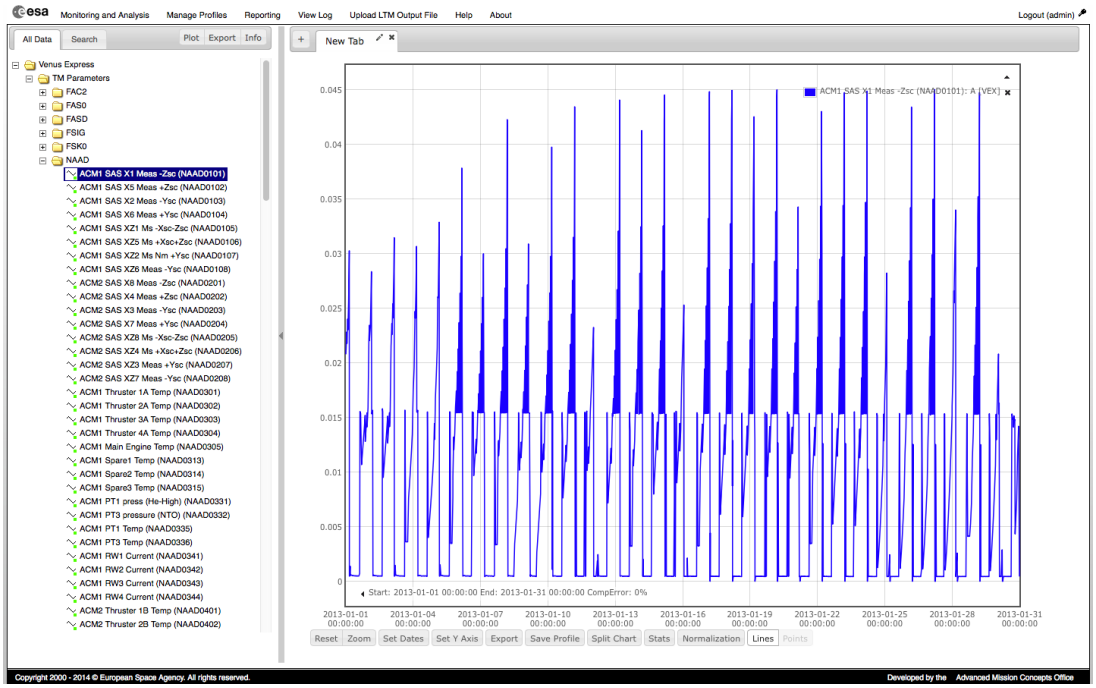

Figure 1.11: MUST Overview [30]

1.2.5 Alén Space's MCS

Alén Mission Control Software (MCS) (see fig. 1.12) is a modular and extendable ground software that uses the ESA Packet Utilization Standard (PUS). Its ground server is separated from the interface, which allows the use of multiple independent frontends. This MCS provides standard functions like telemetry data reception, storage and decoding, as well as tools for telecommand sequences management and their execution. The user interface is web-based. [3]
To further reduce the clients need for software development, Alén Space provides On-Board Software (OBSW).[3]

1.2.6 Kubos: Major Tom

Major Tom (see fig. 1.13) is a ground operation solution developed by Kubos. It provides, in addition to all standard features like telemetry, commanding, data visualisation, also advanced functionality like:

- Satellite tracking through orbital propagation
- Satellite development environment, which contains satellite pre-flight testing tools
- File transfer for software updates or payload files
- Operation scripting
- Interfacing with simulators for testing

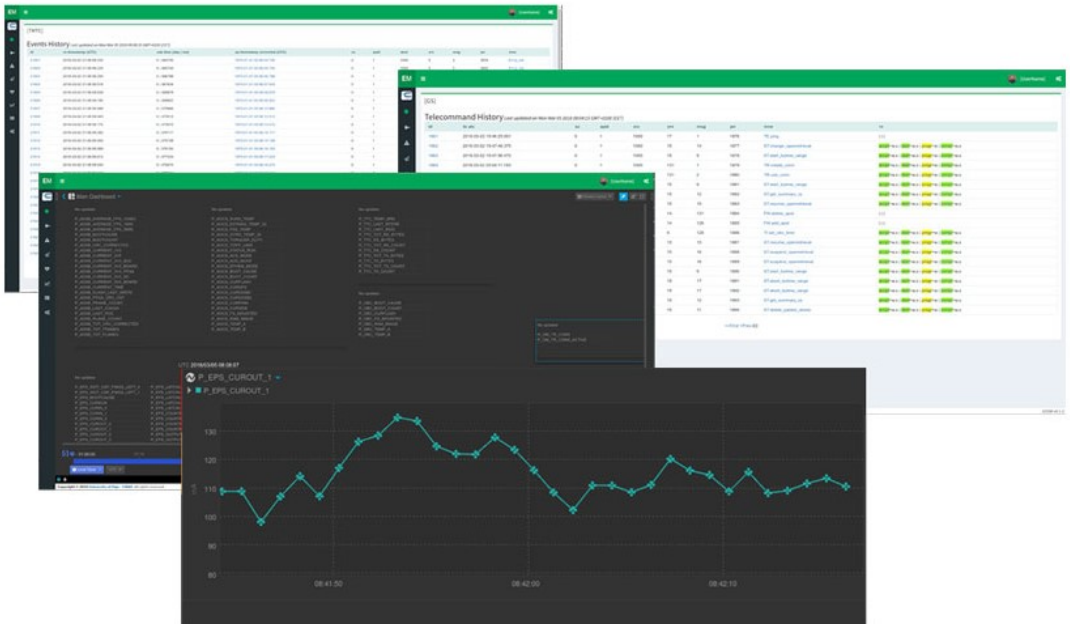

Figure 1.12: Alén Space's MCS Overview [3]

Contrary to its competitors, Major Tom is a cloud-native application and uses a microservice architecture that increases its overall modularity and customizability. Additionally, the software uses web standard API, which, so Kubos claims, allows Major to integrate with a majority of systems and to streamline the user experience. As shown in fig. 1.13, it also utilizes the Grafana framework. [24]

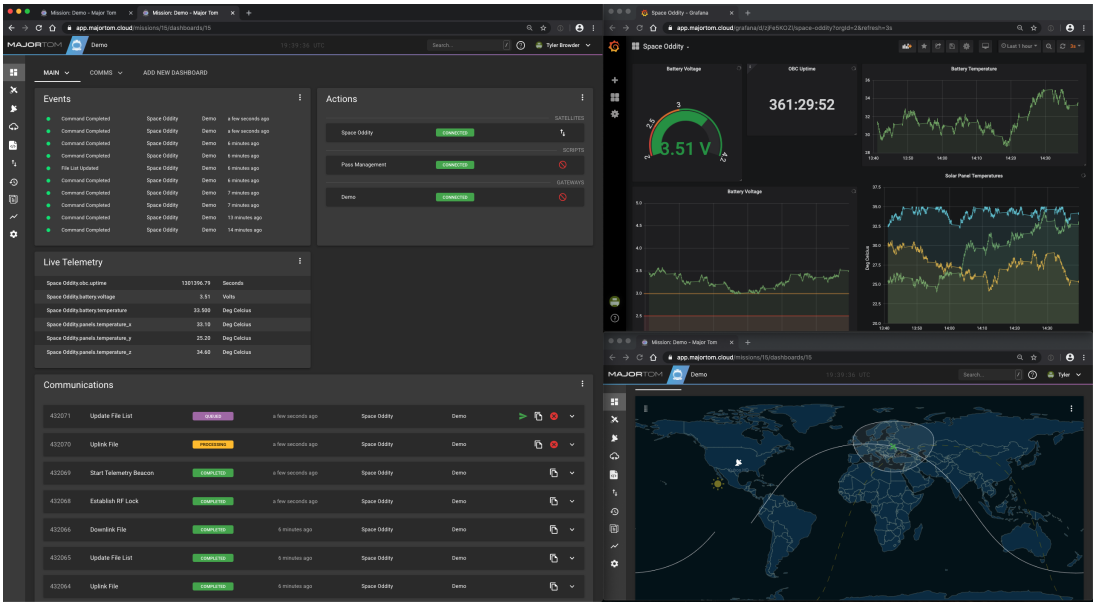

Figure 1.13: Kubos: Major Tom Overview [24]

1.2.7 SatNOGS Dashboard

SatNOGS is an open-source project, which contains not only an MCS but also a ground station and a network connecting all of them worldwide (see section 1.1.2). This network allows observers to access his LEO satellite from multiple ground stations worldwide. [47]
The collected telemetry data is stored inside the SatNOGS DB and can later be visualised through a dashboard based on the Grafana framework. As an example, the page of the satellite Bobcat-1 of the University of Ohio is shown in fig. 1.14. The SatNOGS dashboard does also provide commanding features. [47]

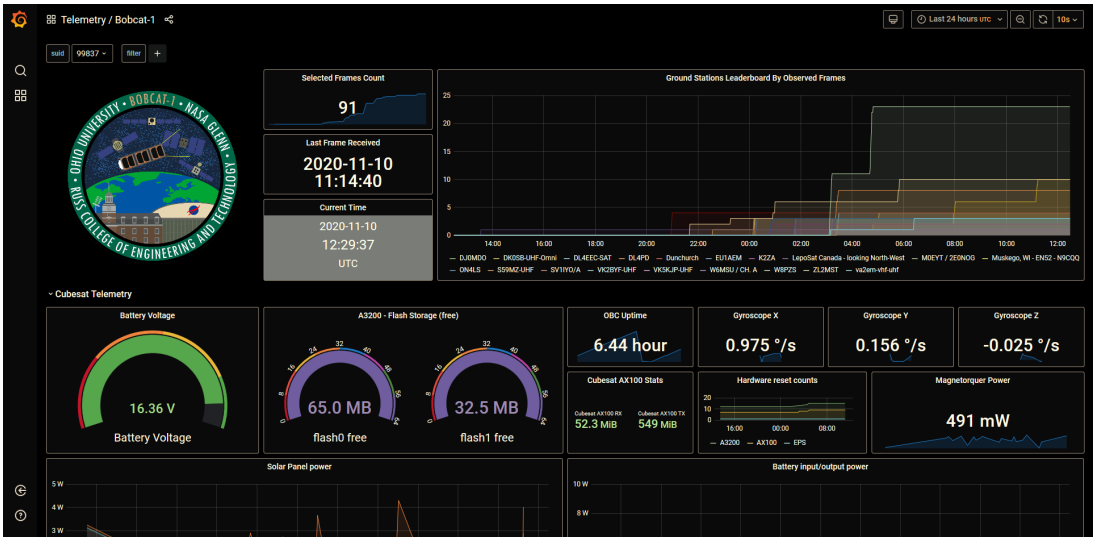

Figure 1.14: SatNOGS Telemetry of Satellite Bobcat-1 [45]

1.2.8 Nova for Windows

Nova (see fig. 1.15) for Windows does not fit the category of an MCS. Its primary goal is satellite tracking. Therefore it supports special features like orbit models (SGP4), turning antennas for auto-tracking or sound alarms when a satellite is in line of sight. As input, it uses TLEs, which can either be typed in directly or updated over the internet. Even though it is the oldest tool mentioned in this work (1. revision in 1996), it still provides Two Dimensional (2D) and Three Dimensional (3D) visualisations of the position and path of the satellite. In 2017 Northern Lights Software made it officially available for free. [37]

1.2.9 JSatTrak

JSatTrak is a Satellite tracking program written in Java. It can predict the position of any satellite in real-time, past or future. It uses advanced SGP4/SDP4 algorithms developed by NASA/NORAD or customisable high precision solvers to propagate satellite orbits. The program also allows for easy updating of current satellite tracking data via CelesTrak.com. Because this application was written in Java, it should run on almost any operating system or directly off the web. [20]

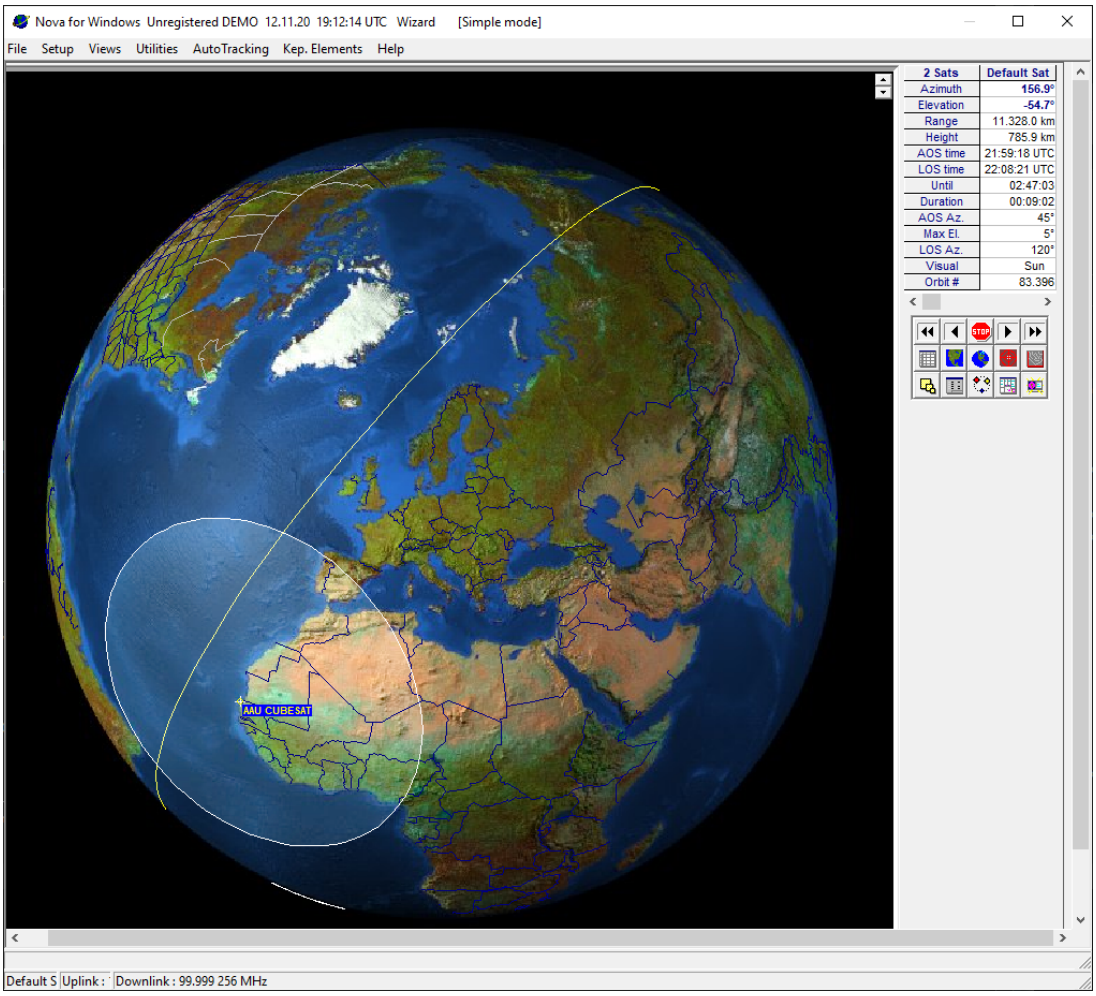

Figure 1.15: Nova for Windows Overview

1.2.10 Comparison of existing Tools

Table 1.2 compares all tools covered in this state of art analyses. In general, it was relatively hard to find in-depth information about the majority of the professional tools.
The results show that most professional ones are targeted for big projects than nanosatellites projects with a limited budget. Therefore all besides the SatNOGS Dashboard require the satellite operator to have their own ground infrastructure to receive transmissions. In general, all of them are older than five years. Only OPEN MCT, Major Tom and SatNOGS were developed more recently. All newer ones are web-based.
Compared to its competitors, NASA's OPEN MCT seems to be the most versatile. Since it is maintained by NASA and also open-source, which means it can be adopted and extended by different projects, OPEN MCT can become a general standard.
For the requirements of the SOMP team, the best fit would be the SatNOGS Dashboard and the OPEN MCT. However, because the dashboard is web-based and runs on SatNOGS servers, its possibilities for extension are limited. Especially for calculation-intensive additions like a 3D orbit visualisation. OPEN MCT on the other hand, could be extended to use SatNOGS data and

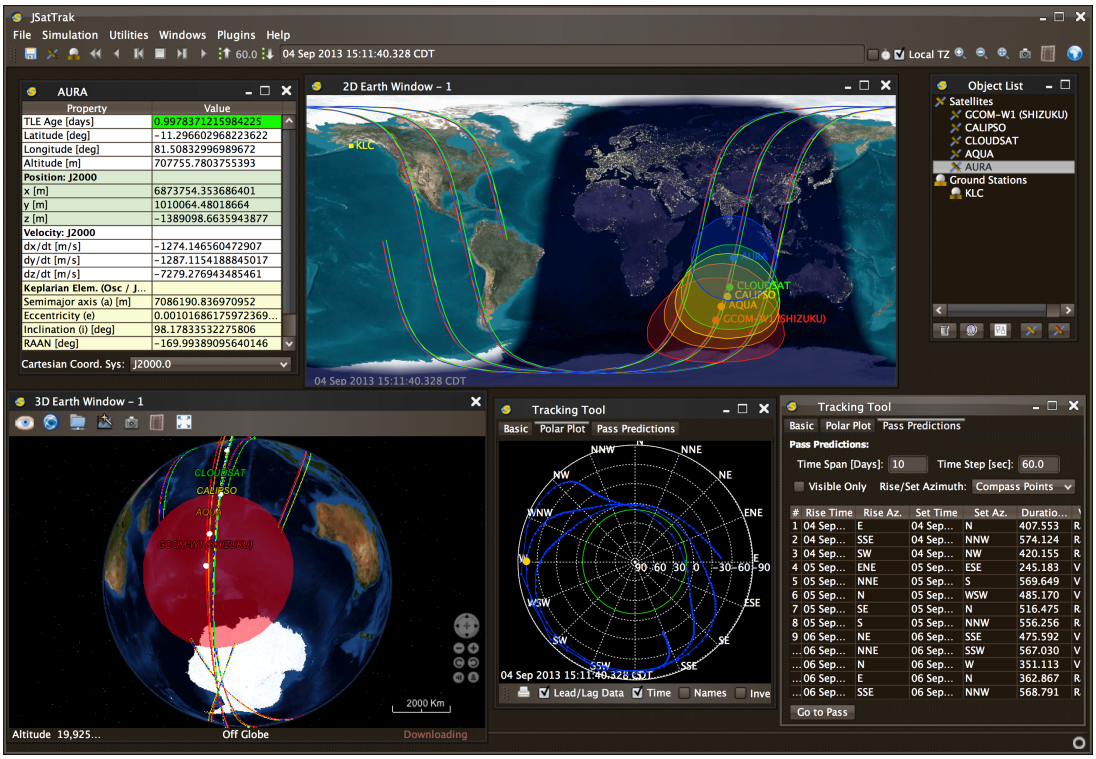

Figure 1.16: JSatTrak Overview [20]

to contain the required calculations and visualisations. Nevertheless, it also requires additional infrastructure like a maintained server. Additionally, it uses completely different programming languages than those used by the SOMP projects. Therefore modifications would result in more effort. Finally, the endeavour to integrate all necessary functionality into OPEN MCT would be similar to implement a specialised in-house tool. Therefore it is decided for the latter.

	OPEN MCT	Elveti	BOSS	MUST	Alén	Major Tom	SatNOGS	Nova	JSatTrak
2D orbit View	no	no	no (?)	no	no	yes	no	yes	yes
3D orbit view	no	no	yes	no	no	no	no	yes	yes
ADCS view	no	no	no	no	no	no	no	no	no
Accessible via Web	yes (web based)	yes	yes (web based)	yes	yes	yes (web based)	yes (web based)	no	no
Still maintained?	yes	?	yes (?)	yes (?)	?	yes	yes	no	yes(?)
Trend views	yes	no	yes	yes	yes	yes	yes	no	no
Data Storage	yes	no	no (?)	yes	yes	yes	yes	no	no
Extendable	yes	yes	limited (?)	yes	yes	yes	yes	no	yes
Commanding	yes	yes	no	no	yes	yes	yes	no	no
Introduced in	2018*	2009	?	2003	?	2017 (?)	2016	1996	before 2008
Organization	NASA & JPL	Solenix	Solenix	Solenix	Alen Space	Kubos	SatNOGS	Northern Lights Software	Shawn Gano
Sources	[38] [39] [40]	[11] [12]	[7]	[30]	[3]	[24] [25]	[47] [45] [52]	[37]	[20]

If it is unsure or not possible to evaluate a certain feature, this is indicated by a "?". *First version which is not labelled as pre-release.

Table 1.2: Comparison of existing Ground Station Software

2 Software Architecture

"The software architecture of a system is the set of structures needed to reason about the system, which comprise software elements, relations among them, and properties of both." [5] The first step during the implementation of software is to develop a suitable software architecture. Thereby all functions of the tool must be recognised, sorted and divided into logical structures. These structures can then again be put in super-structures until all parts of the software are reasonable connected. The decisions made during the creation of this composition have a decisive impact on the systems modifiability, portability, security, performance, extensibility, development efficiency and availability. Over the years of software development, some useful constructs have crystallised. Each of them has its ups and downsides related to the previously mentioned attributes. In general, there are three types of software structure [5]:

- Modules, which divide the system's functionality into multiple components.
- Dynamic structures that focus on the mutual interaction between elements at runtime. They concretely describe the synchronisation and interaction between multiple services in one system, as well as the infrastructure these services interact with.
- And lastly Allocation structures. They are associated with the system's developmental, installation, execution, developmental, and organisational environment.

In reference to the software requirements section 1.1.5, modifiability and extensibility were selected as the essential software architecture attributes. Additionally, the structure should support enough performance to allow for real-time calculation, such as visualisation and predicting passes in a reasonable time. How the attributes are implemented trough structures are described in the following sections.

2.1 Model View Controller

Model-View-Controller (MVC) is a software design pattern that is based in the principle of decomposition. Therefore, it divides the software into parts with similar functionality. Corresponding to its name, the specific modules of the MVC pattern are [5]

- the View,
- the Model,
- and the Controller.

Its main target is to abstract the Graphical User Interface (GUI) from the main application without losing its responsiveness [5]. Abstraction in relation to software architecture is a fundamental

concept of encapsulating information. Its goal is to focus the developer on important information necessary for a task and hide unnecessary ones to separate placer or even remove them entirely. [9]

The first module, the View, contains the GUI to displays data and allows the user to input data. The second module, the Model, includes the application data. Thereby, data is meant in the broader sense and is not limited to raw data like databases. It can also contain physical models or calculation algorithms that need to be invoked. Lastly, the Controller makes the connection between both. Therefore, if the user interacts with the GUI, this event gets passed to the Controller, and it can then invoke the Model to initiate a state change. This is shown in fig. 2.1. In the represented form, the Model afterwards notifies the View so that it can query the new state and display the information accordingly.

During implementation, it was decided against this direct connection between Model and View because of the usage of Qt Modeling Language (QML) (see section 2.1.2) and its style of class and data exposure. These connections were rerouted over the Controller. Therefore it is not necessary to expose the Model to QML and restrict the usage of data types. This is further discussed in chapter 5.

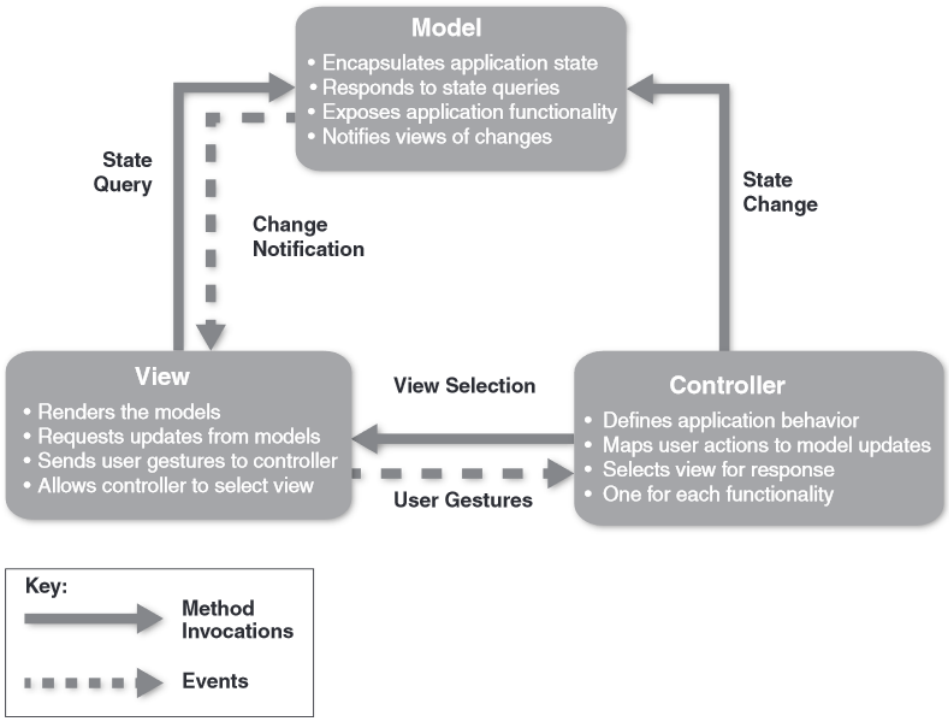

Figure 2.1: The Model-View-Controller Pattern [5]

The separation into these three modules introduces some complexity and thus might not be the best for small projects with simple GUIs. However, it allows for complete abstraction of the data and the GUI. This makes it possible to develop the Model and the View completely independent from each other. Additionally, changes can be made to either of them without

changing the other one. It is even possible to use one of the parts as stand-alone software, such as the backend as a console application. Therefore MVC provides a software structure that is highly maintainable and extendable. [5]
While MVC represents the software's basic structure, the View and Model are each sub-divided into multiple modules to further improve maintainability. Thereby the Decomposition software structure is used. However, to not exceed the scope of this thesis, it is not described in detail. A complete overview of the scope of functionality for the MVC modules is given in the following sections.

2.1.1 Model

As written in the introduction to MVC, the Model contains the data. For the SOMP2b, this includes:

- The SatNOGS data client, which is implemented and verified in chapter 3. It downloads all satellite transmission files from the SatNOGS network when given a query by the user. Afterwards, it decrypts them and provides them to the application in a suitable format.
- The "Space model" which is an umbrella term and contains the
  - satellite ephemeris and orientation model,
  - the earth orientation and the earth ephemeris model.
  - It further contains the calculation routines for the observation pass and elliptic predictions
  - and some minor routines which calculate some information necessary for the general visualisation (like orbit markers).

  Their implementations take place in chapter 4.
- The Model also provides data storage to share the data between different physics-based models and calculations, save their result and make them accessibly for visualisation.

2.1.2 View

In this section, the View's functionality or, respectively, the GUI is defined. Its architecture must be easily modifiable and allow efficient development. The View has to provide a user-friendly, easy-to-use and straightforward interface to access the satellite data, analyse the orbit and change settings. Therefore it contains

- a **Settings Window**, which is used to configure the SatNOGS attributes and initialize the download process.
- Additionally, a **Value Window** is included. It visualizes all values described in the SOMP 2b beacon through adjustable graphs.

- Further a **3D Window** for orbit visualization is contained (see fig. 2.2). It uses the Qt3D renderer, which is based on the Open Graphics Library (OpenGL) engine and the interface markup language QML. All utilized textures are provided by NASA Visible Earth [36] and are partially modified. The satellite model was acquired separately [54].
- Lastly, a **Pass Prediction Window** allows the user to set up the necessary data for the pass prediction and the eclipse prediction. It provides an interactive graph and a list of passes.

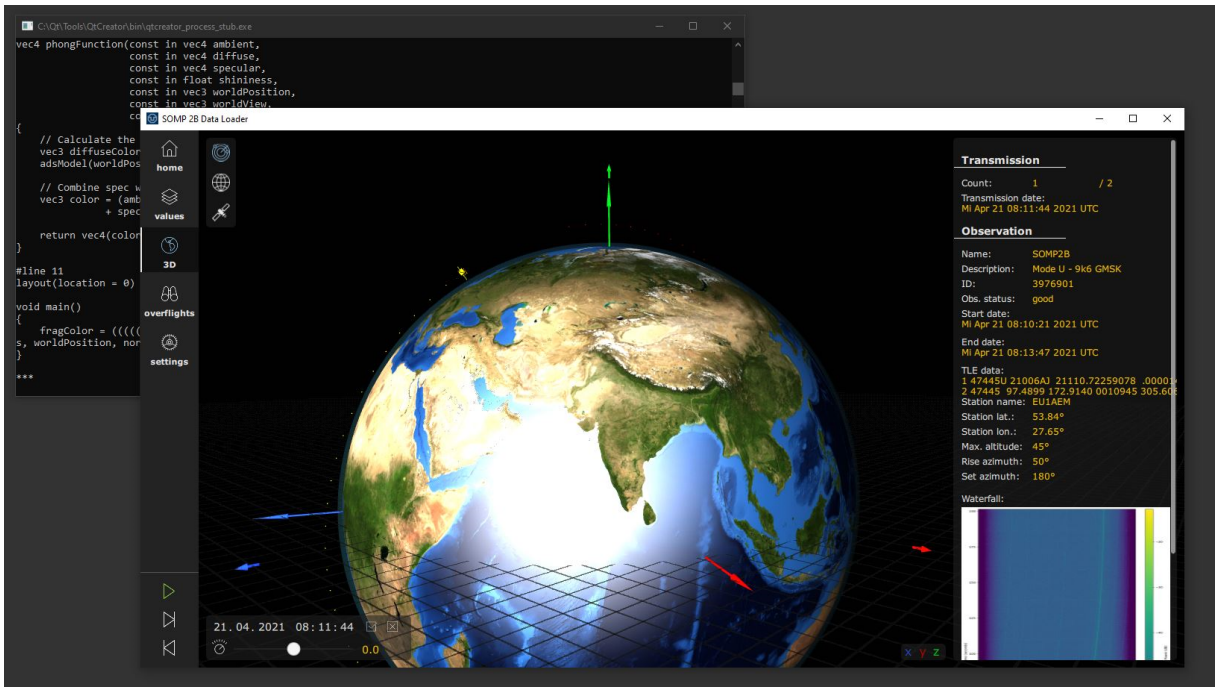

Figure 2.2: 3D Orbit Visualisation Window

All these modules use the Qt GUI framework QtQuick and, similar to Qt3d, QML as interface markup language. Additionally, some JavaScript is used to implement minimal logic into the View.

QML was used over C++ for the View to better fulfil the requirements for modifiability and extensibility. The slight performance advantage C++ has over a QML implementation is negligible because the GUI does only include two expensive computational parts. The first of these, the 3D orbit visualisation, was simple enough to not run into any performance issues. The second, the process of injecting a high amount of data efficiently into the graphs, was solved by outsourcing only this specific part into C++.

All utilised icons are contained in an image collection from flaticon [18].

Since the GUI development is not an essential part of this thesis, its further implementation is not discussed in order not to go beyond the scope.

2.1.3 Controller

The last module of the MVC structure, the Controller, finally connects the data and the calculation routines of the Model with its representation in the Viewer. Thereby the View directly forwards its user inputs, and the Controller either directly provides requested data or initiates calculations inside the Model. After they have finished, it notifies the GUI. The Controller also contains the Control Thread as described in the next section.

2.2 Concurrency

The second major software architecture used in this tool is Concurrency. It allows the software to run parallel processes. Therefore, different parts of the software can be executed at the same time, ideally on different Central Processing Unit (CPU) cores. This technique can highly increase the performance of an application depending on the amount of processing that can be involved efficiently. Theoretically, a proportional performance increase could be possible, but Concurrency often leads to some inefficiencies as well. [5]
In order to integrate Concurrency, the software must first be analysed for mutually independent calculations. Often algorithms are sequential and therefore can not be parallelised. Secondly, access to shared data must be treated carefully and often protected through exclusion methods like mutex. The problem arises when individual processes (from now on called **threads**) try to access the same data simultaneously, which usually leads to undefined behaviour. Therefore the integration of multi-threading must be planned during the architectural phase, and the affected modules are designed accordingly.
In Qt, the View's nature requires its own thread, which is from now on called the GUI thread. It has to handle mouse and keyboard events, changing between different windows, interacting with plots or rendering the 3D view. During the initialisation of the window, the application's main thread becomes the GUI thread. The performance issues occur when a calculation is so computationally expensive that it requires a by the user noticeable amount of time. If such a demanding process is done by the GUI thread, the GUI would completely freeze and not respond to any input. To prevent this, a second thread, the so-called control thread, is introduced. It is running in a loop inside the Controller. Every time the View initiates a calculation, its processing is taken over by the control thread, and the result is then reported back. Its tasks contain

- all routines of the SatNOGS loader and decryption process,
- and all calculations of the Space model. This also contains the real-time satellite propagation for orbit visualisation.

In chapter 5 it will be analysed if the described concurrency approach was sufficient to provide a responsible GUI.

2.3 Best Practices

Additionally, to the software architecture, it is advisable to introduce some best practices for a project. Their goal is to create a consistent code style and methodology. They are usually language or even framework-specific and are often based on personal experience. [10] Commonly used styling guidelines for example contain `CamelCase` or `snake_case`, which are both used in the code for different purposes. Another utilized style aspect is the `i_` input prefix for function parameters.
However, code styling should not be the main topic of this section. Instead, it is concerned with defining a general approach to connecting function calls between the Model, Controller and View. Such an approach should maintain the flexibility between layers introduced by the MVC while making them testable and their components reusable. The solution was taken from Peter Winston's (chief executive officer (CEO) of Integrated Computer Solutions (ICS)) talk "Lessons Learned from Building 100+ Devices with C++/Qt/QML" at the Qt Summer Summit 2020 [58].

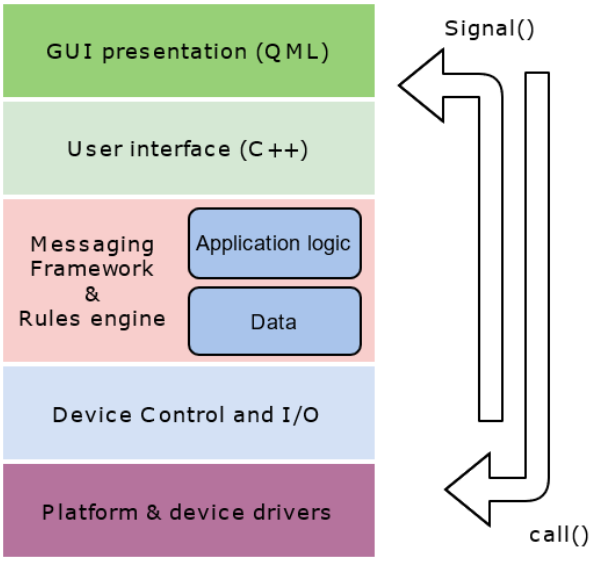

Figure 2.3: GUI Interconnections from Peter Winston's talk "Lessons Learned from Building 100+ Devices with C++/Qt/QML" [58]

In his approach, he defines strict rules on how to communicate between different layers. Therefore he limits the connection on the descend from the GUI to lower layers to direct method calls. This means the View must directly call methods in the Controller, and the Controller does it similar in the Model. For the other direction, Qt's Publisher-Subscriber (Pub-Sub) pattern is used, and reaction in a layer above is invoked via a signal. Figure 2.3 visualizes this method. In this image, the "GUI presentation (QML)" and the "user interface (C++)" correlate to the View, the "application logic" to the Controller and the data and parts of the Input Output (IO) to the Model. This method, in general, puts a bigger emphasis on the lower layers. Since its upwards communications only take place over signals, the upwards directions is fully abstracted. Therefore the approach makes modifications and extension of the respective upper layer even

in later phases of the software development relatively simple. The basis of this approach is the expectation that modifications of higher layers are more common than in lower.
In Peter Winston's Qt user-interface principles, it is also mentioned to not put application logic into QML components and to, in general, use more C++ than QML. Both principles were used during the code implementation.

2.4 UML Schematics

For documentation purposes, a Unified Modeling Language (UML) diagram of the whole project implementation was created. It is attached to this work in fig. A.1, fig. A.2 and fig. A.3. All C++ classes are complete with methods, variables, signals and slots. The representation of the QML classes where reduced, to preserve clarity.

3 Model: SatNOGS Data Client

In this section, the SatNOGS Data Client is implemented as the first part of the Model. It carries out the download of observations and transmission data from the SatNOGS Network. Both are defined in section 1.1.2. Afterwards, the transmissions are decoded, using the AX.25 frame and SOMP 2b beacon (see section 1.1.3).

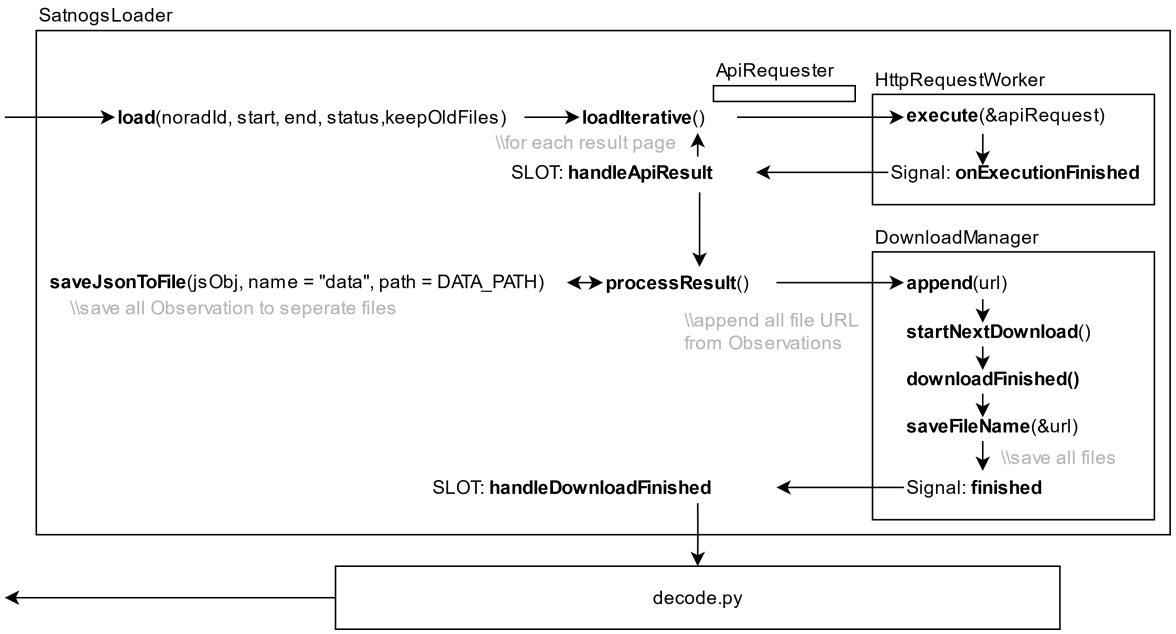

Figure 3.1: Schematic Overview of the Data Client

A schematic representation of the fully implemented algorithm is shown in fig. 3.1. This overview does not claim to be comprehensive and is only intended to provide a better understanding. It displays the separation of the SatNOGS data client into the `SatnogsLoader`, which contains the REST client, and the decode.py script, which contains the decryption routine.
The following sections describe the implementation of these two parts and the decisions during their creation process.

3.1 REST Client

3.1.1 REST Client: Variant Comparison

To develop a client for the SatNOGS Network API a REST client is required. Initially, an online investigation was carried out to find existing REST implementations. Thereby, four possible solutions could be identified.
The first is a python tool specifically developed to interact with the SatNOGS API. The second is the official Microsoft C++ REST Software Development Kit (SDK). The third is another C++

implementation contained in the popular Boost library. The final solution is represented by a proprietary development using the tools Qt provides.

	Glouton SatNOGS [14]	Microsoft REST SDK [29]	Boost REST Library [6]	Proprietary
Programming Language	- - (Python)	+ (C++)	+ (C++)	++ (C++/Qt)
Complexity	+	- -	- -	+
Implementation Effort	-	+	+	-
Modifiable	+	-	-	+
Trustworthy Developer/ requires Validation	-	+	+	-
	-1	0	0	2

Table 3.1: Variant Comparison: REST Client

Through comparison, the best-suited approach for the SOMP 2b application was determined. Table 3.1 shows that a proprietary solution provides the most advantages. Since it is written in a combination of C++ and Qt, no datatype conversion is required between the REST module and the rest of the software. Besides this issue, a possible Python implementation would further have the disadvantage of requiring an installed Python runtime environment since embedding it is not feasible for such a complex module. Even though the Microsoft REST SDK and the Boost library are already fully implemented and verified compared to a proprietary solution, their complexity makes them more challenging to integrate and modify. In the end, it was assessed that the excellent modification and integration properties of the proprietary solution outweigh the efforts of its from ground-up implementation, and it, therefore, represents a better solution than its competitors.

3.1.2 REST Client: Implementation

In this section the implementation of the proprietary REST Client is explained. Thereby fig. 3.1 can be used for visualization.
The API request is initiated by the Controller via the `load(noradId, start, end, status, keepOldFiles)` function of the `SatnogsLoader` class. The start and end attribute represent the observation timespan, and the `keepOldFiles` boolean decides whether all existing files in the `/download` folder are removed or not. It than calls the `loadIterative()` function, which iterates until all response pages of the SatNOGS network are acquired. Thereby the `ApiRequester` is used to built the REST API URLs, which are subsequently provided to the `HttpRequestWorker` class, that eventually carries out the download procedure. After it has acquired a response it is decided whether the request is complete or further pages need to be downloaded through the `loadIterative()` function. An example REST API URL is shown below (it is explained in section 1.1.2):

```
https://network.satnogs.org/api/observations/?end=2021-4-29T20%3A12%3A33&
    page=1&satellite__norad_cat_id=47445&start=2021-04-28T00%3A00%3A00&
    status=good
```

The result of these requests is assembled to a long JSON construct, which is than separated into its individual observation objects by the `processResult()` function. These are afterwards saved as individual files, that are placed inside the local `/download` path under the following name convention:

```
observation_noradID.json
example: observation_3434352.json
```

The downloaded observations contain links to multiple additional files, like transmission binary data, audio files, and waterfalls. These URLs (directing to waterfalls and transmissions) are presented to the `DownloadManager`, which subsequently downloads the files and saves them in the `/download` folder. The name convention for transmission

```
data_noradID_yyyy-mm-ddThh-MM-ss
example: data_3434352_2021-01-09T01-41-49
```

and waterfall are provided as well.

```
waterfall_noradID_yyyy-mm-ddThh-MM-ss.png
example: waterfall_3434352_2021-01-09T01-38-34.png
```

After all, downloads have finished, the python decryption script is started.

3.1.3 REST Client: Verification

To check the accuracy of the REST client, a small verification was performed. Therefore the results of an SatNOGS network query were compared to the downloaded data of the implemented REST Client.
The results of the online query [51] are shown in fig. 3.2. Between the 2021-05-18 00:00 and 12:00 the SatNOGS network observed the SOMP 2b satellite two times. While the first observation did not receive any transmissions the second one (ID 4125933) received four.
Figure 3.3 displays the results of the REST implementation. It consists of two observations with one waterfall each and four transmission data. Therefore both results agree.

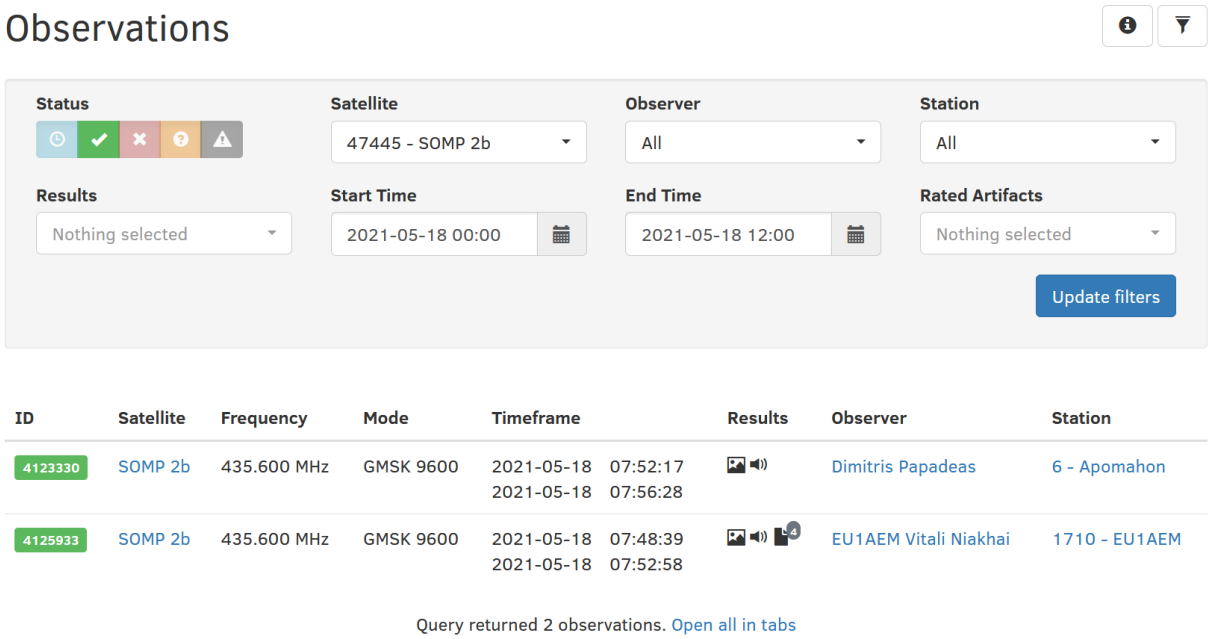

Figure 3.2: REST Client Verification SatNOGS data [51]

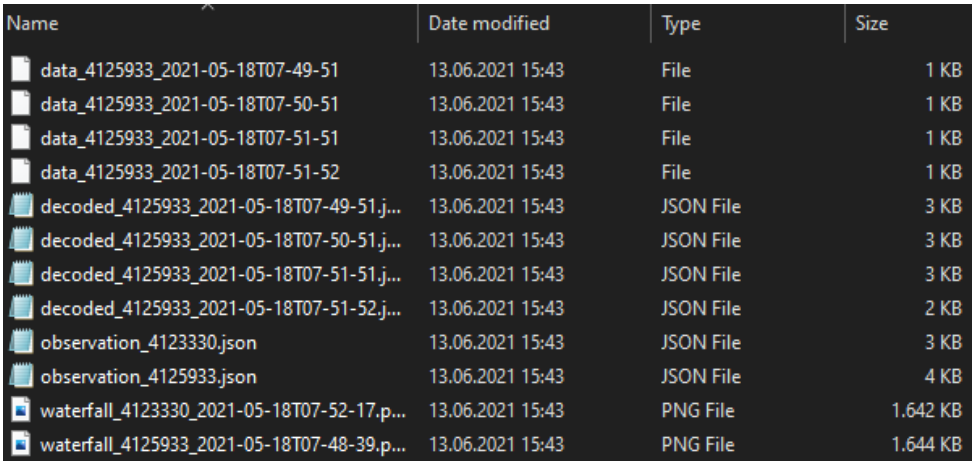

Figure 3.3: REST Client Verification: Client data

3.2 Data Decoder

The second part of the Data Client is the decoder. It takes the raw transmission files downloaded from the first part and decrypts them into JSON files. These and their corresponding observation files are then imported by the Model and stored into lists to provide their information to all software modules. In these lists, each value has a corresponding date that was extracted from the SatNOGS filename. Therefore a small-time deviating to the spacecraft's internal clock might occur.

The downloaded transmission files contain binary information in the hexadecimal presentation (see fig. 3.4). To convert them to the human readable JSON format the SOMP 2b Beacon Definition (appendix B.1) must be used.

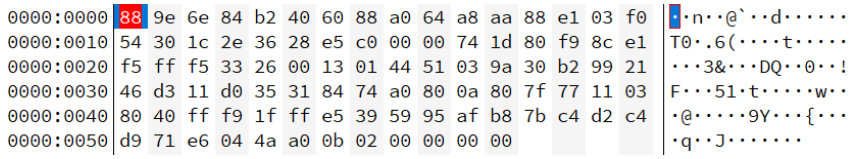

Figure 3.4: Example hexadecimal Transmission of SOMP 2b

3.2.1 Data Decoder: Variant Comparison

Before the decoder can be implemented, its architecture must be decided. The most significant influence on its structure comes from the characteristic that if the transmission is altered or the tool is used for a different satellite, only the data decoder needs to be adjusted to reestablish the tool's full functionality. Therefore its architecture must offer exceptional modifiability.

	Python	C++
Implementation Effort	- -	+
Language Performance	0	+
Compilation Effort	+	- -
Modifiability	+	-
	0	-1

Table 3.2: Variant Comparison: Data Decoder Language

For this reason, it is being considered to implement the Data Decoder in Python instead of C++. Since Python is a scripting language and does not require compilation, the decoder would be human-readable and easily editable. Pythons most significant advantage is that it does not require the compilation toolchain and the built environment of C++/Qt. Typically, it only requires a runtime environment installed on the executing system to interpret its scripts. This downside could be mitigated by embedding Python in C++. Therefore its interpreter is integrated into the applications build, and thus it can execute python scripts without the need for a runtime environment. This type of implementation has its disadvantages. Thus, it is a much more complex solution than a pure C++ integration, and in addition, the integration of libraries into embedded Python is quite limited.
The trade-offs between C++ and Python are weighted and compared in table 3.2. It shows that a scripting language has a slight advantage over C++ for this purpose and therefore is used to implement the Data Decoder.

3.2.2 Data Decoder: Implementation

In this section the implementation of the Data Decoder "decode.py" (see fig. 3.1) is described. To decode SOMP 2b transmissions, the SOMP team already has two implementations of their Beacon Definition. The first one is programmed in C++ and is used in the ground command software. A second one, coded in Python, was developed as integration into the SatNOGS

Dashboard. The latter one utilizes the Kaitai Struct library [21], a language-independent binary data parser. It uses descriptive "YAML Ain't Markup Language" (YAML) files to define values that should be converted, their conversion and individual short descriptions. These files are then compiled into extensions for different programming languages (C++, Python, Java and many more) to read and write binary data. Since this method is used for the SatNOGS dashboard and the YAML file (from now on called Kaitai struct) has already been created by SOMP team, it is also used in the data decoder. The Kaitai struct used during this work is provided in appendix D.4.
In detail, the Data Decoder iterates over all binary transmission files, which are placed in the download path, convert them via the Kaitai struct and saves their result individually as JSON files. To provide more meaningful identifiers than the Kaitai in-code variable names (like `beacon_on_time`), the script tries to find names inside the documentation. Thereby, the Kaitai structure checks for the presence of an `doc:` entry two lines below the variable definition. For the example, it would replace the previously mentioned variable name `beacon_on_time` with `OBC Time Stamp [s]`, thereby also adding the unit in square braces. If no documentation is found, the variable name is used, and an empty bracket added. This implementation has one disadvantage, it requires unique variable names between multiple beacons in the Kaitai struct. The file naming convention contains the NORAD ID and the date, similar to the SatNOGS client.

```
decoded_noradID_yyyy-mm-ddThh-MM-ss.json
example: decoded_3434352_2021-01-09T01-38-34.json
```

3.2.3 Data Decoder: Verification

After the Data Decoder and the visualisation of its data as graphs have been fully implemented, a verification was conducted. Thereby the displayed values were compared with transmissions that were decrypted using the original Kaitai struct provided by the SOMP team using the Kaitai online integrated development environment (IDE). As test data the SOMP 2b transmission on the 18.05.2021 at 07:51:52 from SatNOGS was used.
The comparison results is shown in appendix C.1. All values match up and identify a correct implementation of both the Data Decoder and visualisation process.

4 Model: Space Model

4.1 General

In this chapter, the "Space Model" is discussed and implemented. In general, it provides calculation routines for the orientation of the earth and its relative position to the sun. This part of the model is hereafter called the "Earth Mode" and is described in section 4.2. Afterwards, the orbit model for satellites combined with eclipse and illumination state prediction is implemented in section 4.3.
At the beginning of this chapter, general topics essential to both models are introduced, such as NASA NAIF's SPICE and orientation representation systems.

4.1.1 NAIF SPICE

In this section the functionality of NAIF Spacecraft, Planet, Instrument, Orientation ("C-matrix") and Events (SPICE) toolkit is explained. In the later sections, it will be individually discussed why SPICE is advantageous compared to other models/methods. However, since it will be used in both, it is introduced in advance.
SPICE is developed by NASA NAIF to support the work of their scientist and engineers. It is built around so-called "kernel" files. They usually contain navigation and other ancillary data that is required to provide precision geometry of space objects. These kernels must be structured according to the SPICE standards, which for example means, that they must contain metadata. Many of them are directly produced by NASA and provided for the public. [34]
The kernels defined by SPICE are listed below: [34]

- The Spacecraft and Planet Kernel (SPK) contains ephemeris of different space objects as a function of time.
- Planetary Constants Kernel (PCK) include constants about target bodies, like shape and size specification.
- The Instrument Kernel (IK) provides data about the geometric aspects of scientific instruments. Typical ones are the orientation, shape and size of the field of view.
- The orientation angles and rates of spacecraft structures are contained in the C-Matrix Kernel (CK).
- The rarely used Event Kernel (EK) provides science plans, procedure sequences and notes.
- The Frames Kernel (FK) defines typical reference frames.
- The Leap Second Kernel (LSK) and Spacecraft Clock Kernel (SCLK) are used to convert timestamps between various measurement systems.

- Lastly, the Digital Shape Model Kernel (DSK) provides high fidelity shape models and can replace parts of the PCK.

To create, import, and utilize these kernels, NAIF provides the SPICE toolkit. This toolkit is an extensive open-source collection of APIs and its underlying functions and subroutines. They are built together into a well documented, ready-to-use library. Additionally, a set of pre-built programs is included. Many of them are intended to manipulate or inspect kernels. The toolkit was initially created using FORTRAN but later became available for C, IDL, Matlab and Java.[34] The C version of the library was compiled with an older version of the Microsoft Visual C++ (MSVC) compiler. Therefore it was required to be recompiled with the MSVC 2019.
All kernels used during this work were official releases by NASA and are available on their NAIF website. Each kernel comes as an individual file and is stored in the `\kernels` folder. To avoid multiple imports, a so-called "Meta-Kernel" is utilized to combine the name and path of all required kernels.

4.1.2 Comparison: Euler Angle, Quaternion and Rotation Matrix

In order to represent and calculate rotational states, an orientation representation system is necessary. Typical ones are Euler Angles, Quaternions, or Rotation Matrices. To identify the most suitable, a variant comparison was carried out (see table 4.1).

	Quaternions	Euler Angles	Rotation Matrix
Calculation Performance	+ +	+	0
Complexity	-	+	-
Amount of Values	0	+	-
Gimbal Lock	+	–	+
	2	1	-1

Table 4.1: Variant Comparison: Orientation System

Quaternions are a widely used system in computer graphics. Their mathematics is based on an extension of complex numbers. This addition results in a fourth dimension and makes quaternions particularly difficult to read by a human. Nevertheless, they provided the best computational performance in comparison to the two other orientation systems. Additionally, they do not suffer under the gimbal lock (the loss of a degree of freedom when two rotation axes align) like Euler Angles and are only based on four values compared to the nine values of a rotation matrix. [26]
The low amount of values make them incredibly efficient to transmit from the Controller to the QML environment of the View. In addition, the abstinence of the gimbal lock makes their handling much less problematic when a wide variety of orientation states are present, such as in space. Therefore quaternions are used throughout this work.

4.2 Earth Model

In this section, a model is chosen and implemented to simulate the earth's motion around the sun. It additionally needs to predict the spacecraft's eclipse periods, i.e. when the earth is between the sun and the satellite. Furthermore, it must provide the rotational state of the earth.

4.2.1 HORIZONS vs JPLs Approximation vs Naif CSpice

Different model approaches are available to predict the earth's orbit, ranging from simplified two-body variants over complex n-body models, including time relativity, solar pressure and precision movement, up to pure mathematical approximation formulae. Concerning their usage, these models mainly differ in their long-term accuracy, computational cost, implementation effort, and the number of solar system objects they contain. The Solar System Dynamics department of JPL does provide two different solutions with their HORIZONS system and an orbit approximation (see table 4.2). Additionally, JPL NAIF provides SPICE (see section 4.1.1). To not exceed the scope of this chapter, no further systems will be discussed. [13]

	HORIZONS [16]	JPL's Position Approximation [23]
Solar-system objects	> 10^6 bodies (planets, asteroids, comets, satellites etc.)	9 planets [22]
Accuracy	depends highly on the selected object, for major planets 10cm to 100+ km for asteroids in general > 1 arcsec [17]	for the earth-moon barycenter timespan from 1800 - 2050 $dRA = 20''$ $dDec = 8''$ $dr = 6000km$
Implementation	requires regular API requests to receive up to date ephemerides	based on a small set of equation, Keplerian elements and their rates [22]

Table 4.2: Comparison: JPL's Solar System Models

HORIZONS is internally based on SPICE kernels. Therefore, when the same kernels are used, their accuracy should be similar. The specific HORIZONS SPK (DE432) is together with all official ones available in the NAIF archive [35]. A general downside of the usage of these kernels is their size. Depending on the period, an SPK can take up to 1.7 Gb, for example.
Nevertheless, the SPICE toolset seems to be the best solution for the SOMP 2b tool. Compared to the position approximation, it also contains an orientation model for its objects and is additionally far more accurate. The HORIZON tool can be a good alternative when the program's size must be limited, or no SPICE toolset is available for a given programming language. However, its online dependency makes this approach inflexible and would restrict the usage of the SOMP 2b tool. Therefore the SPICE toolset is used as a model for the earth's position and rotation.

4.2.2 Earth Model: Implementation

The implementation of the Earth Model takes place in the class `SpaceModel`. Since the SPICE library is utilized, only two small methods are required.
The first:

```
Vector3D getSunEphemeris(const QDateTime &i_date);
```

calculates the sun ephemeris relative to the earth. It uses the following SPICE call to receive $\vec{r}_{earth,sun,J2000}$ the position of the sun relative to the earth in the J2000 frame:

```
spkezr_c ( "SUN", et, "J2000", "NONE", "EARTH", state, &lt );
```

The data required by this function is provided by the DE430 SPK. It persists of planet (and lunar) ephemeris from the 01.01.1550 to the 22.01.2650 and therefore is relatively small with only 120 Mb. The direction of relation was selected for an easier implementation in the View, since the ECI is its root frame.
The second method:

```
QQuaternion getEarthRotation(const QDateTime &i_date);
```

provides the conversion quaternion between the ECI and ECEF frame, through the SPICE function:

```
pxform_c ( "J2000", "ITRF93", et, mat );
```

Even though SPICE has a range of build-in reference frames, which therefore don't require any kernel, the integrated earth ECI frame `IAU_EARTH` is not suitable for high-accuracy applications and thus an additional FK is required. NAIF suggests to use the `ITRF93` kernel instead. Utilizing it requires more additions, therefore the PCK `pck00010.tpc` for earth radii and the earth binary PCK `earth_200101_990628_predict.bpc` (from 2000 to 2099) are further included. Lastly a LSK is essential for every time depended calculation. This data is provided by the `naif0012.tls` file. [33]
The second part of this section gives a short overview of the implementation of the earth model in the View. As mentioned before, the 3D root object coincides with the ECI frame. It further contains the object of the sun, the frame of the satellite and the ECEF frame, each with its own rotation and translation. During the implementation of the View, a problem with the Qt3d implementation was noticed. When the general (x, y, z) 3D vector arrangement with an up-vector of $(0, 0, 1)$ is used, Qt's `FirstPersonCameraController` produced a strange behaviour when rotating the camera. To solve this problem the vector arrangement was altered to $(x, z, -y)$ with an up-vector of $(0, 1, 0)$. The transition takes only place inside the QML modules but also compromises quaternions.

4.2.3 Earth Model: Verification

To verify the implemented earth model, its output is compared with an existing space model application. Therefore NAIF's with SPICE enhanced version of Cosmographia [31] is used. To test all features of the model simultaneously, a time near the 2021 spring equinox is chosen (20.03.2021 09:37 [57]), specifically, the 20.03.2021 12:07:43. At this date, earth ECI and ECEF frame are roughly identical, and the suns position vector is approximately identically with both x-axes. This is shown in fig. 4.1.

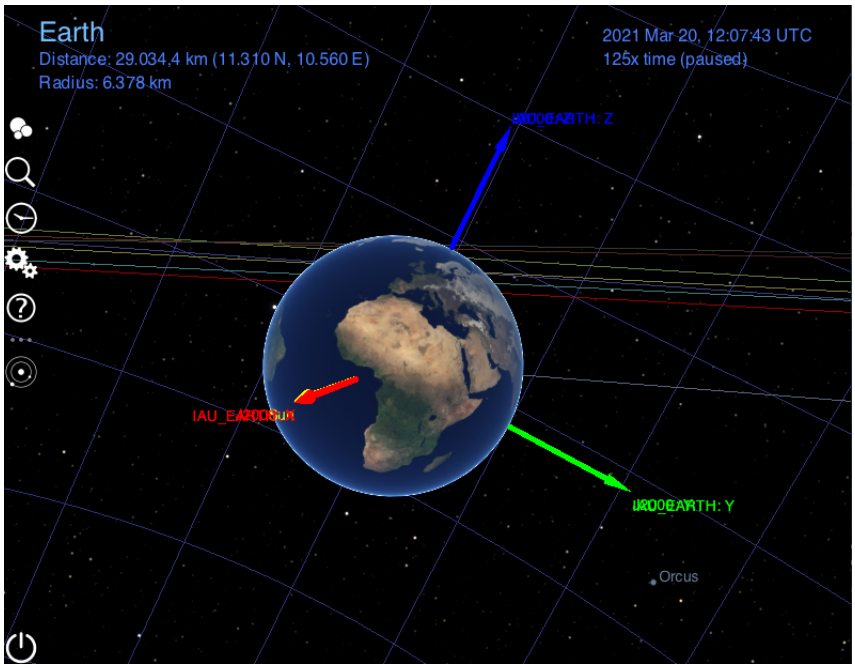

Figure 4.1: Space Model Verification: Cosmographia [31]

The occurrence of the same correlation in the implemented model would indicate a correct implementation. Even so, no conclusion about its accuracy can be drawn. For comparison the same constellation is shown in fig. 4.2.
To represent the sun vector, the size and distance of the sun were scaled down to make it visible in front of the earth. The verification, in its scope, shows a correct match between NASA implementation and the one of the SOMP 2b tool.

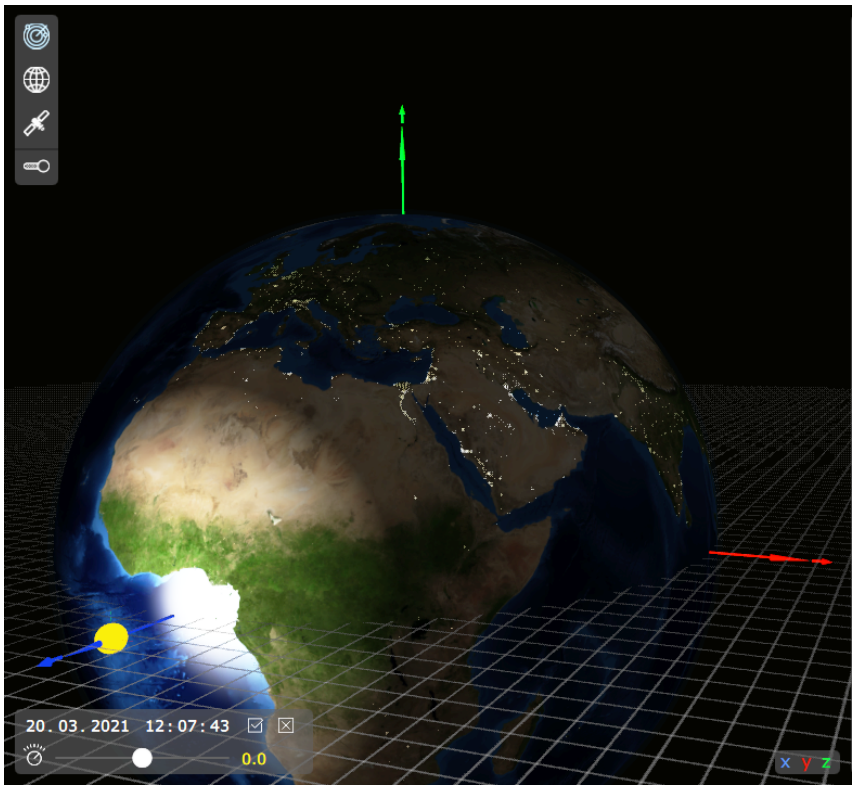

Figure 4.2: Space Model Verification: SOMP 2b Orbit Visualization

4.3 Satellite Model

In addition to the earth model, the satellite model must provide a method to predict the satellites position and orientation. Furthermore, these predictions must be used to predict passes over a selectable position and the illumination state.

4.3.1 SGP4 vs NAIF SPICE

In section 1.1.4 it is described, that the SGP4 model should be used, when only TLE sets are available. However, the NAIF SPICE toolset, which was introduced in this chapter, uses SPK ephemeris data to determine the position of an object in space at a given time.
Nevertheless, it would be beneficial for the integrity of the application to use a unified model API compared to integrating another 3rd party library or the complexity of in-house development. Therefore it is necessary to convert TLE sets into SPICE SPK files. Unfortunately, the SGP4 algorithm that was present in the original FORTRAN version was not made available with the C version of SPICE. In this version, one is supposed to use the MKSPK application, a pre-compiled tool included in the toolset. It provides a wide range of functions to convert different forms of input data into SPK. Among these, there is also one for TLE data available. While in the MKSPK documentation, it is only written that TLE data is treated as a special case, an analysis of the applications source code shows that the SGP4 model is used internally. [32] In summary, it can be concluded that SPICE is not used instead of SGP4, but that both are used together.

4.3.2 Satellite Model: Implementation

The implementation of the satellite model takes place in the `Spacecraft` class. When the position of the spacecraft and thereby its visual position in the View requires an update, the Controller calls the following method:

```
void calcSpacecraftDate(const QDateTime &i_datetime);
```

It calls the SPICE API to receive the position of the satellit (SPICE ID: -48) relative to the earth in the J2000 frame with the following command:

```
spkezr_c ( -48, etSpacecraft, "J2000", "NONE", "EARTH", stateSpacecraft,
    &lt );
```

Afterwards, it calculates the orientation like described in the next code snippet. Thereby `etSpacecraft` contains the currently displayed TDB time and `etInitial` the time of the reference transmission.

```
double dAngularRateX = (etSpacecraft - etInitial) * angularRateX;
double dAngularRateY = (etSpacecraft - etInitial) * angularRateY;
double dAngularRateZ = (etSpacecraft - etInitial) * angularRateZ;

orientation = QQuaternion(quatW,quatX,quatY,quatZ).normalized()
                * QQuaternion::fromEulerAngles(dAngularRateX, 0, 0)
                * QQuaternion::fromEulerAngles(0,0,dAngularRateY)
                * QQuaternion::fromEulerAngles(0, dAngularRateZ,0);
```

The quaternions for the yaw-, pitch-, roll-axis were multiplied separately to preserve the original order of rotation. `quatW, quatX, quatY` and `quatZ` are the variables, which need to be provided by the satellite transmissions and represent the orientation at the instance of transmission. The normalization step is necessary because of floating-point approximations or missing normalization on the spacecraft's side.
The required SPK are provided through the `Model` class. Each time it imports new observation and decoded transmission data, it collects all TLE lines, writes them to a file and runs the MKSPK application to generate the SPK for the satellite. Afterwards, it imports this kernel and thereby provides its data to the routines in the `Spacecraft` class.
On the other hand, for the quaternion data, the Controller checks each time before the spacecraft is updated if orientation data are available. If so, it updates the values before the new orientation is calculated.
The `Spacecraft` class provides a similar method for the calculation of the orbit markers. They are used to visualize one orbit in 3D through orange spheres.

4.3.3 Pass Prediction

The pass prediction is implemented in the function `predictPasses()` of the class `Spacecraft`. Through the pass prediction window in the GUI (see fig. 4.3), the user can provide all necessary data for the calculation. The Controller then passes this information over to the model.

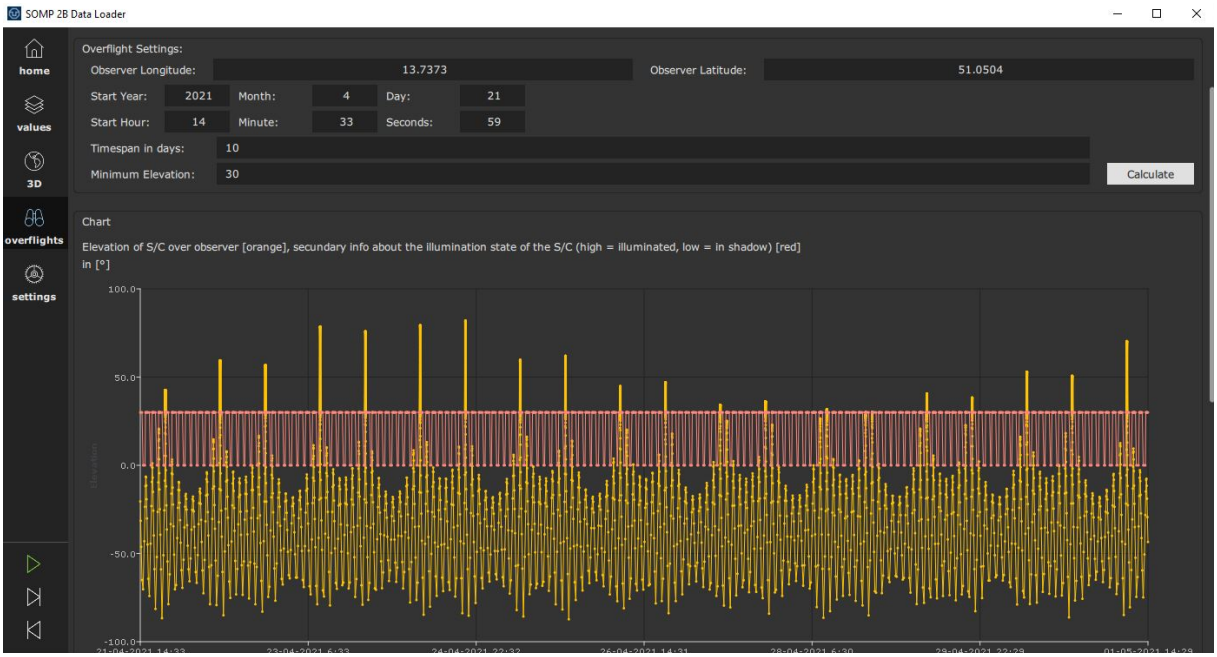

Figure 4.3: SOMP 2b Tool: Pass Prediction Window

For the pass calculation, the prediction routine iterates over the period given by the user. In this loop, it starts with calling the SPICE API to receive the vector $\vec{r}_{observer,satellite,ECEF}$ from the observer position to the spacecraft in the ECEF frame. This is done by the line below.

```
spkcpo_c(sat_spice_id, et_start + dt, "ITRF93" , "OBSERVER" , "NONE",
    position_obs , "EARTH", "ITRF93" , state , &lt);
```

Afterwards the calculation shown in eq. (4.1) is conducted.

$$\alpha_{observer} = 90° - \arccos \frac{\vec{r}_{observer,satellite,ECEF} \cdot \vec{r}_{observer,ECEF}}{\left\|\vec{r}_{observer,satellite,ECEF}\right\| \cdot \left\|\vec{r}_{observer,ECEF}\right\|} \tag{4.1}$$

It calculates the angle between the observer position vector and the observer-spacecraft vector. The subtraction from $90°$ is necessary to represent the elevation above the surface instead of the angle to the zenith. If the elevation is higher than the selected minimum elevation $\alpha_{observer;min}$ a pass is detected.

The algorithms timestep is adjusted to provide a one-second accuracy for elevation start and end times. Nevertheless, running the whole prediction procedure for every second for a possibly multiple month-long prediction period would be extremely inefficient. Therefore the timestep is increased depending on the difference between the last predicted elevation and the required

pass elevation. It is only increased when the result of the difference is negative. The following equation is used:

$$dt = 1s + (-\alpha_{observer}/° + \alpha_{observer;min}/°)^{1.5}s \tag{4.2}$$

All generated data by the pass prediction routine is exported as comma-separated values (CSV) file to the `\output` folder. These files contain the date and time of each iteration, the ECEF position of the satellite, the elevation and illumination state. A second CSV file is available to provide the start and end date of a pass, its max elevation and the exact date when it occurs.

4.3.4 Illumination Model

The logic of the illumination state calculation is contained in the following method.

```
bool predictIlluminated(const QDateTime &i_dt);
```

Its execution is part of the pass prediction and is called in each of its loops. The Prediction is based on eq. (4.3).

$$r_{earth-sun-direction,satellite,ECI} = \left\| \vec{r}_{satellite,ECI} - \frac{\vec{r}_{sun,ECI}}{\left\|\vec{r}_{sun,ECI}\right\|} * \left(\vec{r}_{satellite,ECI} \cdot \frac{\vec{r}_{sun,ECI}}{\left\|\vec{r}_{sun,ECI}\right\|} \right) \right\| \tag{4.3}$$

The dot product multiplied by the normalized sun vector provides the point along the direction of the sun vector, perpendicular to the satellite's position. The amount of the difference between this vector and the satellite's position provides the shortest radius of the satellite's location to the direction between the sun and earth. If this radius is smaller than the earth's radius, the planet may be covering the satellite. In this calculation, the angle of the sunlight ($2.4° * 10^{-3}$) is neglected because of its minor influence. To predict if the spacecraft is in front or behind the earth, the sign of the dot product is analyzed. In case it is positive, the satellite is between sun and earth, and if it is negative, the satellite is behind the earth.
Therefore the `predictIlluminated` function can prognosticate the satellites eclipse when the radius of the satellite is lower than the radius of the earth and when it is additionally behind the earth relative to the sun.

4.3.5 Satellite Model: Verification

To verify the Satellite Model, a pass prediction at the 30.05.2021 from 18:43 until 19:35 for the SOMP 2b was carried out. A minimum elevation of $-180°$ was selected to have a prediction for every second, and as observer position, Dresden was chosen. To provide TLE data, all observations from the 29.5 were downloaded. The result is shown in fig. 4.4.

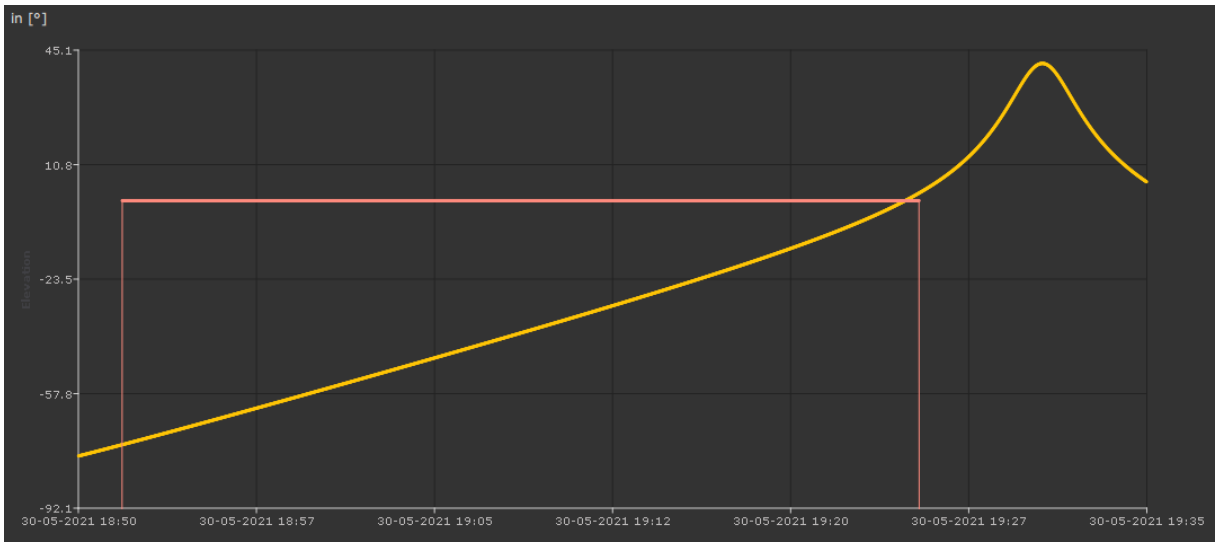

Figure 4.4: Satellite Model Verification: Pass and Illumination Prediction at the 30.05.2021 from 18:43 until 19:35, Minimum Elevation -180 °

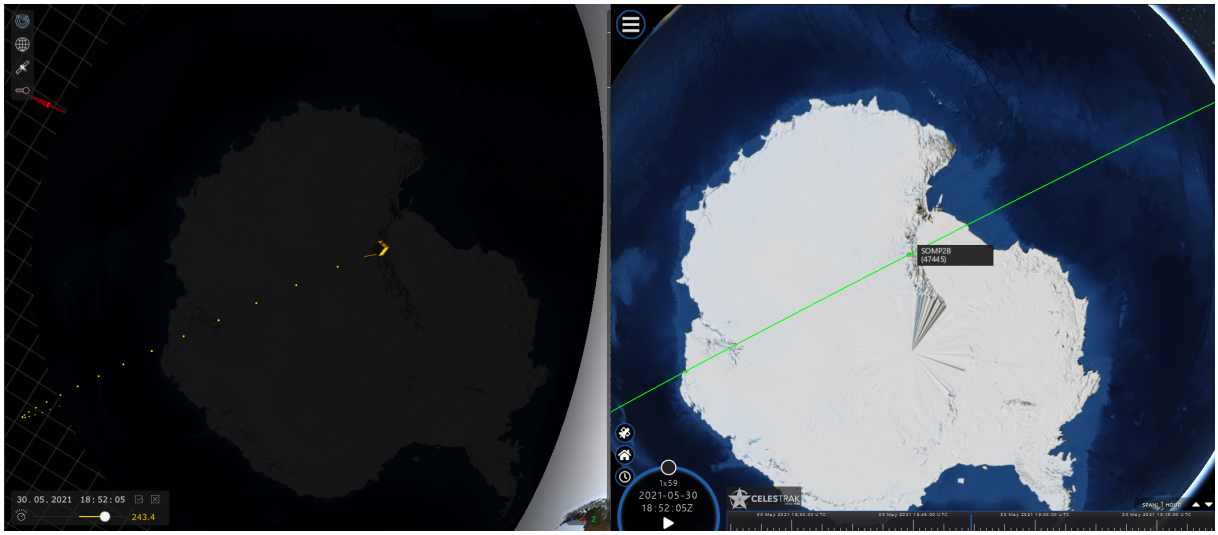

Figure 4.5: Satellite Model Verification: Comparison 1 to CelesTrak 30.05.2021 18:52:05

It shows that exactly one eclipse is taking place during this time period, and directly afterwards, the satellite's elevation has a maximum. In the next steps, the start of the eclipse (fig. 4.5), its end (fig. 4.5) and the maximum (fig. 4.7) are analyzed. Therefore the 3D orbit visualization of the SOMP 2b tool is compared to the official CalesTrak visualization [8].
Since both visualizations are 3D, there might be slight differences in the camera position, which result in visual deviations. Nevertheless, all three comparisons show identical satellite position and indistinguishable orbits.
In comparison three, the position of SOMP 2b seems to be in the northeast of Poland with an approximate air distance to Dresden of $570km$. Together with SOMPs average height ($535km$) this would result in an estimated elevation of $\approx 43°$. A similar value can be seen in fig. 4.4.

Figure 4.6: Satellite Model Verification: Comparison 2 to CelesTrak 30.05.2021 19:25:13

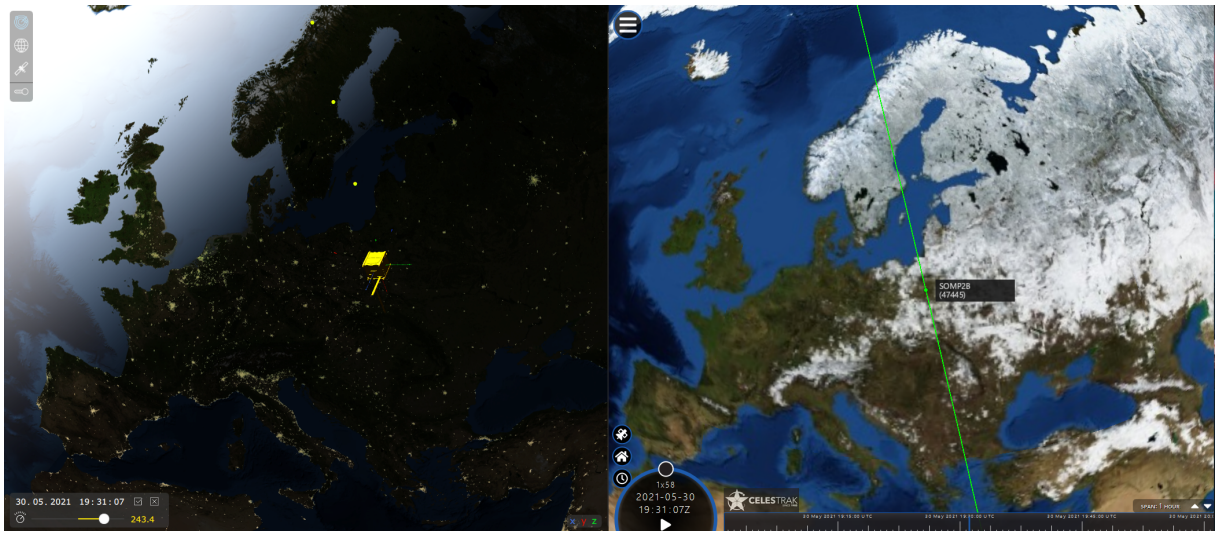

Figure 4.7: Satellite Model Verification: Comparison 3 to CelesTrak 30.05.2021 19:31:07

In terms of the illumination prediction, it is visible that the satellite just entered the earth's shadow in comparison one. In fig. 4.6 its flight direction is toward the end of the shadow. At this point, it needs to be mentioned that the 3D rendering does not contain the calculation of shadow maps. Consequently, the earth does not throw any shadow on the satellite in the 3D View.

To provide an additional visualization of the satellite entering the shadow, fig. 4.8 shows comparison one in a different perspective. From the corona around the earth, the viewer can estimate the sun's position to the top right and visualize how the satellite enters the shadow region behind the earth.

Lastly, a basic verification for the satellite orientation model is established. Therefore a decoded transmission file is altered to contain $\texttt{quatX}, \texttt{quatY}, \texttt{quatZ} = 0$ and $\texttt{quatW} = 1$. Thus the satellites frame aligns with earths ECI frame at the instance of the transmission. Furthermore,

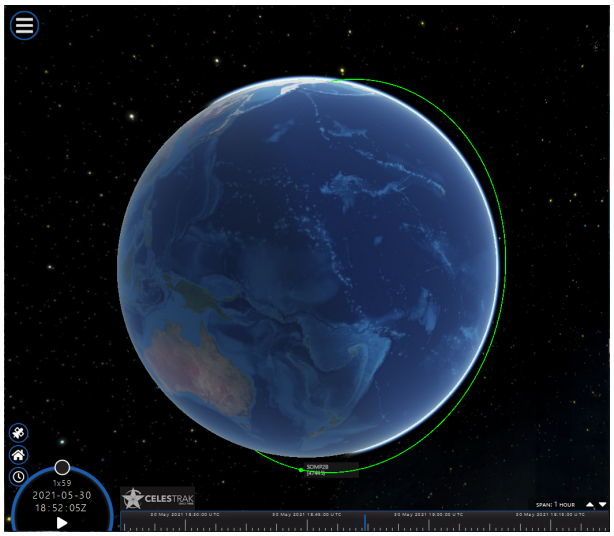

Figure 4.8: Satellite Model Verification: Comparison 1 to CelesTrak Sun Angle 30.05.2021 18:52:05

the values for `angular_rate_x`, `angular_rate_y` and `angular_rate_z` are modified to the following:

$$\dot{\gamma} = 1°/s; \dot{\beta} = 2°/s; \dot{\Psi} = 4°/s \tag{4.4}$$

These data were then loaded into the SOMP tool and propagated for $10s$. The result is shown in fig. 4.9. According to the rotation order, the expected result is displayed (the x-axis first, then y and finally the z-axis).

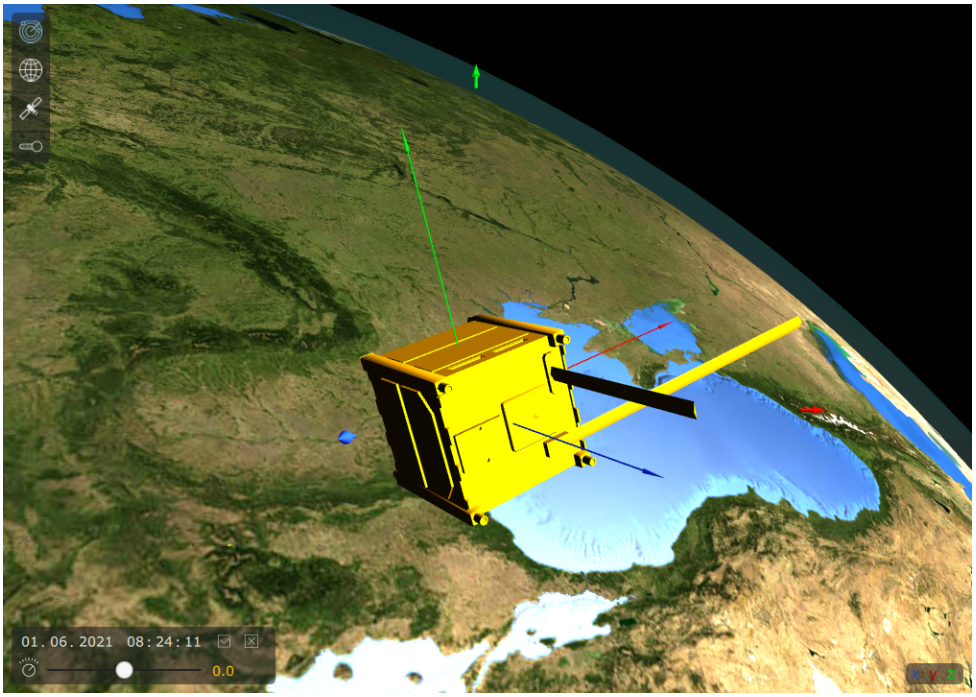

Figure 4.9: Satellite Model Verification: Orientation after $10s$ from ECI alignment with $\dot{\gamma} = 1°/s$, $\dot{\beta} = 2°/s$ and $\dot{\Psi} = 4°/s$

In general, these verifications indicate a correct implementation, but they do not provide any estimation of the model's accuracy.

5 Results and Discussion

In this chapter, the result of this work is compared to the initial goals and requirements defined in section 1.1. Thereby The different decisions that have been made are evaluated.
One of the first targets that the SOMP tool is required to archive is the visualization of all kinds of data transmitted by the satellite. This was full-filled through a generic interactive graphs implementation on the one hand. And a detailed 3D rendering of the earth, the satellite and its orbit on the other. For these visualisations, it is not possible to quantitatively measure the quality compared to the requirements. In general, it can only be said that it can successfully display SatNOGS data and exporting them as JSON and CSV files. In terms of the orbit visualization, it is comparable to a modern application like the ones listed in the State of Art analysis (see section 1.2) or programs like Cosmographia [31].
Another minor point mentioned in section 1.1.5 is to keep the possibility of integrating the developed software modules into the existing ground command software. Since the limitations on C++ and the Qt framework introduced by this requirement were defined from the beginning, no compatibility issues should have arisen during the development. The QML View can easily be integrated into existing QtQuick GUIs. Afterwards, the C++ classes must be added, and the Controller thread started before the initialization of the View. Lastly, the folder structure and the dependency's must be added to the built environment.

Modifiability and extensibility were defined as the major requirements concerning the structure of the software. These characteristics were the primary influence during software architecture development and thereby have had a significant impact on the design. First of all, MVC was introduced as the primary segmentation. This achieved an almost complete abstraction between the View and the rest of the application. This separation has proved particularly useful, especially since the View has gown to an exceptionally vast and complex part of this project. For example, it was necessary to almost completely rewrite the Model during the development. In an earlier instance, the SPICE framework was not discovered yet, and therefore, an approach utilizing the JPL position approximation algorithm and a separate orientation earth model was implemented. However, NAIF's framework had too many advantages that the entire Earth Model was rebuilt. At this point, the separation of the View and the thereby added complexity paid them-self off. The absence of any visualization logic or GUI parts significantly decreased the complexity of the re-implementation process. Here, the introduction of a general communication pattern between layers as mentioned in section 2.3 supported the reconnection of different modules. A downside of MVC with Qt was noticed during the implementation. The transmission of data between C++ and QML happened through multiple `Q_INVOKABLE` methods (which can be called from within QML) placed inside the Controller class. They were necessary to separate the View from the model, provide the data in the correct format in collections, and provide

optimized methods for expensive computational processes (like the preprocessing of graph data). Nevertheless, they significantly extended the Controller and made it less readable and maintainable.
This problem could be solved by giving the View direct access to data in the Model. On the other hand, this would reduce the separation and restrict the usage of data types. Another solution would be to put an additional C++ layer between the View and the Controller. The thereby created C++ View could perform data preparation and conversion, as shown by Peter Winston in fig. 2.3.
A further decision was made regarding modifiability by utilising the scripting language Python for the transmission decoder. It allows using the SOMP tool for other satellites without the need for recompilation. The only requirements are that the new satellite provides its orientation as quaternion relative to the ECI frame.
The additional use of the Kaitai struct would make it for other SatNOGS users particularly easy since they might have the struct already created. An essential downside of this library is the limitation of their YAML format. It considerably restricts variable names (only small letters and numbers are allowed), and even though the contained "documentation" is used to extract names, the readability of the values still suffers. Further, it is required to create a conversion rule even for single Bit types like Boolean. Otherwise, the received numbers in the application are not usable, like for some combined values in the SOMP Kaitai struct.
The decision to store all downloaded and translated files in JSON format has proven to work well for small amounts of data. Moreover, it has the added possibility of directly integrating non-SatNOGS data. Nevertheless, as soon as the data grows, the lack of structure makes its presence felt. A database, ideally hosted on a server, similar to other ground command and visualization tools, would be advisable.
The last requirement on the software is that enough performance is provided to create a streamlined user experience. Therefore, concurrency was added to the Controller. It has the advantage that it can overtake parts of the calculation not to overload the GUI thread. In the final application, it works as expected for the 3D orbit visualization. Thereby View is rendering the image, and all model calculations are run in the background. If a new state is obtained, the GUI get informed and updates itself. Therefore the View stays responsive even when the propagation is fast-forwarded.
Nevertheless, this method was not able to remove all instances of processing. For example, $10 - 30s$ are required to initially load the 3D textures into the graphics memory. A background loader on another thread could mitigate this. Furthermore, the preparation of graphs with multiple thousands of data point and the pass and illumination state prediction takes can take a noticeable amount of time. Both could be improved through enhanced multi-threading.

It also has to be mentioned that in the release build of the SOMP tool, random crashes were noticed. An investigation has shown that the reason is inside the Qt3dRenderer. An upgrade to their newer Qt version 6 with DirectX support might solve this problem.
Besides these general requirements on the software, the goal was to develop an application that can predict the exact moment of an eclipse and pass occurrences. During chapter 3 and chapter 4 multiple verifications provide a base inspection of the implementation. In section 1.1.4 it was assessed, that for a LEO satellite with a velocity of $v_{satellite} \approx 7.6km/s$ the SGP 4 model can predict events with a $1s$ accuracy for a period of 2-3 days. In the last part of this chapter, this statement will be compared to the results of the implemented model.
Therefore the SOMP tool was used to download TLE data for a period of four-month. Each daily data was then individually put into the `\downloads` folder and was used to predict the position of SOMP 2b on the 31.05.2021 at 12:00 o'clock. Each result was then compared with the position calculated out of the TLE data from 31.05.2021. Even though this comparative data is also inflicted by the instantaneous error of the SGP4/TLE model, its approximate error of $\leq 500m$ (see section 1.1.4) is relatively small to an error that has built up over one month. It is even small enough to estimate the period the model is accurate enough to provide a $1s$ accuracy ($7.6km$ error threshold).
The data that was generated by this process is provided in appendix C.2. For further analysis the RMS position error and the z-axis position error were calculated. Both errors are visualized in fig. 5.1 and fig. 5.2.

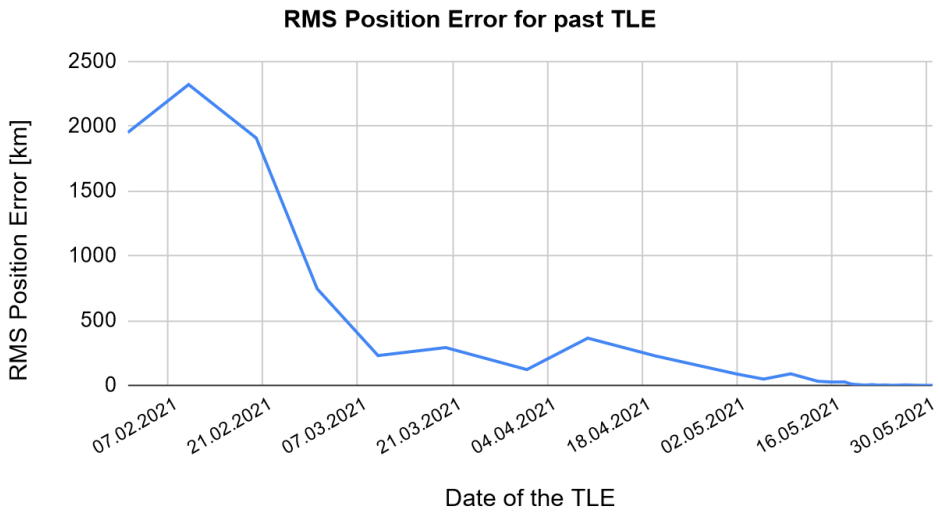

Figure 5.1: Longtime propagation RMS Position Error of implemented SGP 4 Model

The RMS position error graph shows a general error increase over the observation period with some minor fluctuations. After around three months, a steep increase is visible. Compared to the data shown in NASA's investigation [27] for MEO satellites, the increase is far less steady. Therefore the estimated error increase of $\approx 5km/day$ is not recognisable. Furthermore, the period in which a $1s$ event epoch accuracy can be expected is more than tripled from 2-3 days

to 10 days. Reasons for this might be that the accuracy optimisation that happened in 2015 for TLE generation were undervalued, while the LEO orbit influence was overvalued.
This whole estimation is based on the assumption that the along-track error dominates the cross and radial error of the orbit. Since this investigation takes place at a latitude of $\approx 13°$, this assumption can be validated if the RMS error is almost entirely dependent on the z-axis position error.

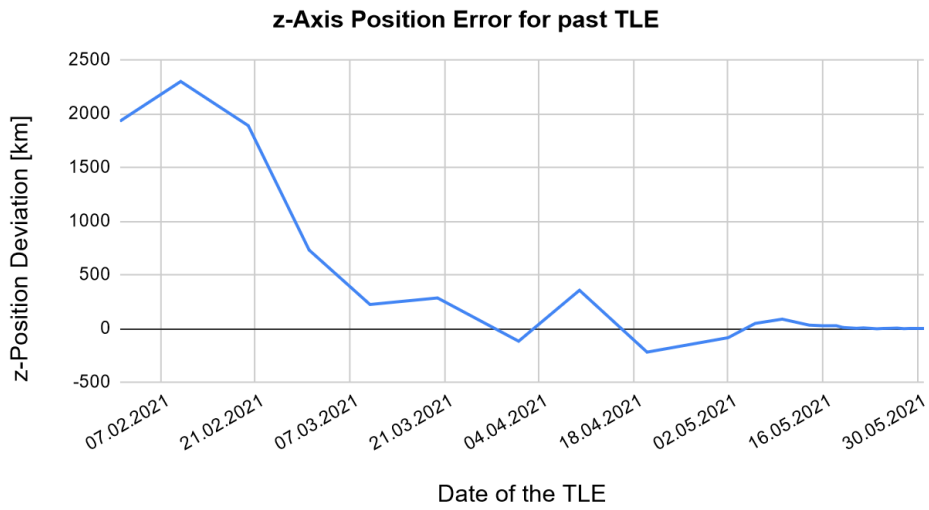

Figure 5.2: Longtime propagation z-Axis Position Error of implemented SGP 4 Model

Figure 5.2 and the data indicate that for 77% of the data entries the RMS error has less than a 5% deviation from the z-axis error. The graph also shows an interesting behaviour, where the z-axis error oscillates around a value of zero until, in February, a strong positive bias arises. The huge deviation in this time might result from lower tracking accuracy directly after the satellite's launch. In general, however, it can be concluded from this additional validation that the event epoch prediction of the implemented satellite model provides more precise results than initially estimated and can provide accuracy to the second for more than a week. Nevertheless, further examinations, especially for the orientation, pass and eclipse prediction through direct satellite data, could greatly improve confidence.

6 Conclusion

The objective of this thesis was to provide an extension for the ground command software for mission planning purposes, comprising of eclipse and overflight events, and visualisation of ADCS parameters for the nanosatellite SOMP 2b.
To obtain an overview, a state of the art analyses was used to identify already available applications. Its result mainly was comprised of expensive solution aimed at more extensive projects as well as outdated tools. Nevertheless, NASA's Open MCT was identified as a versatile and modern approach, which is also available as open-source software. However, since it has, similar to most other tools, a server-based structure that is not suitable for the SOMP project, it was decided for proprietary development.

After analysing the requirements, it was possible to create software with excellent capabilities for extension and modifiable through an architecture centred software development process. Primarily due to the usage of MVC design pattern, it was possible to abstract an extensive and complex View. The additional multi-threading provided the required computational performance for real-time calculations. An optimisation, as mentioned in chapter 5, of both structural patterns could further enhance the user experience and maintainability.
It has also been demonstrated that the usage of the embedded scripting language python for modules that can constantly change during operation, like the transmission decoder, is very beneficial since no recompilation is required. For the binary conversion of AX.25 and SOMP Beacon frames, the Kaitai struct library was used. It further extends the capability of the tool to be adapted for other SatNOGS satellites than SOMP 2b.
Furthermore, a REST API was successfully implemented and provides a good solution for the integration into data servers.

For the prediction of earth's orbit and orientation as well as the satellite orbits NASA NAIF's SPICE toolset was chosen. It provides a versatile approach for high accuracy calculation of space geometries based on scientific kernels. Through an analysis performed in chapter 5, it could be demonstrated that the integrated SGP4/TLE model in combination with the SPICE calculation routines can provide a position accuracy below $7.6km$ for more than a week. Therefore the event epoch predictions in the same period can be approximated to be below one second. These results were surprising since an investigation of research in section 1.1.4 has pointed to a significantly lower level of accuracy.
An eclipse and pass detection algorithm was developed and successfully tested for their basic functionality to extend these models.

7 Future Work

In this final chapter, some suggestions are made for extending this work. The first ones are concerning software architecture. As mentioned in chapter 5, the scope of Controller is currently overloaded, and therefore its maintainability and readability suffer. To mitigate this downside of the MVC implementation, it is advised to separate the data preparation logic for the GUI from the Controller into an additional layer between them. Thereby the performance of C++ and the abstraction of the View can be kept. A further improvement on the architecture is to expand the use of multi-threading and, consequently, enhance the application's performance.

Additionally, multiple advancements can be made in terms of orbit visualisation. A collection of useful features are listed below:

- A shadow map can be added to the earth, thereby visualising the eclipse phases.
- For better orientation in the space, a visible sun can be implemented, and a starfield as a skybox could be added.
- A switch to the newest major Qt version 6 and DirectedX instead of OpenGL could be conducted to remove random crashes of the 3D rendering.
- Lastly, country borders could be added to give a better intuition about the satellite's position.

As a last major topic, the SGP 4 model could be exchanged for a more precise model to improve the predictions.

Bibliography

[1] William A., Douglas E., and Jack Taylor.
AX.25 Link Access Protocol for Amateur Packet Radio. English.
Tucson Amateur Packet Radio Corporation, 1998.
URL: http://www.ax25.net/AX25.2.2-Jul%5C%2098-2.pdf5 (visited on 05/11/2021).

[2] Saika Aida and Michael Kirschner. *ACCURACY ASSESSMENT OF SGP4 ORBIT INFORMATION CONVERSION INTO OSCULATING ELEMENTS*. English. DLR, 2013.
URL: https://elib.dlr.de/87081/1/5a_P6_aida.pdf (visited on 05/09/2021).

[3] *Alen Space's Mission Control Software Shop*. English. CubeSatShop. URL:
https://www.cubesatshop.com/product/mission-control-and-flight-software/
(visited on 11/13/2020).

[4] *An Overview of Reference Frames and Coordinate Systems*. English. NAIF.
URL: https://naif.jpl.nasa.gov/pub/naif/toolkit_docs/Tutorials/pdf/individual_docs/17_frames_and_coordinate_systems.pdf (visited on 04/30/2021).

[5] Len Bass, Paul Clements, and Rick Kazman. *Software Architecture in Practice*. English.
3rd ed. Pearson Education Inc, 2012. ISBN: 9780321815736.

[6] *Boost library*. English. Github.
URL: https://github.com/boostorg/beast (visited on 05/20/2020).

[7] *BOSS Dashboard Homepage*. English. Solenix.
URL: https://www.solenix.ch/boss (visited on 11/13/2020).

[8] *CalesTrak Orbit Visualization*. English. CalesTrak.
URL: https://celestrak.com/cesium/orbit-viz.php?tle=/pub/TLE/catalog.txt&satcat=/pub/satcat.txt&referenceFrame=1
(visited on 05/30/2020).

[9] T. Colburn and G. Shute. "Abstraction in Computer Science". English. In: (2007).
DOI: 10.1007/s11023-007-9061-7.

[10] *cppbestpractices*. English. gitbooks.
URL: https://lefticus.gitbooks.io/cpp-best-practices/content/ (visited on 05/18/2021).

[11] *Elveti Mission Control System Homepage*. English. Solenix.
URL: https://www.solenix.ch/elveti (visited on 11/13/2020).

[12] *Elveti Mission Control System Shop*. English. CubeSatShop.
URL: https://www.cubesatshop.com/product/elveti-mission-control-system/
(visited on 11/13/2020).

[13] *Ephemerides*. English. Jet Propulsion Laboratory.
URL: `https://ssd.jpl.nasa.gov/?ephemerides` (visited on 03/17/2021).

[14] *Glouton SatNOGS*. English. Github.
URL: `https://github.com/deckbsd/glouton-satnogs-data-downloader/releases` (visited on 05/20/2020).

[15] Felix R. Hoots and Ronald L. Roehrich.
SPACETRACK REPORT NO. 3 Models for Propagation of NORAD Element Sets. English.
Celestrak, 1980.
URL: `http://www.celestrak.com/NORAD/documentation/spacetrk.pdf` (visited on 05/03/2021).

[16] *HORIZONS System*. English. Jet Propulsion Laboratory.
URL: `https://ssd.jpl.nasa.gov/?horizons` (visited on 03/17/2021).

[17] *HORIZONS System*. English. Jet Propulsion Laboratory. URL:
`https://ssd.jpl.nasa.gov/?horizons_doc#limitations` (visited on 03/17/2021).

[18] *Icon Pack: Essential Set*. English. flaticon.
URL: `https://www.flaticon.com/packs/essential-set-2` (visited on 05/15/2021).

[19] F. Landis Markley und John L. Crassidis.
Fundamentals of Spacecraft Attitude Determination and Control. English.
Springer New York, 2014. ISBN: 978-1-4939-0801-1.

[20] *JSatTrak Homepage*. English. JSatTrak.
URL: `https://www.gano.name/shawn/JSatTrak/index.html` (visited on 11/13/2020).

[21] *Kaitai Struct Homepage*. English. Kaitai.
URL: `https://kaitai.io/` (visited on 05/24/2020).

[22] *Keplerian Elements for Approximate Positions of the Major Planets*. English.
Jet Propulsion Laboratory. URL:
`https://ssd.jpl.nasa.gov/txt/aprx_pos_planets.pdf` (visited on 03/17/2021).

[23] *Keplerian Elements for Approximate Positions of the Major Planets Overview*. English.
Jet Propulsion Laboratory.
URL: `https://ssd.jpl.nasa.gov/?planet_pos` (visited on 03/17/2021).

[24] *Kubos: Major Tom Homepage*. English. Kubos.
URL: `https://www.kubos.com/` (visited on 11/13/2020).

[25] *Kubos: Major Tom launch news*. English. Cision.
URL: `https://www.prweb.com/releases/2017/08/prweb14571023.htm` (visited on 11/13/2020).

[26] K. Kunze and H. Schaeben. *The Bingham distribution of quaternions and its spherical radon transform in texture analysis*. English. Springer-Verlag, 2004.
DOI: 10.1023/B:MATG.0000048799.56445.59.

[27] Creon Levit and William Marshall.
Improved orbit predictions using two-line elements. English. NASA, 2010.
URL: https://arxiv.org/pdf/1002.2277.pdf (visited on 05/09/2021).

[28] *List of cubesat suppliers*. English. cubesat.org.
URL: https://www.cubesat.org/suppliers (visited on 11/13/2020).

[29] *Microsoft REST SDK*. English. Github.
URL: https://github.com/microsoft/cpprestsd (visited on 05/20/2020).

[30] *MUST Homepage*. English. Solenix.
URL: https://www.solenix.ch/products/must (visited on 11/13/2020).

[31] *NASA NAIF Cosmographia*. English. NASA.
URL: https://naif.jpl.nasa.gov/naif/cosmographia.html (visited on 05/29/2020).

[32] *NASA NAIF MKSPK*. English. NASA. URL: https://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/ug/mkspk.html#Input%5C%20Data%5C%20Type%5C%20TL_ELEMENTS%5C%20/%5C%20Output%5C%20SPK%5C%20Type%5C%2010 (visited on 05/30/2020).

[33] *NASA NAIF SPICE earth_assoc_itrf93.tf*. English. NASA.
URL: https://naif.jpl.nasa.gov/pub/naif/generic_kernels/fk/planets/earth_assoc_itrf93.tf (visited on 05/28/2020).

[34] *NASA NAIF SPICE Homepage*. English. NASA.
URL: https://naif.jpl.nasa.gov/naif/index.html (visited on 05/26/2020).

[35] *NASA NAIF SPICE SPK Kernels*. English. NASA.
URL: https://naif.jpl.nasa.gov/pub/naif/generic_kernels/spk/planets/ (visited on 05/28/2020).

[36] *NASA visible earth*. English. NASA.
URL: https://visibleearth.nasa.gov/collection/1484/blue-marble?page=2 (visited on 05/15/2021).

[37] *Nova for Windows Homepage*. English. NLSA.
URL: https://www.nlsa.com/nfw.html (visited on 11/13/2020).

[38] *OPEN MCT github*. English. NASA.
URL: https://github.com/nasa/openmct.git (visited on 11/13/2020).

[39] *OPEN MCT homepage*. English. NASA.
URL: https://nasa.github.io/openmct/about-open-mct/ (visited on 11/13/2020).

[40] *OPEN MCT live example*. English. NASA. URL: https://openmct-demo.herokuapp.com/#/browse/demo:root/demo:3?view=layout&tc.mode=local&tc.timeSystem=utc&tc.startDelta=900000&tc.endDelta=0 (visited on 11/16/2020).

[41] *Planet Labs 18 SPCS Matching Report v1*. English. Planet Labs. URL: https://ephemerides.planet-labs.com/18SPCS_matches_v1.txt (visited on 05/10/2021).

[42] *Qt Homepage*. English. Qt. URL: https://www.qt.io/ (visited on 05/10/2021).

[43] Danielle Racelis and Mathieu Joerger. *High-Integrity TLE Error Models for MEO and GEO Satellites*. English. 2018. URL: https://www.aoe.vt.edu/content/dam/aoe_vt_edu/people/faculty/joerger/publications/2018_aiaa_space_forum_paper_racelis-joerger.pdf (visited on 05/09/2021).

[44] *REST Wikipedia*. English. Wikipedia. URL: https://en.wikipedia.org/wiki/Representational_state_transfer (visited on 04/29/2021).

[45] *SatNOGS Bobcat-1*. English. SatNOGS. URL: https://dashboard.satnogs.org/d/BFU00FHMz/bobcat-1?orgId=1&refresh=10s (visited on 11/13/2020).

[46] *SatNOGS Database*. English. SatNOGS. URL: https://db.satnogs.org/ (visited on 04/29/2021).

[47] *SatNOGS Homepage*. English. SatNOGS. URL: https://satnogs.org/about/ (visited on 11/13/2020).

[48] *SatNOGS Network*. English. SatNOGS. URL: https://network.satnogs.org/ (visited on 04/29/2021).

[49] *SatNOGS Network API*. English. Wikipedia. URL: https://satnogs.org/2014/10/30/api-on-satnogs-network (visited on 04/29/2021).

[50] *SatNOGS Network Observation 4007887 of SOMP2b*. English. SatNOGS. URL: https://network.satnogs.org/observations/4007887/ (visited on 04/29/2021).

[51] *SatNOGS Observation Example*. English. SatNOGS. URL: https://network.satnogs.org/observations/?future=0&bad=0&unknown=0&failed=0&norad=47445&observer=&station=&start=2021-05-18+00%5C%3A00&end=2021-05-18+12%5C%3A00 (visited on 05/20/2020).

[52] *SatNOGS Wikipedia*. English. Wikipedia. URL: https://en.wikipedia.org/wiki/SatNOGS (visited on 11/13/2020).

[53] *SOMP-2B orbit overview*. English. N2YO.com.
URL: https://www.n2yo.com/satellite/?s=47445 (visited on 05/12/2021).

[54] *Turbosquid satellite model*. English. TurbuSquid. URL:
https://www.turbosquid.com/3d-models/litsat-1-cubesats-3d-3ds/877199
(visited on 05/15/2021).

[55] David A. Vallado et al. *Revisiting Spacetrack Report NO. 3: Rev 2*. English. Celestrak, 2006.
URL: http://celestrak.com/publications/AIAA/2006-6753/AIAA-2006-6753-Rev2.pdf (visited on 05/03/2021).

[56] *Wikipedia ECEF*. English. Wikipedia.
URL: https://en.wikipedia.org/wiki/ECEF (visited on 04/30/2021).

[57] *Wikipedia Equinox*. English. Wikipedia.
URL: https://en.wikipedia.org/wiki/Equinox (visited on 05/30/2020).

[58] Peter Winston. *Qt World Summit Online - Lessons Learned from Building 100+ Devices with C++/Qt/QML*. English. Qt. 2020.
URL: https://www.qt.io/qtws20-online (visited on 05/18/2021).

A Additional Figures

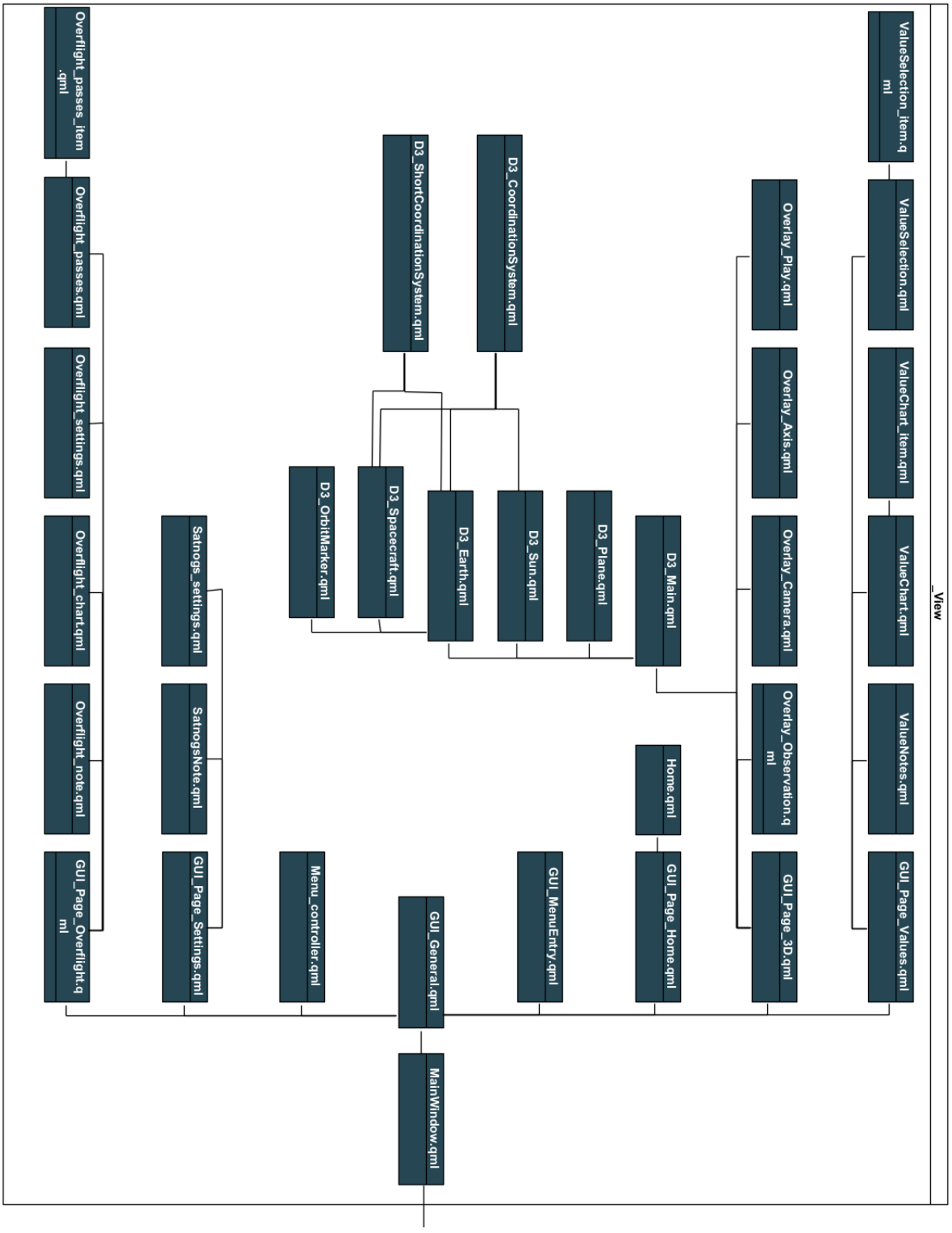

Figure A.1: UML: View

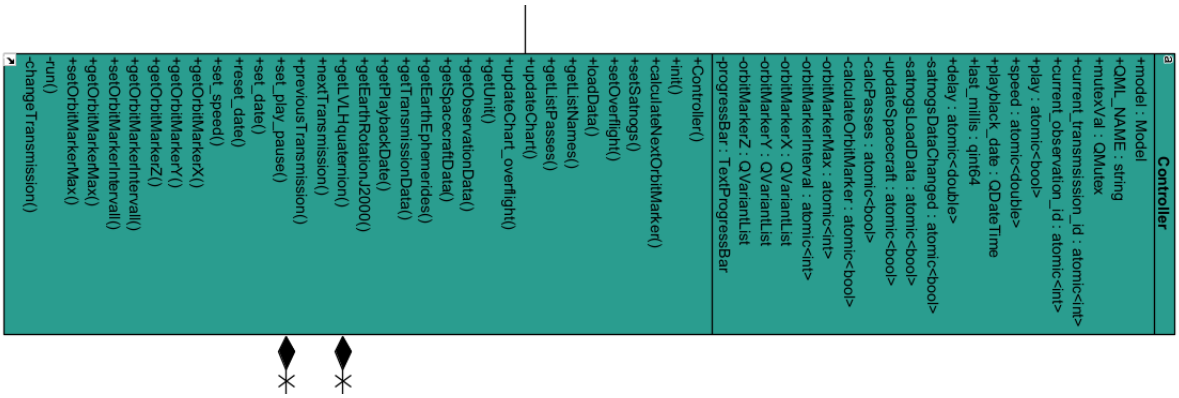

Figure A.2: UML: Controller

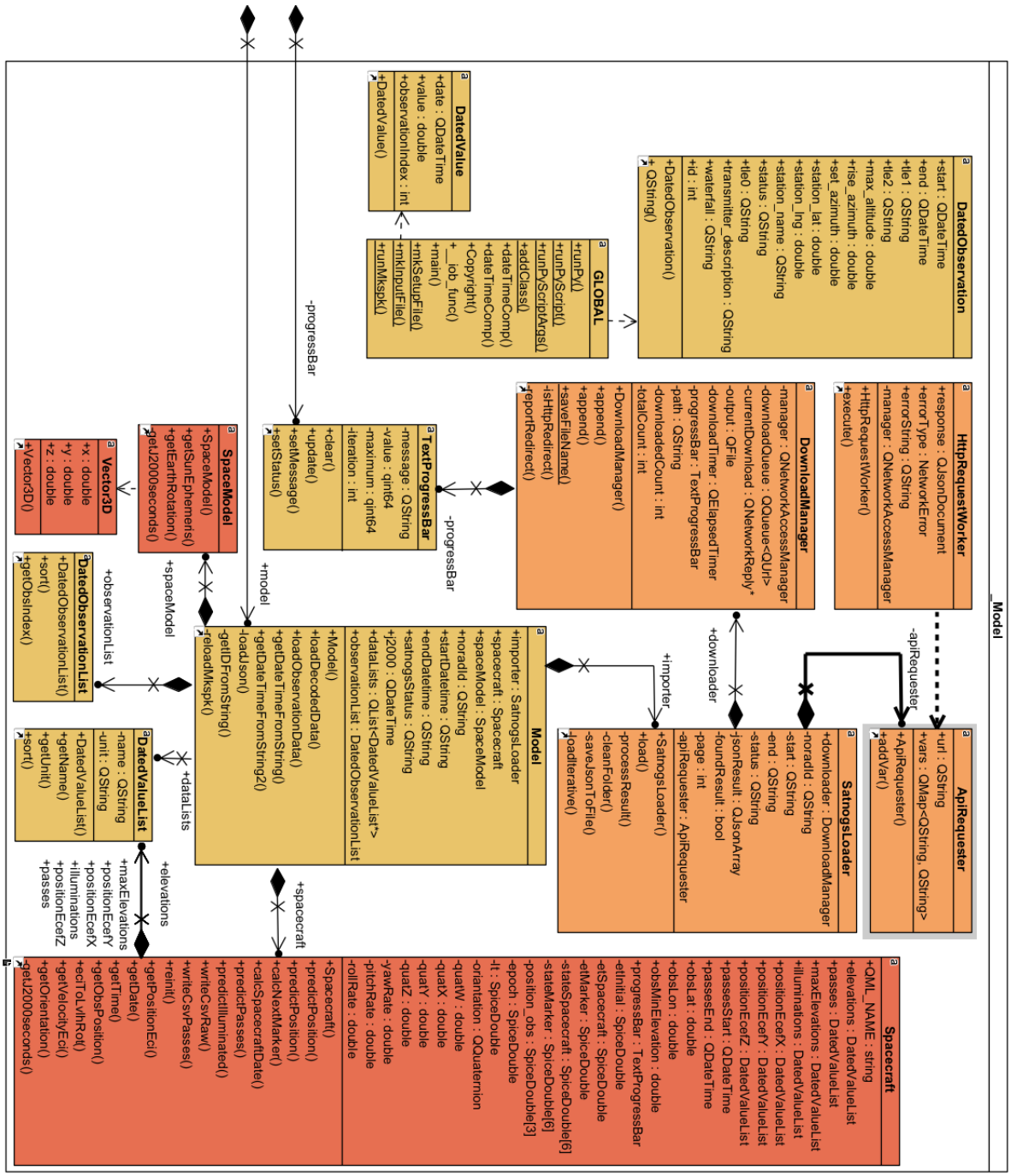

Figure A.3: UML: Model

B Additional Information

B.1 SOMP2b beacon definition

Downlink:

- Frequency: 435.600 Mhz
- Modulation: 9k6 GMSK

AX.25 Frame properties

- Call sign from: DP2TUD
- Call sign to: DO7BY
- Frame type: UI frame

Byte and bit order notes

- Byte order:Least significant byte first on multi-byte numbers (little endian)
- Call sign to: DO7BY
- Bit order: Most significant bit first

Beacon Decoder Telemetry ID 0x30 (Standard Beacon, every 60s)

Bits	Data Description	Data Information	Unit
08	Message Type	0x54, fix	[-]
08	Telemetry ID	0x30, fix	[-]
32	OBC Time Stamp	(Unix UTC since 01/01/2000)	[s]
32	Uptime since last reboot		[s]
12	Battery1 – Voltage	Divider: 2.00 , Offset: -2500.0	[mV]
12	Battery1 – Charge current	Divider: 0.50 , Offset: 4094.0	[mA]
10	Battery1 – Battery Temperature	Divider: 0.70 , Offset: 400.0	[0.1·°C]
08	Battery1 – State of Charge		[%]
12	Battery2 – Voltage	Divider: 2.00 , Offset: -2500.0	[mV]
12	Battery2 – Charge current	Divider: 0.50 , Offset: 4094.0	[mA]
10	Battery2 – Battery Temperature	Divider: 0.70 , Offset: 400.0	[0.1·°C]
08	Battery2 – State of Charge		[%]
10	Y- Panel – Current from SCs to Battery	Divider: 0.50 , Offset: 00.0	[mA]
10	X- Panel – Current from SCs to Battery	Divider: 0.50 , Offset: 00.0	[mA]
10	Y+ Panel – Current from SCs to Battery	Divider: 0.50 , Offset: 00.0	[mA]
10	X+ Panel – Current from SCs to Battery	Divider: 0.50 , Offset: 00.0	[mA]
10	Z+ Panel – Current from SCs to Battery	Divider: 0.50 , Offset: 00.0	[mA]
10	Y- Panel – Temperature	Divider: 0.70 , Offset: 400.0	[0.1·°C]
10	X- Panel – Temperature	Divider: 0.70 , Offset: 400.0	[0.1·°C]
10	Y+ Panel – Temperature	Divider: 0.70 , Offset: 400.0	[0.1·°C]
10	X+ Panel – Temperature	Divider: 0.70 , Offset: 400.0	[0.1·°C]
10	Z+ Panel – Temperature	Divider: 0.70 , Offset: 400.0	[0.1·°C]
12	3V3 Bus – Voltage	Divider: 1.00 , Offset: -2789.0	[mV]
10	3V3 Bus – Current Total	Divider: 1.00 , Offset: 00.0	[mA]
10	3V3 Bus – Current Converter 1	Divider: 1.00 , Offset: 00.0	[mA]
10	3V3 Bus – Current Converter 2	Divider: 1.00 , Offset: 00.0	[mA]
10	CS Module – Temperature	Divider: 0.70 , Offset: 400.0	[0.1·°C]
02	CS Module – Antenna Deployment	Bit 0: Antenna1, Bit 1: Antenna2 (Bit set if deployed)	[-]
06	CS Module – Active System	Bit 0…1: System, Bit 2…6: Padding	[-]
08	CS Beacon Interval	10s fix, reserved for future use	[s]
16	OBC Reset Counter		[-]
16	OBC Telemetry Packet ID		[-]
10	OBC Temperature	Divider: 0.70 , Offset: 400.0	[0.1·°C]
08	OBC CPU Load		[%]
08	OBC Last Reboot Reason		[-]
04	ADCS Mode	Bit 0: Eclipsed, Bit 1…3: Mode (0x0 – off, 0x1 – ADS, 0x2 – ADCS, 0x3 – Detumbling)	[-]
04	OBC Mode		[-]

Figure B.1: Beacon Decoder Telemetry ID 0x30 (Standard Beacon, every 60s); Part 1

Bits	Data Description	Data Information	Unit
12	ADCS Magnetic Field X (normalised)	Divider: 2.00 , Offset: 1023.0	[-]
12	ADCS Magnetic Field Y (normalised)	Divider: 2.00 , Offset: 1023.0	[-]
12	ADCS Magnetic Field Z (normalised)	Divider: 2.00 , Offset: 1023.0	[-]
12	ADCS Sun Vector X	Divider: 2047.00 , Offset: 01.0	[-]
12	ADCS Sun Vector Y	Divider: 2047.00 , Offset: 01.0	[-]
12	ADCS Sun Vector Z	Divider: 2047.00 , Offset: 01.0	[-]
16	ADCS Estimated Attitude $q_{ECI \leftarrow Body}$ w	Divider: 32767.00 , Offset: 01.0	[-]
16	ADCS Estimated Attitude $q_{ECI \leftarrow Body}$ i	Divider: 32767.00 , Offset: 01.0	[-]
16	ADCS Estimated Attitude $q_{ECI \leftarrow Body}$ j	Divider: 32767.00 , Offset: 01.0	[-]
16	ADCS Estimated Attitude $q_{ECI \leftarrow Body}$ k	Divider: 32767.00 , Offset: 01.0	[-]
12	ADCS Estimated Angular Rate X	Divider: 59.00 , Offset: 35.0	[rad/s]
12	ADCS Estimated Angular Rate Y	Divider: 59.00 , Offset: 35.0	[rad/s]
12	ADCS Estimated Angular Rate Z	Divider: 59.00 , Offset: 35.0	[rad/s]
06	Padding		[-]
32	Payload Information		[-]

Value Conversion:

$$val_{out} = \left(val\_beacon_{packed} / val\_beacon_{divider}\right) - val\_beacon_{offset}$$

Figure B.2: Beacon Decoder Telemetry ID 0x30 (Standard Beacon, every 60s); Part 2

Beacon Decoder Telemetry ID 0x31 (Alternative Beacon, every 180s)

Bits	Data	Data Type	Unit
08	Message Type	0x54, fix	[-]
08	Telemetry ID	0x31, fix	[-]
08	Antenna 1 deployed? (0 - No)	uint8_t	[-]
08	Antenna 2 deployed? (0 - No)	uint8_t	[-]
16	Battery1 – Voltage	uint16_t	[mV]
16	Battery1 – Charge current	int16_t	[mA]
16	Battery1 – Battery Temperature	int16_t	[0.1·°C]
16	Battery1 – IC Temperature	int16_t	[0.1·°C]
16	Battery1 – Remaining Capacity	uint16_t	[mAh]
16	Battery1 – State of Charge	uint16_t	[%]
16	Battery1 – Maximum Charge Current	int16_t	[mA]
16	Battery1 – Average Power	uint16_t	[mWh]
16	Battery1 – Full Available Capacity	uint16_t	[mAh]
16	Battery1 – Cycle Count of Charge Cycles	uint16_t	[-]
16	Battery1 – State of Health	uint16_t	[%]
16	Battery1 – Time to Empty in min	uint16_t	[min]
16	Battery1 – State Flags	uint16_t	[-]
16	Battery1 – Passed Charge	int16_t	[mAh]
16	Battery2 – Voltage	uint16_t	[mV]
16	Battery2 – Charge current	int16_t	[mA]
16	Battery2 – Battery Temperature	int16_t	[0.1·°C]
16	Battery2 – IC Temperature	int16_t	[0.1·°C]
16	Battery2 – Remaining Capacity	uint16_t	[mAh]
16	Battery2 – State of Charge	uint16_t	[%]
16	Battery2 – Maximum Charge Current	int16_t	[mA]
16	Battery2 – Average Power	uint16_t	[mWh]
16	Battery2 – Full Available Capacity	uint16_t	[mAh]
16	Battery2 – Cycle Count of Charge Cycles	uint16_t	[-]
16	Battery2 – State of Health	uint16_t	[%]
16	Battery2 – Time to Empty in min	uint16_t	[min]
16	Battery2 – State Flags	uint16_t	[-]
16	Battery2 – Passed Charge	int16_t	[mAh]
16	CS module – Operation Counter	uint16_t	[-]
16	CS module – Temperature [°C]	int16_t	[°C]
08	CS module – Time Stamp #1	uint8_t	[-]
08	CS module – Time Stamp #2	uint8_t	[-]
08	CS module – Time Stamp #3	uint8_t	[-]
08	CS module – RSSI	uint8_t	[-]
32	CS module – # Received Bytes	uint32_t	[-]

Figure B.3: Beacon Decoder Telemetry ID 0x31 (Alternative Beacon, every 180s); Part 1

32	CS module – # Transmitted Bytes	uint32_t	[-]
16	3V3 Bus – Total Current	int16_t	[mA]
16	3V3 Converter 1 – Current	int16_t	[mA]
16	3V3 Converter 2 – Current	int16_t	[mA]
16	3V3 Bus – Voltage	int16_t	[mV]
16	3V3 Converter 1 – Voltage	int16_t	[mV]
16	3V3 Converter 2 – Voltage	int16_t	[mV]
32	Reset Counter	uint32_t	[-]
32	Time Stamp from last Reboot (Unix UTC)	uint32_t	[-]
32	Time since last Reboot	uint32_t	[s]

Figure B.4: Beacon Decoder Telemetry ID 0x31 (Alternative Beacon, every 180s); Part 2

C Additional Tables

C.1 Data Decoder Comparison Results

	Chart Visualization	Kaitai Struct conversion
adcs_mode [ ]	1	1
angular_rate_x [ ]	0.034	0.034
ADCS Angular Rate X [deg/s]	2067	2067
angular_rate_y [ ]	0.017	0.017
ADCS Angular Rate Y [deg/s]	2066	2066
angular_rate_z [ ]	0.492	0.492
ADCS Angular Rate Z [deg/s]	2094	2094
Antenna 1 deployed [0 - No]	1	1
Antenna 2 deployed [0 - No]	1	1
b_field_x [ ]	-0.5	-0.5
ADCS Magnetic Field X (normalised) [-]	2045	2045
b_field_y [ ]	0.0	0.0
ADCS Magnetic Field Y (normalised) [-]	2046	2046
b_field_z [ ]	-1.0	-1
ADCS Magnetic Field Z (normalised) [-]	2044	2044
bat1_i_charge [ ]	4.0	4.0
bat1_i_charge_raw [ ]	2049	2049
Battery1 - State of Charge [%]	99	99
bat1_temp [ ]	-4.429	-4.429
bat1_temp_raw [ ]	249	249
Battery1 - Voltage [mV]	3444	3444
bat2_i_charge [ ]	0.0	0.0
bat2_i_charge_raw [ ]	2047	2047
Battery2 - State of Charge [%]	99	99
bat2_temp [ ]	-3.857	-3.857
bat2_temp_raw [ ]	253	253
Battery2 - Voltage [mV]	3448	3448
Time since last Reboot [s]	49381	49381
OBC Time Stamp [s]	674639388	674639388
CS Beacon Interval [s]	10	10
cs_module [ ]	1	1
cs_status [ ]	7	7
CS module - Temperature [°C]	12.857	12.857
cs_temp_raw [ ]	289	289
eclipsed [ ]	0	0

Table C.1: Verification: Data Decoder (18.05.2021 07:51:52 ID: 4125933) Part 1

	Chart Visualization	Kaitai Struct conversion
obc_adcs_mode [ ]	2	2
OBC Last Reboot Reason [-]	0	0
OBC CPU Load [%]	12	12
obc_mode [ ]	0	0
Reset Counter [ ]	168	168
OBC Telemetry Packet ID [-]	30712	30712
obc_temp [ ]	-0.143	-0.143
obc_temp_raw [ ]	279	279
pnl_i_bat_xm [ ]	152.0	152.0
pnl_i_bat_xm_raw [ ]	76	76
pnl_i_bat_xp [ ]	162.0	162.0
pnl_i_bat_xp_raw [ ]	81	81
pnl_i_bat_ym [ ]	4.0	4.0
pnl_i_bat_ym_raw [ ]	2	2
pnl_i_bat_yp [ ]	2.0	2.0
pnl_i_bat_yp_raw [ ]	1	1
pnl_i_bat_z [ ]	106.0	106.0
pnl_i_bat_z_raw [ ]	53	53
pnl_temp_xm [ ]	400.0	400.0
pnl_temp_xm_raw [ ]	560	560
pnl_temp_xp [ ]	367.143	367.143
pnl_temp_xp_raw [ ]	537	537
pnl_temp_ym [ ]	480.0	480.0
pnl_temp_ym_raw [ ]	616	616
pnl_temp_yp [ ]	485.714	485.714
pnl_temp_yp_raw [ ]	620	620
pnl_temp_z [ ]	0.0	0.0
pnl_temp_z_raw [ ]	280	280
q_ib_i [ ]	-0.414	-0.414
q_ib_i_raw [ ]	19217	19217
q_ib_j [ ]	-0.195	-0.195
ADCS qECI j [ ]	26387	26387
q_ib_k [ ]	0.201	0.201
ADCS qECI k [ ]	39367	39367
q_ib_w [ ]	0.866	0.866
ADCS qECI w [ ]	61154	61154
science_pld_info [ ]	0	0
stuffingbits [ ]	0	0
sun_vect_x [ ]	0.810	0.810
ADCS Sun Vector X [ ]	3705	3705
sun_vect_y [ ]	-0.326	-0.326
ADCS Sun Vector Y [ ]	1380	1380
sun_vect_z [ ]	0.488	0.488
ADCS Sun Vector Z [ ]	3045	3045
v3v3_bus_i_c1 [ ]	215	215
v3v3_bus_i_c2 [ ]	49	49
v3v3_bus_i_tot [ ]	257	257
3V3 Bus - Voltage [mV]	467	467

Table C.2: Verification: Data Decoder (18.05.2021 07:51:52 ID: 4125933) Part 2

C.2 SGP4 Model Propagation Accuracy Examination

Date	x-Position ECI [km]	y-Position ECI [km]	z-Position ECI [km]	RMS Position Error [km]	z-Axis Error [km]
31.05.2021	5.783,06	3.490,96	-1.419,92	0,00	0,00
30.05.2021	5.783,18	3.491,04	-1.419,69	0,27	0,23
29.05.2021	5.783,08	3.491,14	-1.419,38	0,57	0,54
28.05.2021	5.783,03	3.490,56	-1.421,93	2,05	-2,01
27.05.2021	5.783,71	3.491,88	-1.416,21	3,88	3,71
26.05.2021	5.783,55	3.491,42	-1.418,32	1,74	1,60
25.05.2021	5.783,49	3.491,17	-1.419,55	0,60	0,37
24.05.2021	5.783,16	3.490,41	-1.423,07	3,20	-3,15
23.05.2021	5.783,73	3.491,48	-1.418,42	1,72	1,50
22.05.2021	5.784,22	3.492,38	-1.414,46	5,76	5,46
21.05.2021	5.783,85	3.491,54	-1.418,26	1,93	1,66
19.05.2021	5.784,74	3.493,22	-1.410,80	9,54	9,12
18.05.2021	5.786,54	3.496,93	-1.394,33	26,51	25,59
16.05.2021	5.786,56	3.496,87	-1.394,83	26,01	25,09
14.05.2021	5.787,20	3.498,19	-1.389,08	31,95	30,84
10.05.2021	5.792,90	3.510,52	-1.333,13	89,51	86,79
06.05.2021	5.788,84	3.501,96	-1.372,89	48,64	47,03
02.05.2021	5.773,96	3.472,17	-1.505,60	88,19	-85,68
20.04.2021	5.755,82	3.440,33	-1.640,07	227,53	-220,15
10.04.2021	5.812,21	3.566,80	-1.064,28	364,80	355,64
01.04.2021	5.766,94	3.465,93	-1.538,37	122,13	-118,45
20.03.2021	5.804,83	3.554,84	-1.135,86	291,97	284,06
10.03.2021	5.798,97	3.544,66	-1.196,59	230,25	223,33
01.03.2021	5.821,59	3.640,29	-689,85	746,18	730,07
20.02.2021	5.753,49	3.778,88	467,74	1.909,72	1.887,66
10.02.2021	5.688,25	3.804,34	879,30	2.322,41	2.299,22
01.02.2021	5.745,75	3.787,30	509,24	1.952,14	1.929,16

Table C.3: SGP4 Model Propagation Accuracy Examination Data

D CD Data

D.1 Digital Version of the Thesis

D.2 Software

D.2.1 Source Code

D.2.2 Compiled SOMP 2b Application

D.3 UML Diagrams

D.4 Kaitai Struct