universität wien

MASTERARBEIT | MASTER'S THESIS

Titel | Title

Software Integration Testing for SMILE

verfasst von | submitted by

Sebastian Miksch BSc

angestrebter akademischer Grad | in partial fulfilment of the requirements for the degree of

Master of Science (MSc)

Wien | Vienna, 2024

Studienkennzahl lt. Studienblatt | Degree programme code as it appears on the student record sheet:	UA 066 861
Studienrichtung lt. Studienblatt | Degree programme as it appears on the student record sheet:	Masterstudium Astronomie
Betreut von | Supervisor:	Univ.-Prof. Mag. Dr. Franz Kerschbaum
Mitbetreut von | Co-Supervisor:	Mag. Dr. Roland Ottensamer

UNIVERSITY OF VIENNA

Abstract

Faculty of Earth Sciences, Geography and Astronomy
Department of Astrophysics

Master of Science

Software Integration Testing for SMILE

by Sebastian MIKSCH

Der Solar Wind Magnetosphere Ionosphere Link Explorer (SMILE) untersucht die Wechselwirkungen zwischen den geladenen Teilchen der Sonne und dem Magnetfeld der Erde in bisher unerreichter Detailtiefe. Für diesen Zweck werden vier Instrumente mitgeführt: das SXI, das UVI, das MAG und das LIA. Jedes dieser Instrumente muss mit äußerster Sorgfalt gefertigt werden. Wir haben das Test Specification Tool (TST) entwickelt, um die Prüfung der Data Processing Unit (DPU) des SXI zu unterstützen. Indem Datenpakete an den Satelliten gesendet und von diesem empfangen werden, überprüft das TST die Ausgabe gegen die ECSS-Datenbanken, um sicherzustellen, dass die Software korrekt funktioniert. Ein Test ist dabei in zwei Teile gegliedert: den Testbefehl und die Verifizierung. Der Nutzer des TST gibt beide Teile des Tests an. Nachdem der Test spezifiziert wurde, erstellt das TST die erforderlichen Testskripte, die anschließend ausgeführt werden können. Im Progress Viewer zeigt das TST die Testergebnisse in einer für Menschen lesbaren Form an. Das TST vereinfacht die Erstellung der notwendigen Testskripte und automatisiert den Testprozess so weit wie möglich.

UNIVERSITY OF VIENNA

Abstract

Faculty of Earth Sciences, Geography and Astronomy
Department of Astrophysics

Master of Science

Software Integration Testing for SMILE

by Sebastian MIKSCH

The Solar Wind Magnetosphere Ionosphere Link Explorer (SMILE) sets out to examine the interactions between the Sun's charged particles and Earth's magnetic field in unprecedented detail. Four instruments are carried for this purpose: the SXI, the UVI, the MAG, and the LIA. Each of these instruments must be constructed with the utmost care. We built the Test Specification Tool (TST) to assist with the testing of the SXI's Data Processing Unit (DPU). By sending data packets to and from the satellite, the TST checks the output against the ECSS databases to ensure the software functions correctly. A test is split into two parts: the test command and the verification. The user of the TST specifies both parts of the test. After the test is specified, the TST creates the necessary test scripts, which can then be executed. In the Progress Viewer, the TST displays the testing results in a human-readable form. The TST simplifies the creation of the necessary test scripts and automates the testing process as much as possible.

Acknowledgements

This thesis would not have been possible without the support and encouragement of many people. First and foremost, I would like to express my deepest gratitude to my advisors, Roland Ottensamer and Franz Kerschbaum, for their invaluable guidance, patience, and insight throughout my research journey. Their expertise and dedication have been instrumental in shaping this work, and I am profoundly grateful for their mentorship.

I am also deeply thankful to the faculty and staff of the University of Vienna's Institute for Astrophysics, who provided the resources and supportive environment that made this research possible. My heartfelt appreciation also goes to my friends and fellow students, who have been by my side through the long hours, setbacks, and breakthroughs that marked this journey.

A special mention goes to Marko Mecina, whose support and guidance were crucial at every step of my work. Without his technical expertise and patience, this thesis would not have been possible.

I would also like to express my deepest gratitude to my girlfriend, Denise, whose unwavering support and understanding have been my anchor during these trying times. Her encouragement, patience, and belief in me gave me the strength to persevere, even in moments of doubt and exhaustion.

Finally, I would like to thank my family for their unconditional support and patience throughout my studies. Their love and encouragement gave me the strength to keep going, even in the most challenging moments.

Thank you all for helping me reach this milestone.

Contents

List of Figures

List of Tables

List of Abbreviations

ASW	**A**pplication **S**oft**W**are
BSW	**B**asic **S**oft**W**are
BEE	**B**ack **E**nd **E**lectronics
CAS	**C**hinese **A**cademy of **S**cience
CHEOPS	**CH**aracterising **E**x**OP**lanets **S**atellite
CME	**C**oronal **M**ass **E**jection
DBS	**D**PU **B**oot **S**oftware
DPU	**D**ata **P**rocessing **U**nit
ECSS	**E**uropean **C**ooperation for **S**pace **S**tandardization
ESA	**E**uropean **S**pace **A**gency
FDIR	**F**ault **D**etection **I**solation and **R**ecovery
FEE	**F**ront **E**nd **E**lectronics
HEO	**H**ighly **E**lliptical **O**rbit
IASW	**I**nstrument **A**pplication **S**oft**W**are
IBSW	**I**nstrument **B**asic **S**oft**W**are
LEO	**L**ow **E**arth **O**rbit
LIA	**L**ight **I**on **A**nalyzer
MAG	**MAG**netometer
MIB	**M**anagement **I**nformation **B**ase
PLM	**P**ay**L**oad **M**odule
PUS	**P**acket **U**tilization **S**tandard
SCOS 2000	**S**atellite **C**ontrol and **O**peration **S**ystem 2000
SMILE	**S**olar wind **M**agnetosphere **I**onosphere **L**ink **E**xplorer
SSO	**S**un **S**ynchronous **O**rbit
SST	**S**ervice**S**ub**T**ype
ST	**S**ervice**T**ype
SXI	**S**oft **X**-Ray **I**mager
TC	**T**ele**C**ommand
TM	**T**ele**M**etry
TST	**T**est **S**pecification **T**ool
UVI	**U**ltra**V**iolet **I**mager

Chapter 1

Introduction

1.1 Introduction

SMILE is a mission of ESA and CAS. As of the publishing date of this thesis, the mission's launch date is December of 2025.

The main goal of the mission is to learn more about the interaction between Earth's magnetosphere and the solar wind. To this end, the satellite will move around Earth in a highly elliptical orbit, changing from observation modes to data transfer modes periodically.

This thesis concerns testing the software that will run on SMILE's SXI Data Processing Unit (DPU). In the past, tests like the ones described here have been conducted manually, taking up a lot of time and human effort. The focus of this thesis, therefore, lies in automating this test process. To this end, the Test Specification Tool (TST) was created.

This tool can test scripts automatically and run the tests automatically. The testing results are then given in the form of log files that can be easily read and interpreted by humans. After the test scripts have been generated, they can be saved for reuse later. This will significantly reduce the burden on humans involved in the testing process.

The TST is a collaboration of many people. The special focus of this work lies on the generation and execution of telecommands (TCs) and the reception of telemetry packets (TMs).

Starting from these points, I developed a Verification system that creates readable output to judge the outcome of the tests, making the TST easy and comfortable to use.

To this end, the thesis will provide a guided tour of the mission and its software. After this first section (the Introduction), it will examine the scientific objectives that define the satellite's functions.

In the third section, I will delve into the construction of the satellite, providing an overview of the hardware on which the software will be running and which it will control.

The fourth section depicts the software that will be tested using the TST.

In section five, I examine the test process in detail, describing the involved standards and scripts.

Section six provides a detailed description of the software framework for automated testing.

Section seven deals with Verification, how it is depicted in the TST, and how the Verification scripts are generated.

Finally, the eighth section brings chapters 6 and 7 together to demonstrate how fully automated testing works. Lastly, should anyone need information on how to use the TST, a user manual is included in the Appendix.

Chapter 2

Science

2.1 Magnetosphere

The fundamental science of SMILE consists of two main parts: Earth's magnetosphere and the solar wind. The magnetosphere is the area around Earth dominated by its magnetic field. This field takes the shape of a dipole field. It is generated through a dynamo process powered by the liquid metal in Earth's outer core.
The magnetic poles do not coincide with Earth's geographic poles. The magnetic south pole lies near the geographic north pole and the magnetic north pole lies near the geographic south pole. The magnetic field does not remain in place. The poles wander, and every 800.000 years, they shift. The magnetosphere is not empty. It is full of charged particles deflected by the magnetic field. The magnetic induction of a dipole field is as follows:

$$B_D = -\mu \nabla \Phi_D = -\nabla \frac{\mu_0 \vec{M} \vec{r}}{4\pi r^3} \tag{2.1}$$

From this equation (2.1), it is clear that the magnetic field around a dipole takes the form of a sphere. This spherical form around earth is only a theory. Under the pressure of the solar wind, the magnetic field is deformed. The dayside is compressed, and the nightside is elongated. The distance between Earth's magnetic field and Earth's surface changes with respect to which side faces the sun.

The area where solar wind and Earth's magnetic field meet is called the bow shock. Here the solar wind plasma is shocked and runs around the magnetosphere through the magnetosheath. The bow shock is followed by the magnetopause, a sharp transition from dense, shocked solar wind plasma to tenuous magnetospheric plasma.
The cusps are the areas where Earth's magnetic field lines start diverging towards the opposite poles. In these areas, the magnetic field is relatively weak. This allows the solar wind plasma to penetrate Earth's ionosphere., causing auroras.
The form of Earth's magnetosphere, including its day-night deformation and its areas, are pictured in Figure 2.1.

The different measurements of the parts of the magnetosphere are the following (Hanslmeier, 2004a):

- On the sun-facing side: $10 - 12$ R_E
- Over the magnetic poles: 15 R_E

There are two pressures that give the magnetosphere its form:

- The pressure of the solar wind compressing the magnetic field. This pressure changes over time. It depends on the solar activity and the angle α at which the charged particles hit the magnetosphere.

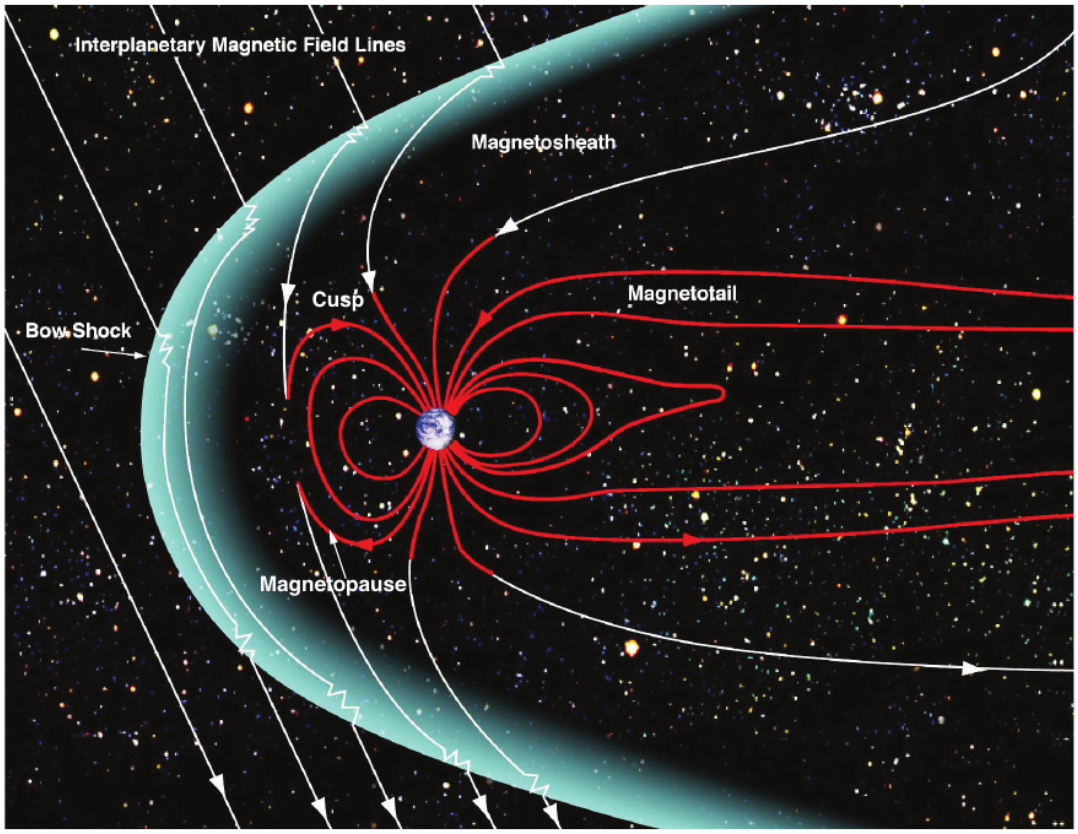

FIGURE 2.1: Earth's magnetosphere as deformed by the solar wind. The different areas of the magnetosphere are clearly marked. (Branduardi-Raymont et al., 2018a)

- The magnetic field pressure working against the solar wind: $\frac{B^2}{2\mu_0}$. This pressure is constant.

The equilibrium condition therefore is:

$$fnmv^2 * \cos^2 \alpha = \frac{B^2}{2\mu_0} \tag{2.2}$$

Examining equation 2.2 shows that the solar wind's strength varies with the number of particles n, the speed of said particles v, and the angle α at which the particles hit the magnetosphere. The form of the magnetosphere is, therefore, highly dependent on solar activity.

Earth's magnetosphere consists of seven parts (Hanslmeier, 2004b):

- Bowshock: The front region. This is where the solar wind particles come into first contact with the magnetosphere.
- Magnetosheath: This region lies between the bow shock and the magnetopause. Particles become thermalized - kinetic energy is turned into thermal energy. The plasma in this region is highly turbulent.
- Tail region (Magnetotail): The elongated magnetic field region at Earth's nightside. The field lines close at large distances ($\sim 3000 r_E$)
- Plasmasheath: A sheet of plasma in the tail regions separating the lobes.
- Lobes: Parts of the magnetotail that have opposite directions and are separated by the plasmasheet.
- Plasmasphere: Torus shaped region surrounding Earth. It has a very sharp edge at the plasmapause, extending to $4 - 6\ r_E$. It can also be regarded as an extension of the ionosphere. Inside the plasmapause the plasmasphere's field lines rotate with Earth. Outside, the pressure of the solar wind's influence is too great. This area is mainly composed of hydrogen.
- Van Allen radiation belts: These belts are shaped like a torus, like the plasmasphere. The inner radiation belt extends from 400 to 12000 km. The outer radiation belt extends from 12000 to 60000 km.

2.1.1 Geomagnetic Storms

Form and position of Earth's magnetopause are always changing. The changes are a product of the changes in the sun's radiation. Additionally to the solar wind dynamic pressure, coronal mass ejections (CME) can disturb and deform the magnetopause. CMEs are fast-moving bursts of plasma. When these bursts hit the magnetosphere, they deform the magnetic field, disturbing its direction and strength. Additionally they induce large electrical currents into the magnetosphere. These currents are called geomagnetic storms. (Branduardi-Raymont et al., 2018b)
Figure 2.2 illustrates the difference of Earth's magnetosphere before (left panel) and during a geomagnetic storm (right panel). As indicated by the color, the pressure of the solar wind plasma against Earth's magnetosphere rises. The heightened dynamic pressure compresses the magnetosphere so that it takes up less than half of its pre-storm form (Branduardi-Raymont et al., 2018b).
During the storm electric currents are induced into the magnetosphere. This can be seen in the bottom right insertions in both panels.

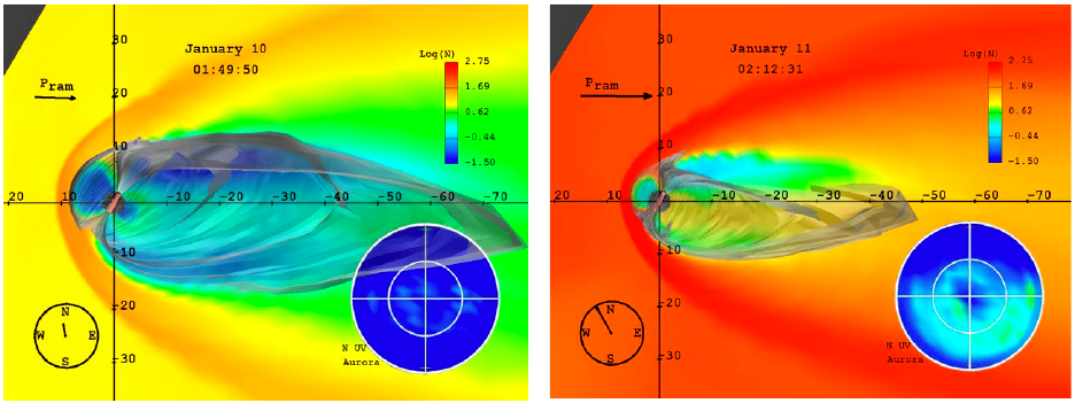

FIGURE 2.2: Magnetohydrodynamics model simulation of Earth during a geomagnetic storm.
Left: Earth's magnetosphere before the storm on 10. January 1997.
Right: Earth's magnetosphere during the storm on 11. January 1997.
(Branduardi-Raymont et al., 2018b)

2.1.2 What is a Substorm

Substorms are also called auroral substorms. This name stems from the fact that these storms can be observed as brightness changes and motion of the aurora. Contrary to the geomagnetic storms, these substorms last only a few hours and are only observable from the planet's poles. Substorms are not dependent on geomagnetic storms and can form without the latter. However, the number of substorms during a geomagnetic storm is higher than without one. The most recent models explain substorms via magnetic reconnection. During this process, regions of opposite magnetic fields come together, and the magnetic field lines can break and reconnect in new combinations. During this breaking and reconnecting, energy is released from the magnetotail. (Hanslmeier, 2004c)
During substorms, magnetic flux is taken from the dayside magnetosphere and added to the magnetotail lobes.

The process of how substorms actually come to be is largely unknown.

2.2 Solar Wind

The sun is continuously losing mass. This mass radiates into space and is called the solar wind. The solar wind's strength is not steady but varies with solar activity, which changes in an 11-year cycle.
The solar wind consists of charged particles. Mostly electrons, protons, and alpha particles.
In a distance of 1 AU, when the solar wind reaches Earth, these particles have an average speed of $\sim 400\frac{km}{s}$. The sun's average mass loss due to the solar wind is $10^{-14}\frac{M_\odot}{yr}$. This corresponds to $\sim 1 \cdot 10^{12}\frac{kg}{s}$. During its entire lifespan up until now (about $5 \cdot 10^9$ years), the mass loss via solar wind comes to $\sim 1 \cdot 10^{-4} M_\odot$.
The plasma of the solar wind leaves the sun through holes in the corona, where the magnetic field lines extend into interplanetary space and moves along these field lines.
The speed of the solar wind reduces with the distance it travels from the sun. At the heliopause, where the solar wind collides with interstellar radiation, its speed is at about $20\frac{km}{s}$. (Hanslmeier, 2004a)

Another way for the sun to lose mass are the Coronal Mass Ejections (CME)

2.3 Measurements of the Solar Wind - Magnetosphere Interactions

Multiple explanations have been put forward to describe the nature of the solar wind-magnetosphere interaction. Proposed magnetosphere entry mechanisms are as follows: the battering of the magnetosphere by solar wind pressure fluctuations, the Kelvin-Helmholtz instability on the magnetopause, diffusion driven by wave-article interactions, and magnetic reconnection (Branduardi-Raymont et al., 2018b). Proposed magnetotail release mechanisms are as follows: multiple plasma instabilities, e.g., ballooning or cross-tail current-driven instabilities, and magnetic reconnection. (Branduardi-Raymont et al., 2018b)

Different statistical studies point to reconnection as the dominant mode of solar-wind magnetosphere interaction. (Branduardi-Raymont et al., 2018b). A host of plasma instabilities proposed to happen within the near-Earth magnetotail may be caused by reconnection.
Reconnection itself is a microphysical process with macrophysical consequences. (Branduardi-Raymont et al., 2018a) Various spacecraft missions like Themis, the Van Allen Probes, and MMS have been launched to gain a better understanding of reconnection.

To better understand the microphysics of the reconnection process in situ measurements are necessary. The global variations in Earth's magnetic field are a product of multitudinous single reconnection events. Therefore multiple single-point measurements are needed to gather a global understanding of the interaction of solar wind and magnetosphere.
A good way to gather these data would be a flotilla of microsatellites performing said in situ measurements. While this method would deliver a trove of measurements about reconnection, it would also have shortcomings. Namely that microsatellites with their technological limitations (little fuel capacity, small propulsion systems) cannot measure the entire deformed form of the magnetosphere.
To avoid these issues another type of mission is available: Instead of multiple in situ measurements, imagers can supply global measurements.

A good area for global observation of the magnetosphere is soft X-ray radiation. Looking at the magnetosphere in the soft X-ray reveals a set of boundaries within the magnetic field. The boundaries correspond to plasma density structures like the bow shock and the magnetosheath. Soft X-ray observation, therefore, is well suited to observe the location of the magnetosphere's structures and measure their location during geomagnetic substorms. During these storms, the magnetopause erodes, and after the storms it regenerates. The rates and extent of erosion and regeneration allow us to collect information about the steadiness and the single components of reconnection.

A good complement to soft X-ray observations is high inclination, high altitude images of the auroral oval. Its dimensions indicate the open magnetic flux within Earth's magnetotail. The movements of the auroral oval towards the poles and the equator provide insights into frequency and incidence of reconnection events at the dayside magnetopause and within Earth's magnetotail (Branduardi-Raymont et al., 2018b).
One of the hypothesis that can be tested with such a set-up is the onset of substorms

caused by plasma flows. According to this hypothesis a main factor in the build-up of substorms are plasma flows within the magnetosphere. These plasma flows originate in the dayside auroral oval. Due to the pressure exerted by the solar wind, the plasma flows stream across the polar cap, into the nightside auroral oval. This transition triggers the substorm onset Buggey, 2021.

To complete these observations and measurements of solar wind and magnetosphere, it is necessary to observe their interaction from within. Observations from within Earth's magnetosphere help measure the arrival times of possible triggers of magnetospheric events. They also help measure the dimensions of solar wind structures transversing the Sun-Earth line (Branduardi-Raymont et al., 2018b).

2.4 Science of the three Key Questions

Previous missions observed snapshots of magnetosphere, solar wind and their interaction. The goal of SMILE is to take the first global picture with in situ measurement, and observations from inside and outside the magnetosphere.

To prepare the mission, three big questions have been chosen to be answered by SMILE (Branduardi-Raymont et al., 2018b).

- What are the fundamental modes of the dayside solar wind/magnetosphere interaction?
- What defines the substorm cycle?
- How do CME-driven storms arise and what is their relationship to substorms?

2.4.1 What are the fundamental modes of the dayside solar wind/magnetosphere interaction?

Reconnection is a magnetic process occurring in plasma. Two plasma elements are connected via a magnetic field line. If the elements move, they stay connected via that field line. If there is an area of electrically conducting material nearby, that magnetic field line may dissipate, and the connection between the two elements disappears. Other particles nearby may now be connected via the magnetic field line. During this process the topology of the magnetic field changes and part of the magnetic energy is converted into kinetic energy.

Reconnection is a main process for transferring mass and momentum within Earth's magnetosphere and a major driver of large-scale magnetospheric convection. It takes place in the area above the poles, where the magnetic field lines are vertical relative to Earth's surface.

Reconnection can be steady. However, unsteady reconnection also exists. Current hypotheses hold that steady reconnection occurs for low beta solar wind and in the magnetosheath, while unsteady reconnection is thought to occur for high beta solar wind (Branduardi-Raymont et al., 2018b).
Testing this hypothesis is difficult as unsteady magnetopause reconnection may be a direct result of variations in the solar wind. In situ measurements are not well suited to test the hypothesis either, as the energy transfer via reconnection occurs over a large area of the magnetopause.

Reconnection and solar wind pressure have different effects on the form of the magnetopause: Reconnection causes the shape to become blunter. Solar wind pressures cause self-similar changes in magnetospheric dimensions. By measuring the

size and curvature of the magnetopause and by measuring the location, size, and shape of the cusps, the effects of solar wind pressure changes and magnetic reconnection can be distinguished (Branduardi-Raymont et al., 2018b).

The following measurements are required to answer the first key question (Branduardi-Raymont et al., 2018b):

- steady/unsteady solar wind variations
- steady/unsteady motion of the dayside magnetopause
- transient brightenings and equatorward leaps in the dayside auroral oval
- transient brightenings and equatorward leaps in the cusp

2.4.2 What defines the substorm cycle?

The area in which geomagnetic substorms can be found is within the auroral oval. An area of the magnetosphere where the magnetic field lines are vertical in respect to Earth's surface. This allows solar wind plasma to penetrate deep into the ionosphere. The auroral oval can be easily identified by the energetic discharges within it, setting up the border of the polar cap.
The auroral oval changes form and size continuously. This results from the changing plasma flux in the polar cap and the changes caused by the varying solar wind ((Branduardi-Raymont et al., 2018b).
What exactly causes a substorm is contested. Different hypotheses are offered: The substorm could be triggered by changes in the interplanetary magnetic field. (Hsu and McPherron, 2004) Or it could be triggered by dynamic pressure changes of the solar wind. (Hubert et al., 2009)
There remain a number of unanswered questions concerning the substorm. How solar wind drives magnetospheric convection is also not well understood. What controls the evolution of a substorm once it is triggered?

The following measurements are required to answer the second science question (Branduardi-Raymont et al., 2018b):

- location and motion of the dayside magnetopause boundary
- location and motion of the auroral oval
- substorm brightenings of the auroral oval
- solar wind input (Branduardi-Raymont et al., 2018a)

2.4.3 How do CME-driven storms arise and what is their relationship to substorms?

While variations in the solar wind dynamic pressure occur on a daily basis, there are trigger mechanisms for geomagnetic storms that are more uncommon. A major uncommon trigger are the Coronal Mass Ejections (CMEs). CMEs propagate with super-magnetosonic speed relative to the surrounding solar wind. The largest geomagnetic disturbances are associated with CMEs. The strength of these disturbances directly correlates with the speed and strength of the CME. But sometimes CMEs do not have the expected effects associated with geomagnetic storms. Under certain

circumstances, their effect can be weakened or heightened, depending on Earth's magnetic field's northward- and southward components (Branduardi-Raymont et al., 2018c).

Understanding CMEs' effect on Earth's magnetosphere has both scientific merit and practical usage. Understanding how the structure of the CME causes different phases of geomagnetic storms is crucial in understanding the interaction of solar plasma and the magnetosphere. On the practical side, CMEs have a huge influence on magnetopause and ionosphere, posing a threat to human information infrastructure.

A host of questions remain: Is the variation in the solar wind the only indicator whether a storm will occur? What is the relationship between storm and substorm? On the practical side, the question of duration is important. How long does it take for a storm to run its course? Where does the energy causing the storm come from? Does a storm run as long a there is stored energy left the magnetotail or does a storm rage as long as the conditions by the solar wind are given? Once the solar wind condition is removed, how long does it take the magnetosphere to recover (Branduardi-Raymont et al., 2018c)?

To answer these questions and more the following measurements need to be taken (Branduardi-Raymont et al., 2018a):

- location and motion of the dayside magnetopause boundary
- location and motion of the auroral oval
- substorm brightenings of the auroral oval
- solar wind input

Chapter 3

The Mission

3.1 Scientific requirements

The requirements that the mission is supposed to fulfill are derived from three main questions, as defined in the "SMILE Definition Study Report". These three questions are further split into four sub-questions each. The 12 questions derived that way give a clear vision of what the SMILE mission is supposed to accomplish. Thus a picture emerges of what instruments are required and what observations are necessary for the mission to succeed. The three main questions and their sub-questions are pictured in Table 3.1. Each of the shown questions demands a number of scientific and technical requirements, shown in Table 3.1.

Cosmic Vision	**2. How does the Solar System work?**									
SMILE Science Theme	**Dynamic response of the Earth's magnetosphere to the solar wind impact**									
Science Questions		**L1 Requirements**								
		R1	**R2**	**R3**	**R4**	**R5**	**R6**	**R7**	**R8**	**R9**	**R10**
Q1. What are the fundamental modes of the dayside solar wind/magnetosphere interaction?	Steady/unsteady solar wind variations	●									
	Steady/unsteady motion of the dayside magnetopause		●								
	Transient brightenings and equatorward leaps in the dayside auroral oval			●	●						
	Transient brightenings and equatorward leaps in the mid-altitude cusp					●	●				
Q2. What defines the substorm cycle?	Location and motion of the dayside magnetopause boundary							●			
	Location and motion of the auroral oval								●		
	Substorm brightenings of the auroral oval									●	
	Solar wind input	●									
Q3. How do CME-driven storms arise and what is their relationship to substorms?	Location and motion of the dayside magnetopause boundary							●			
	Location and motion of the auroral oval								●		
	Substorm brightenings of the auroral oval										●
	Solar wind input	●									
Optional questions											
Q4. What are the characteristics of the aurora at small scales in the Southern hemisphere?											
Q5. What is the interaction of solar wind at comets and how does this compare with at Earth?											
Q6. How are transpolar cap arcs created during Northward IMF and how do they evolve?											

FIGURE 3.1: The scientific questions and their requirements. (Branduardi-Raymont et al., 2018d)

Scientific Measurement Requirement Level 1	Instrument Requirement Level 2
R1: The solar wind and magnetosheath plasma and magnetic field should be measured at over 2 min. time resolution	**RP1:** H+ moments (density, velocity and temperature) shall be measured at 2 min. or better time resolution with at least 20 % accuracy **RP2:** H+ velocity shall be measured up to 800 km/s **RP3:** The three components of the magnetic field B shall be sampled with at least 2 min. time resolution, 0.1 nT resolution in the solar wind, and an absolute accuracy of 2 nT.
R2: For solar wind flux $> 4.9 \cdot 10^8/\mathrm{cm}^2\mathrm{s}$, the location of the subsolar magnetopause shall be determined with an accuracy better than 0.5 Earth radii (R_E) and better than 5 min. time resolution from locations on orbit greater than 15 R_E geocentric	**RP4:** X-ray images of emission gradients corresponding to the magnetopause shall be obtained with $1.5^\circ (= \arctan(0.5R_E/20R_E))$ or better angular resolution at least every 5 min. for solar wind flux $> 4.9x10^8/\mathrm{cm}^2\mathrm{s}$ from locations on orbit greater than 15 R_E geocentric
R3: Transient brightenings of 100 R or more in the dayside aurora shall be measured with a time-resolution of at least 1 min	**RP6:** UV images shall have a sensitivity of 100 Rayleigh (R) or better at 1 min resolution
R4: The position of features in the aurora shall be measured with an accuracy of at least 150 km at 1 min. time resolution from locations on orbit greater than 15 R_E geocentric.	**RP7:** Global UV images of the aurora shall have 150 km or better spatial resolution at 1 min. time resolution from locations on orbit greater than 15 R_E geocentric **RP8:** Auroral structures (e.g. arcs) shall have a position accuracy of 500 km with respect to the Earth's surface
R5: The poleward and/or equatorward edges of the mid-altitude cusp shall be determined with a spatial resolution of at least 0.25 R_E and a time resolution of at least 5 min for solar wind flux $> 4.9 \cdot 10^8/\mathrm{cm}^2\mathrm{s}$ from locations on orbit greater than 15 R_E geocentric.	**RP9:** The poleward and/or equatorward edges of the mid-altitude cusp in X-ray images shall be identified to within $0.75^\circ (\arctan(0.25R_E/20R_E))$ angular resolution at time resolution of at least 5 min for solar wind flux $> 4.9 \cdot 10^8/\mathrm{cm}^2\mathrm{s}$) from locations on orbit greater than 15 R_E geocentric

R6: The cadence and spatial resolution of the soft X-ray imager shall suffice to determine angular equatorward/poleward motions of the mid-altitude cusp at speeds between 0.1 and 0.5° (ILAT)/min.	**RP10:** X-ray images of the mid-altitude cusp shall have a spatial resolution of $0.4^\circ (= \arctan(900\text{km}/20\text{R}_E))$ or better at temporal resolution of 4 min. from locations on orbit greater than 15 R_E geocentric and for solar wind flux $> 9 \cdot 10^8/\text{cm}^2\text{s}$.
R7: For solar wind flux $> 3.6^8/\text{cm}^2\text{s}$, the location of the subsolar magnetopause shall be determined with a spatial accuracy better than 0.5 R_E and a time resolution better than 10 min. from locations on orbit greater than 15 R_E geocentric	**RP11:** The position of the subsolar magnetopause shall be identified with at least $1.5^\circ (\arctan(0.5\text{R}_E/20\text{R}_E))$ angular resolution at time resolution of 10 min. from locations on orbit greater than 15 R_E geocentric
R8: The poleward and equatorward boundaries of the auroral oval shall be identified at all local times with a spatial resolution of at least 300 km at 1 min. time resolution from locations on orbit greater than 15 R_E geocentric	**RP12**: UV auroral images with better than 300 km spatial resolution shall be obtained at 1min time resolution from locations on orbit greater than 15 R_E geocentric
R9: Transient brightenings of 100 R or more in the aurora shall be measured with a cadence of at least 1 min.	**RP6:** UV images shall have a sensitivity of 100 R or better at 1 min resolution
R10: 200 R changes in brightness of 30 kR aurora shall be identified and measured with a cadence of at least 1 min.	**RP13:** UV variations of as little as 200 R upon backgrounds of 30 kR shall be obtained at 1 min time resolution
R2, R5, R6, R7	**RP14:** X-ray images in the energy band 0.2-2.5 keV shall be obtained **RP15:** The field of view of SXI shall be at least $26.5 \cdot 15.5^\circ$

TABLE 3.1: The scientific measurement requirements (Level 1) and their traceability to the payload performance requirements (Level 2) (Branduardi-Raymont et al., 2018d)

3.2 Mission requirements

Since the mission's main objective is to gain scientific insight, the mission requirements are designed with the scientific objectives in mind. This ensures that the main questions can be answered and that the chosen instruments work as efficiently as possible.

Table 3.2 presents the major mission requirements.

For an orbit to fulfill these requirements, the spacecraft has to traverse both the magnetopause and bowshock (see Chapter 2). An orbit shaped in this way will permit the orbiter to measure the solar wind at its apogee and take images of the cusps/magnetosheath. The orbit also allows the spacecraft to measure the solar wind inside the magnetopause. This leads to a direct comparison of the solar wind strength hitting the earth with the solar wind strength hitting the bow shock. An example of such an orbit is given in Figure 3.2.

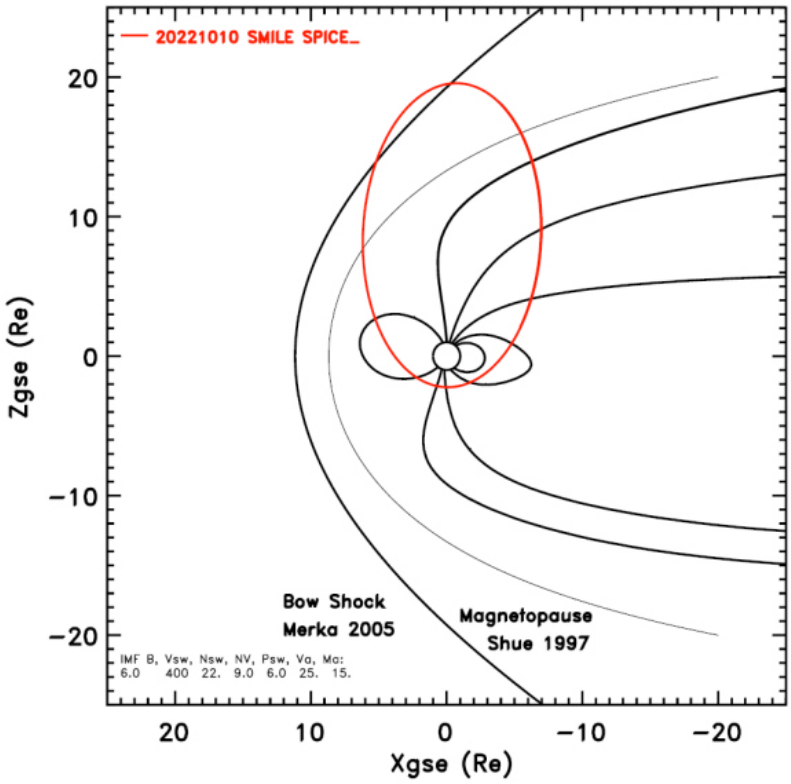

FIGURE 3.2: One of two possible orbits for SMILE. This highly elliptical orbit with an apogee of 127,000 km is inclined by 98° inclination. (ESA, 2019b)

Mission requirements
R11 Orbit requirements
• The apogee of the orbit shall be 20 Re +/- 2 Earth radii geocentric distance • The perigee of the orbit shall be above 5000 km altitude • The inclination of the orbit shall be 90 +/- 27° • The argument of perigee shall be in the range 225-315°
R12 pointing requirements
The spacecraft pointing accuracy shall be: • Absolute Pointing Error (0.5°), • Attitude Knowledge Error (0.5°), • Attitude stability 0.05°/1 min (UVI), 0.5°/10 (SXI) or better
R13 Electromagnetic compatibility (EMC) requirements
• The magnetic disturbance from the spacecraft shall be less than 10 nT DC at the outboard MAG sensor • The absolute spacecraft surface potential relative to the surrounding plasma shall be less than 30V in the solar wind and magnetosheath

TABLE 3.2: Mission and spacecraft level requirements (Branduardi-Raymont et al., 2018d)

3.3 The spacecraft

The SMILE spacecraft consists of two major parts: the payload module at the top and the platform at the bottom.
Both modules combined stack to a height of 3.15m, which is compatible with both the Ariane 6 and the Vega-C rockets. The platform is 3-axis stabilized and provides the spacecraft with resources like fuel and cooling agents. It also contains the thruster and, therefore, gives the spacecraft the means to maneuver in space. Thus, it is the platform that is responsible for raising the spacecraft's orbit from its insertion orbit (SSO or LEO) to its science orbit.
At the base of the platform, four fuel tanks are fastened. Each of these tanks contains 350 l of fuel, leading to a combined weight of 2200 kg.
Below the fuel tanks the engine, with a thrust of 490 N is placed. The highest reachable value of the apoapsis that can be reached lies at 121,000 km. The platform is constructed from aluminum and carbon fiber to satisfy the demands of both stability and lightness.
The platform further provides space for 2 unfoldable solar panels. At their deployed state, these panels will cover an area of 5.8 m². The power generated will supply a set of lithium-ion batteries with a total capacity of 60 Ah. At full capacity, the panels will generate 850 W of electricity.
More utilities fixed to the platform are two-star trackers and two antennas to send signals between the spacecraft and ground control. For more space maneuverability, twelve small thrusters are attached to the platform. Each of these thrusters is capable of delivering forces of up to 10 N.
A single 20 mm thick, 1 m by 1 m aluminum honeycomb sandwich plate forms the platform's top cover and the payload module's base plate.
The platform is constructed by the Chinese Academy of Science (CAS) and is depicted in image 3.3 (ESA, 2020). The DPU is also located on the platform.

The payload module consists of three instruments: the imagers SXI and UVI and the in situ MAG. While the entirety of both SXI and UVI is fixed to the payload module's main platform, the magnetometer's two sensors are positioned on a boom

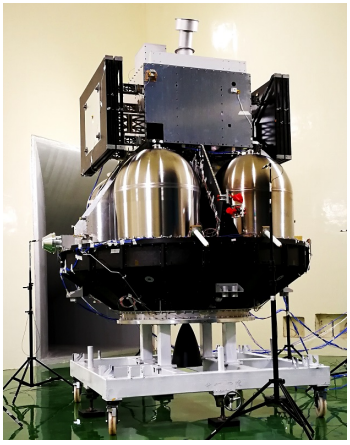

FIGURE 3.3: SMILE platform model. (ESA, 2020)

with an overall length of 3 m. This boom is necessary as the spacecraft itself generates magnetic fields that could disturb the sensors' measurements. To accurately measure fluctuations in the magnetic field, the distance of the two sensors is 80 cm. Similar to the MAG, the SXI has an overall height of 95 cm and contains a baffle to prevent light pollution from the spacecraft itself.
The LIA is the only instrument not fixed to the payload module. Instead, it is built into the platform.
The payload module and all instruments are constructed by ESA and its partners (ESA, 2020). The main duties of the PLM are (Branduardi-Raymont et al., 2018e):

- Command and control functions for the PLM units and the hosted instruments
- Interface with the platform for commanding, FDIR, PF telemetry, and power
- Storage of science data from the instruments and telemetry from the platform which the X-band downlinks
- power distribution to all PLM units on the PLM
- high data rate (∼Gb/orbit) X-band transmission to the ground
- design and implementation of the PLM thermal control and mechanical (Branduardi-Raymont et al., 2018f)

The entire SMILE spacecraft with both payload module and platform is depicted in Figure 3.4.

3.4 Payload and Performance

This section details the payload and instruments of SMILE.

To tackle the three key questions discussed in Chapter 3.1, the mission is equipped with four scientific instruments:

- Soft X-ray Imager (SXI)
- Ultraviolet Imager (UVI)
- Magnetometer (MAG)

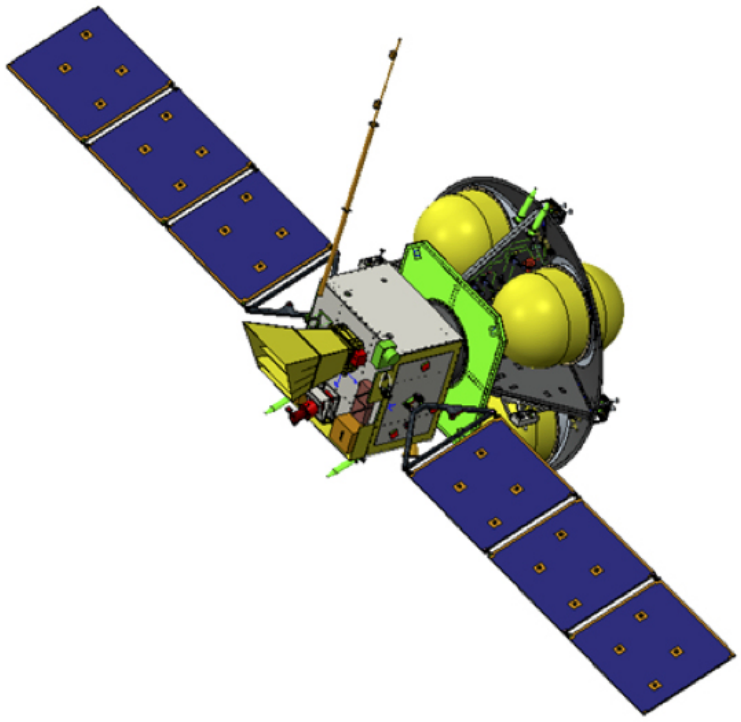

FIGURE 3.4: The complete SMILE spacecraft. Both platform and payload module are depicted. (ESA, 2020)

- Light Ion Analyzer (LIA)

With these instruments, the mission will inspect both Earth's magnetosphere and the solar wind. To meet this demand, the instruments have different points of observation: The SXI and the UVI will observe Earth's properties, while the MAG and the LIA will observe the solar wind's properties. The SXI will create a spectral map of the magnetopause, the magnetosheath, and the magnetospheric cusps, while the UVI is dedicated to imaging Earth's auroral regions. LIA and MAG will measure the solar wind properties together with the imaging instruments.

3.4.1 The Soft X-Ray Imager

The SXI was developed by the University of Leicester with contributions from institutes in the UK, Austria, Hungary, Spain, Czechia, Norway, Ireland, and China. It will observe the magnetospheric boundaries, which include the bow shock, magnetopause, and cusp region in the soft X-ray band. It will also observe the time-dependent solar wind composition from its emission lines (Branduardi-Raymont et al., 2018g).

SXI Instrument Description

The SXI is shown in Figure 3.5.

It is built from four major hardware units (Branduardi-Raymont et al., 2018g):

- SXI Telescope
- Thermal Control System
- Front End Electronics Assembly

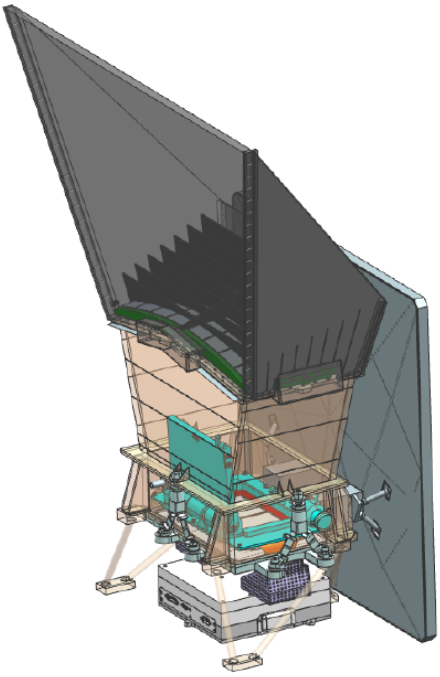

FIGURE 3.5: Model of the SXI and Front End Electronics. At the top of the instruments sits the Straylight baffle (black), followed by the Telescope Tube (orange-brown). The CCD detector sits at the bottom of the Telescope (turquoise) right above the FEE. ((Branduardi-Raymont et al., 2018g))

- Back End Electronics Assembly

The radiation is registered by CCD detectors sitting at the bottom of the Telescope tube. Each of these detectors is connected to the Front Ende Electronics (FEE) Box via flex cables, protected by EMC shields. The FEE is connected to the Back End Electronics (BEE), which mainly consists of the DPU and its Software (OBSW).
Due to the sensitivity of the Instrument, the radiation shutter will be closed at altitudes below 50,000 km. During one orbital passage, the Instrument will be closed for about 9 hours.
The SXI has a nominal mass of 35.66 kg and, depending on its usage, consumes between 17 and 66 W. Its nominal operating power is 51 W (Branduardi-Raymont et al., 2018g).

3.4.2 The Ultraviolet Imager

The UVI will monitor Earth's northern aurora. This observation aims to better understand the processes solar wind particles go through whenever Earth's magnetic field diverts them into Earth's atmosphere near the poles. The instrument is designed to obtain UV images of the aurora even in sunlit regions and conditions. The UVI was developed by the University of Calgary and the Canadian Space Agency, with contributions from Belgium and China. The instrument has an overall mass of 15.73 kg and an average power consumption of 20.3 W. The instrument is pictured in Figure 3.6. (Branduardi-Raymont et al., 2018h)

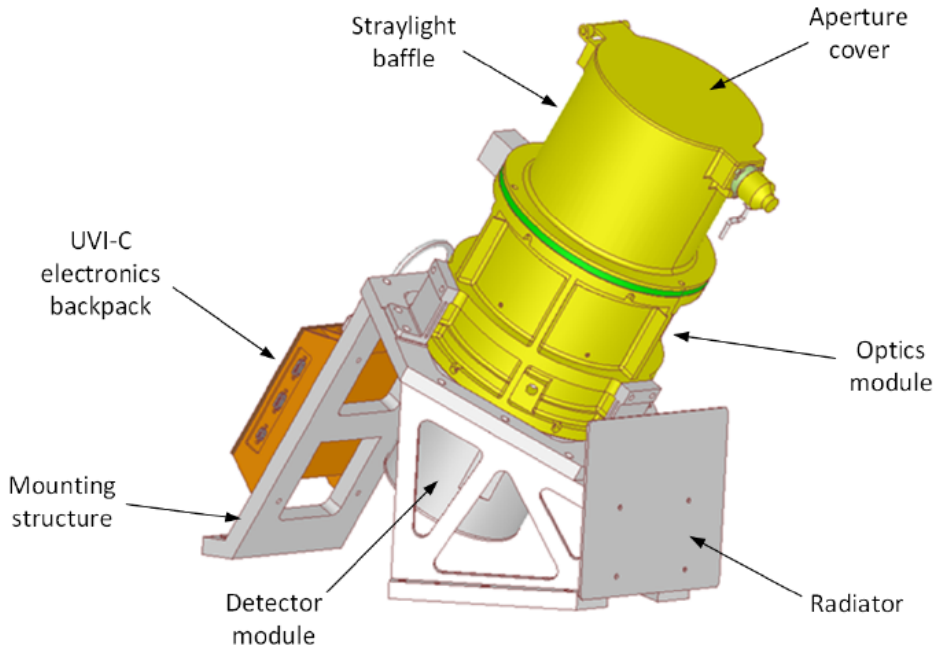

FIGURE 3.6: Model of the UVI. (Branduardi-Raymont et al., 2018h)

3.4.3 The Magnetometer

The MAG consists of two sensors fixed to a 3-meter-long boom. Each of the sensors is comprised of three ring-core fluxgate sensors oriented along three orthogonal axes. This allows us to sample the three spatial components of the local magnetic field. The sensors are mounted on an aluminum platform and are covered with a polyimide lid. The design of the MAG sensor heads is shown in Figure 3.7 (Branduardi-Raymont et al., 2018i).
The MAG boom is divided into three segments. When deployed, the segments build one large connection. Unfolded, the boom has a total length of 3028 mm. Along the boom the sensor heads are mounted so that one sits at the tip of the deployed boom at 3028 mm. The position of the second sensor is around 2185 mm from the boom mounting (Branduardi-Raymont et al., 2018i).
No part of the MAG sensors requires thermal control.

During launch the MAG boom will be folded and held down by a set of two flexible hold-down and release mechanisms (FHRM). 40 days after launch, after the solar panels have been deployed, the boom will unfold, and the MAG will start its observation. (Branduardi-Raymont et al., 2018a)
The Boom is shown in Figure 3.8

The instrument's total mass is 8.7 kg. During nominal science operations, the MAG consumes an average of 4.17 W.

3.4.4 The Light Ion Analyzer

The National Space Science Center in China developed the LIA with contributions from the UK and France. It is an in-situ plasma analyzer to measure the solar wind and magneto sheath ion distributions (Branduardi-Raymont et al., 2018j). It measures the three-dimensional velocity of the ions. The measurements are taken at all

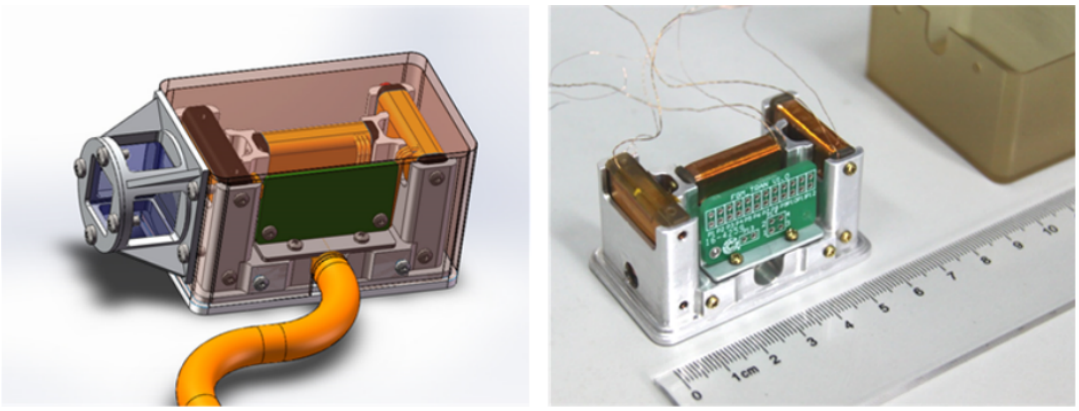

FIGURE 3.7: One of the two identical MAG sensor heads. Left panel: 3D rendition from CAD model. Right panel: Finished engineering model. (Branduardi-Raymont et al., 2018i)

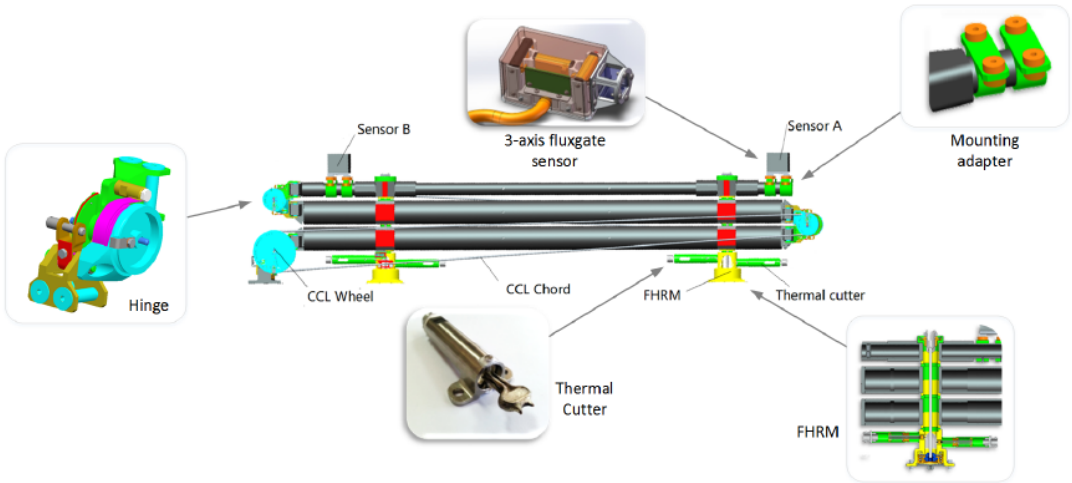

FIGURE 3.8: Folded MAG boom. Some of the main elements are highlighted. (Branduardi-Raymont et al., 2018a)

points along the orbit. This makes it possible to create a three-dimensional distribution of the solar wind and the magnetosheath along the spacecraft's route (ESA, 2019a).

3.4.5 The Data Processing Unit

The Data Processing Unit (DPU) is a central part of each instrument. The UVI, SXI, and MAG each have their own DPU, acting in a similar role. Since this thesis concerns the DPU of the SXI, any subsequent mention of a DPU, concerns said component of the SXI. The basic functioning of the DPU is as follows: It receives telemetric signals (TM) from the instruments and sends it together with the housekeeping data to the spacecraft's mass storage. To ensure instrument safety, the DPU changes the modes of operation (e.g.: entering Science Mode, closing the Radiation Shutter, ...). At the same time, the DPU processes telecommands (TC) from ground control and reduces observational data to be sent back to Earth. For a closer look at how the software running on the DPU hardware works, see Chapter 4. A prototype of the DPU is depicted in Figure 3.9.

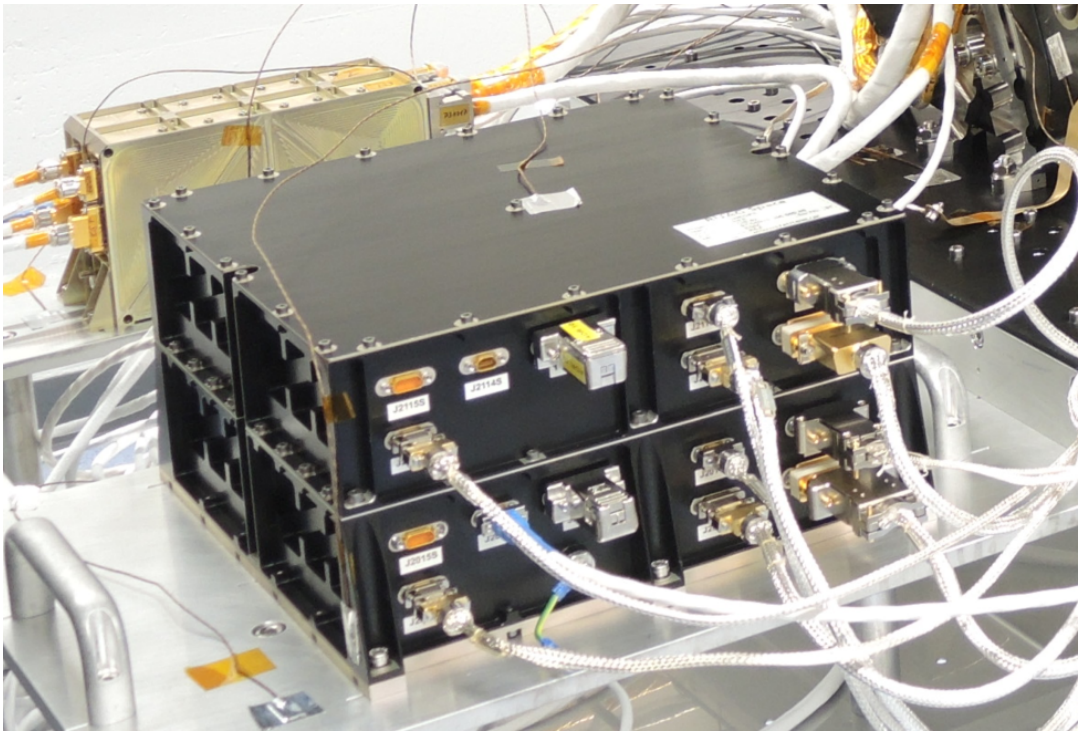

FIGURE 3.9: A prototype of the DPU at the IWF Graz

3.4.6 Mission Definition

SMILE will launch with the VEGA-C rocket, which takes the satellite to a low earth orbit. The launch will take place at the Centre Spatial Guyanais northwest of Kourou, French Guinea. (ESA, 2024)

The satellite will be brought to a circular LEO with a height of 700 km. There, the spacecraft orbits earth in a LEO and waits for a maximum of 6 months in order to reach the Right Ascension of the Ascending Node range.
The planned argument of perigee lies around 288° to allow communications with the baseline ground station in Antarctica (Liu et al., 2024).

Once the Argument of perigee and the Orbital Nodes have reached the necessary values, the satellite starts to raise its apogee and perigee. Every time the spacecraft reaches its perigee near Antarctica the main engine gives a burn, raising the orbit. After 18 burns and 18 orbits, the final orbit is reached. This orbit is no longer circular. It is a highly elliptical orbit (HEO) and also referred to as Science orbit (Branduardi-Raymont et al., 2018f).
After arriving at this orbit, the spacecraft starts a two-month commissioning period. After this period the satellite takes up its scientific operations on its 51-hour orbit. (Branduardi-Raymont et al., 2018a)
The nominal science operation of the satellite can be chunked into 5 mission-level modes (Branduardi-Raymont et al., 2018f):

- Full Science Mode: All 4 instruments are operating
- Reduced Science Mode 1: The UVI is not operating. This is due to Sun exclusion constraints.
- Reduced Science Mode 2: The SXI is not operating due to Sun exclusion constraints or because the spacecraft is below 5000 km.
- Reduced Science Mode 3: Only the in situ instruments, MAG and LIA are operating.
- X-band Communication Mode: No instruments are operating. PLM transmits in the X-band all collected science and telemetry data of the last orbit to the baseline ground stations.

The 18 iterations needed to enter the HEO are pictured to the left in Figure 3.10. The two pictures to the right show the satellite entering Science operation mode.

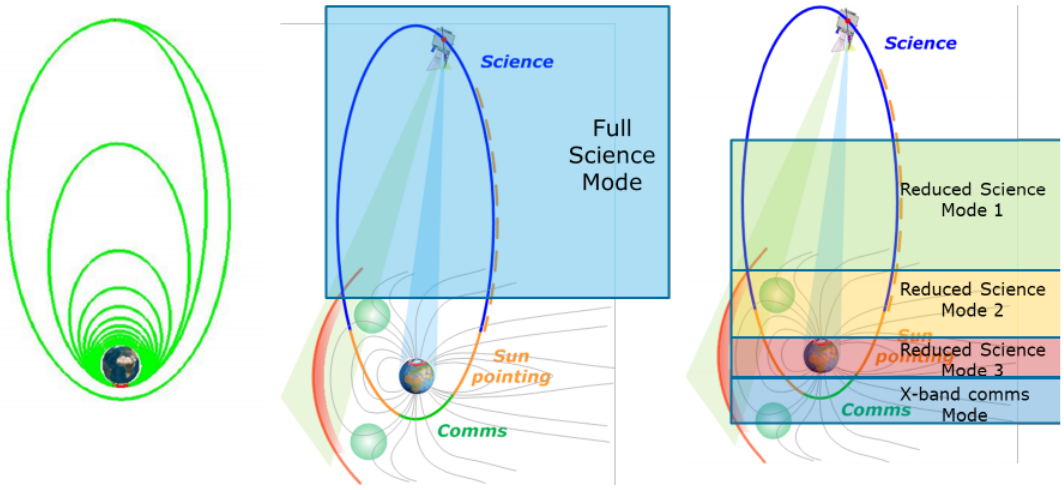

FIGURE 3.10: The science orbit of SMILE
Left: The different orbits the spacecraft goes through before reaching its science orbit. The red near the perigee shows the burn phases by the main engine.
Middle and Right: The different science operation modes and how they correspond to the science orbit. (Branduardi-Raymont et al., 2018f)

The HEO brings with it a number of advantages and challenges. Firstly, this orbit accommodates the high ambitions of the SMILE project. Its orbit leads through the

magnetopause, allowing to measure the solar wind outside Earth's magnetosphere. At the same time, it allows simultaneous observation of the magnetosheath, polar cusps, and aurora with X-ray, UV, and in-situ measurements (Branduardi-Raymont et al., 2018a).
Secondly, the high ellipticity of the orbit leads to long continuous observation times. Overall, SMILE will spend 40 hours of a 51-hour orbit in one of its science modes. (CAS, 2022)
Just as Kepler's third law leads to long observation times during orbit, it also leads to a very short time during which the spacecraft is near enough to transmit data, leading to a challenge in data compression.

Chapter 4

The Software

4.1 Software Architecture

This section details the architecture of the DPU software of the SXI of SMILE.

4.1.1 Basic Structure

The structure of the software of the DPU is modeled on the software used for CHEOPS's DPU. (Willy Benz, 2013) It consists of three basic layers. Each of which handles different tasks. A diagram of the software of the DPU of the SXI is shown in Figure 4.1.

The three basic layers are:

- Instrument Application Software (IASW)
- Basic Software
- Boot Software

The DPU's software is designed in three interconnected layers: the Boot, Basic, and Application Software. The hardware on which these software layers run is the DPU Processor. Once the DPU has been started, the application and basic software run continuously. The Boot Software, on the other hand, only runs sporadically. Its main task is to start the instrument in a safe condition. The Boot Software runs in two instances: When the Basic Software is booted and when it is shut down. While the Basic Software runs, the Boot Software is not active.
The startup of the software follows the same pattern:
As soon as the DPU processor is started, the software layers are initialized in order of support: The startup process starts with the boot software and ends with the IASW.

The Basic Software provides three services:

- the operating system
- a middleware to communicate with the FEE and the PLM ICU
- the DPU hardware management functions

The top layer consists of the IASW. It implements the high level services of the software. Overall the IASW has 5 interfaces: two physical interfaces and three functional interfaces. These functional interfaces are toward the Ground, the FEE, and the PLM ICU. (Christian Reimers, 2019a) The communication between service-provider and service-user is implemented via the PUS. The physical interfaces build the communication to the Front End Electronics (FEE) and the PLM ICU. This connection is built upon Space Wire. (Christian Reimers, 2019a)

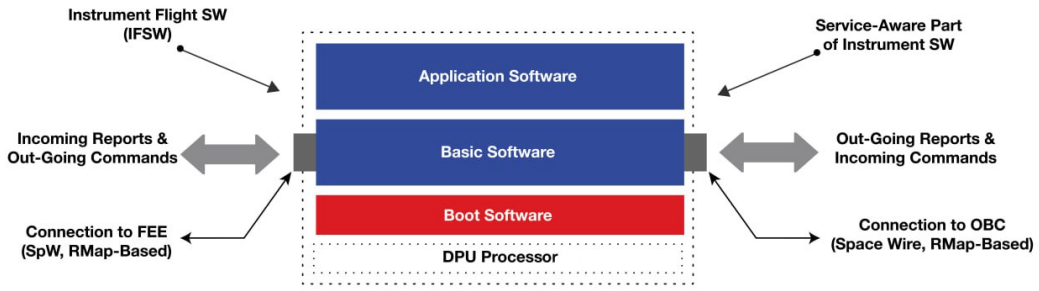

FIGURE 4.1: The structure of the software running on the DPU. The dependency of the respective layers of software is visible.

4.1.2 The IASW

As the focus of this work is testing the IASW according to the ECSS standards, its architecture and interfaces are described more detailed than those of Basic or Boot software.

The IASW takes the form of several state machines. A logical model of the IASW is given in Figure 4.2. The instrument basic software (IBSW) sets up the hardware and schedules the IASW. The specification of the IASW is implemented using the CORDET framework (a framework originally developed for CHEOPS, predefining certain functions of applications. See Subsection 4.1.2 and (Cechticky, Ottensamer, and Pasetti, 2015)). The hardware used to run the IASW has two cores: core0 and core1. Core0 runs the "system" tasks, which consist of the IBSW and the IASW Framework. Science Data Processing is run on Core1. Memory access is managed by the IBSW using MMU functionality.

While core0 runs permanently to keep the IBSW and IASW running, science data processing is only run sporadically, once enough data have accumulated.

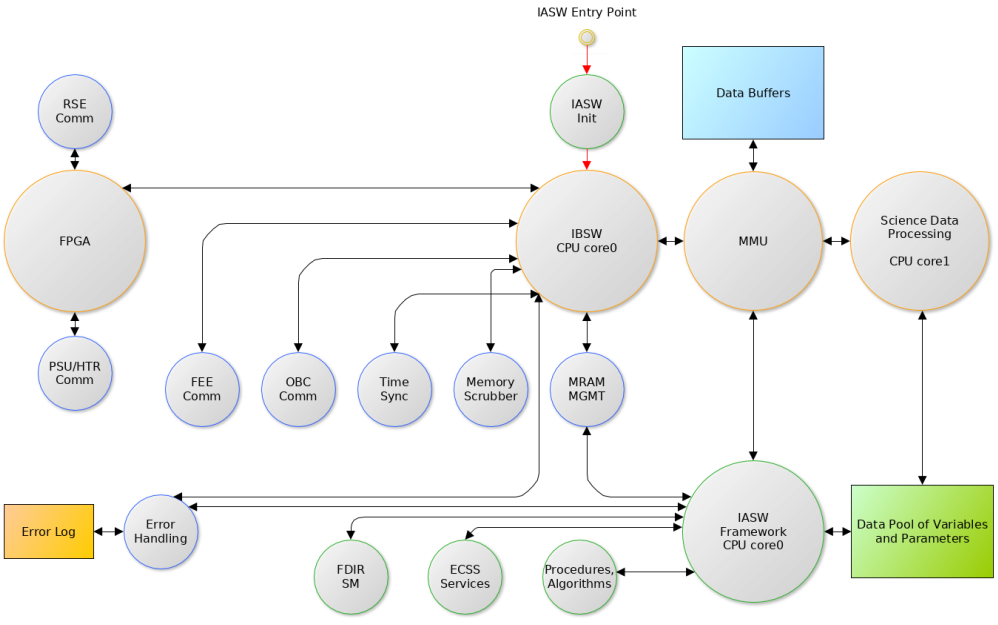

FIGURE 4.2: Logical Model of the IASW. (Christian Reimers, 2019a)

General Software Design

The IASW consists of several software entities:

- Framework - The ECSS services implemented using the CORDET framework
- SDP - The science data processing software takes care of on-board processing tasks, e.g. data compression

SW Function and Purpose

The main functions of the IASW are the following:

- Maintain the health of the instrument
- Command the FEE to acquire images from the CCD
- Command the FEE to carry out event detection
- Store and manage telemetry and data acquired from the FEE
- Store and manage instrument HK telemetry
- Provide data compression for acquired science data
- Provide timestamp information to the instrument telemetry and HK
- Maintain logs of instrument activity and faults
- Perform fault detection isolation and recovery (FDIR)
- Control the opening and closing of the telescope door
- Calculate and provide thermal control to the instrument heaters
- Support diagnostics on instrument hardware
- Support on-board diagnosis of instrument hardware (Roland Ottensamer, 2017)

The services provided by the IASW are given in Table 4.1. Services below 191 are general services provided to the entire system. The services 191 and higher are private services, provided specifically to the SXI. (Roland Ottensamer, 2017)

The IASW has a few core functions it needs to accomplish:

- the processing of incoming commands from the PLM ICU
- the processing of incoming packets from the FEE
- the generation of out-going reports to the PLM ICU
- the generation of out-going packets to the FEE

Overall the functional architecture of the IASW consists of three kinds of components:

- Data Structure Components: encapsulate access to IASW data structures
- IASW Specific Components: encapsulate groups of logically related state machines
- Framework Components: are obtained as customization of the generic components of the CORDET Framework

Type	Name	Description
1	Ver	Request Verification Service
3	Hk	Housekeeping Service
5	Evt	Event Reporting Service
6	Mem	Memory Management Service
9	Time	Time Management Service
13	Ldt	Large Data Transfer Service
17	Tst	Test Service
20	ParamMngt	Parameter Management
191	Fdir	FDIR Service
193	Mode	IASW Mode Control Service
194	Algo	Algorithm Control Service
197	Boot	Boot Report Service
198	Proc	Procedure Control Service
210	Mngt	DPU Management Service
211	Param	Parameter Update Service
212	Data	Data Operation Service
213	Maint	SW Maintenance Service

TABLE 4.1: PUS services implemented in the IASW.

Software Memory Allocation

Overall the DPU provides 256 MB of RAM to accomplish its tasks. The memory is allocated in the following way:

- DBS Area (252 KiB): Used exclusively by the DPU Boot Software
- Exchange Area (4 KiB): Reserved for the data exchange between the DBS and the IASW.
- IASW Area (255.75 MiB): Used exclusively by the IASW

The DPU memory map is given in Figure 4.3

Data Structure Components

Data Structure Components define access to the data structures used by the IASW. Only one such component exists in the IASW, the CrIaDataPool component. This component defines the access to the Data Pool.
The Data Pool is a database in which sets of variables and and parameters are collected to give an overview of the IASW state.

IASW-Specific Components

These components consist of groups of logically related state machines or procedures. Two variable types represent them: FwSmDesc_t for state machines and FwPrDesc_t for procedures.

CORDET Framework Components

The IASW is an instantiation of the CORDET Framework. Said instantiation was originally developed for the CHEOPS mission. (Cechticky, Ottensamer, and Pasetti,

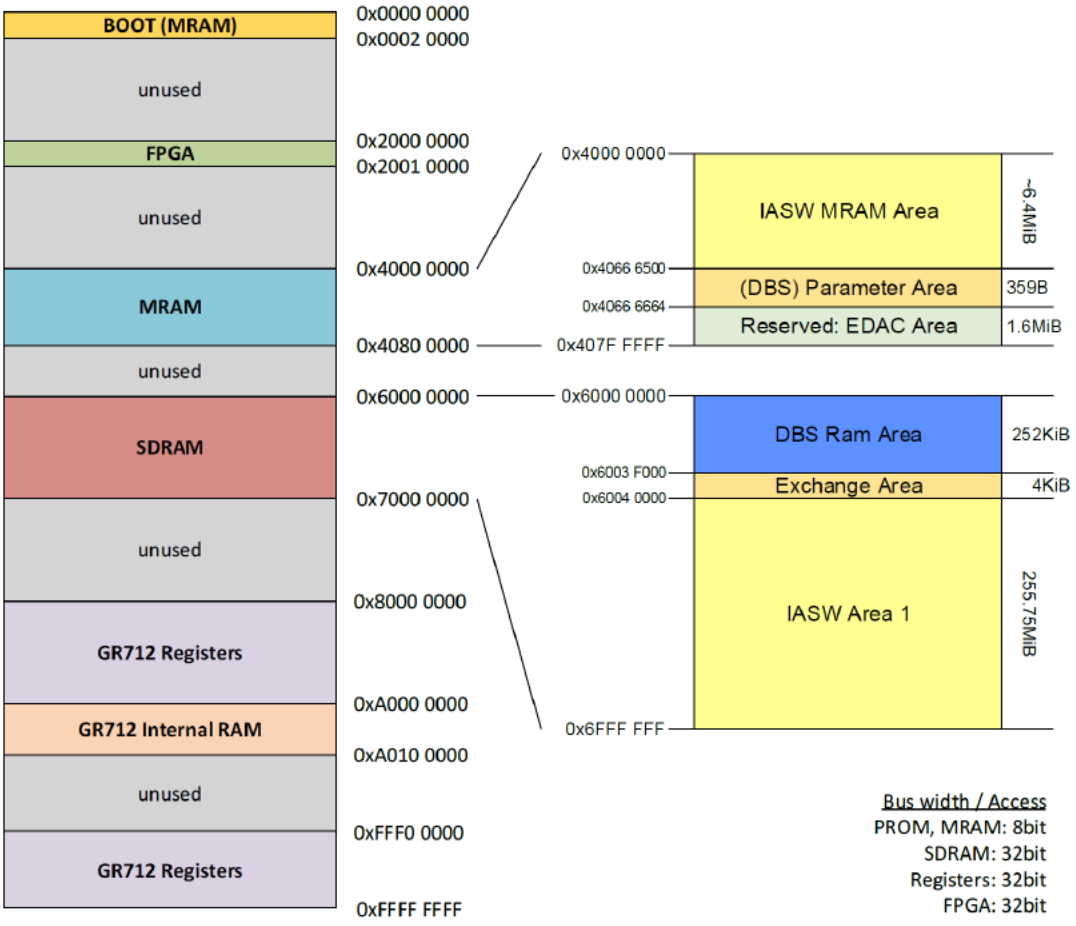

FIGURE 4.3: DPU memory allocation. (Christian Reimers, 2021)

2015) The defined components given within the Framework are used to receive, send, distribute, and process commands and reports. (Vaclav Cechticky, 2022).
The CORDET Framework provides more facilities for initialization, reset, and shutdown of applications. However, these facilities are not used, as the IASW is initialized by the BSW (see fig. 4.1).

The components are modeled as state machines and procedures, and the services and sub-services used in this project and described in 5.2 are derivatives of the components supplied by CORDET.

The BSW and ASW are bundled into the single executable IASW, which is started by the DPU boot Software (DBS). The boot process starts after the DPU receives command (210,4) from the PLM ICU from Ground Control. The BSW initializes the IASW.

During its own initialization, the BSW performs the following tasks:

- Initialization of OS-level services
- Initialization and configuration of SpaceWire interface

When the IASW starts operating, the communication links to the PLM ICU are ready and all lower-level services are available.

The IASW itself is unable to perform an orderly reset or shutdown. The only way to bring the IASW to fulfil these operations is via the DPU. Resetting the software is only possible with resetting the DPU processor.

Cyclical Activities

The IASW runs through a loop of commands with a frequency of 8 Hz. This is the timeframe within which new commands can be accepted and executed. Within one cycle, 18 activities are executed. Table 4.2 gives a list of which.

Command Output Reporting

The IASW accepts new commands in four stages:

- command acceptance
- command start
- command progress
- command termination

Each of these phases ends either in success or failure. Depending upon the step at which an error occurred or did not occur, the IASW sends different TMs to help diagnose the error. Whenever an error occurs, it is unconditionally reported. If the command terminates successfully, it is only reported if requested in the command itself. The following actions are taken when a command passes one of the stages:

- TM(1,1) is sent when the command has cleared the command acceptance step successfully.
- TM(1,2) is sent when an error occurs at the command acceptance step. The `NOfAccFailedTc` counter is incremented by 1.

N	Cyclical Activity
1	Execute the FEE Unit State Machine
2	Execute, in sequence, each FdCheck and its associated Recovery Procedure; The Recovery Procedure is executed immediately after the FdCheck to which it is associated
3	Execute the IASW State Machine
4.1	Execute all Science Procedures
4.2	Execute the non-Science Procedures
5	Execute all the Cyclical Algorithms
6	Execute the Update operation on the Data Pool and FEE Register Map Store.
7.1	Execute CrFwInStreamPcktAvail on InStreamPlm
7.2	Execute CrFwInStreamPcktAvail on InStreamGrd
8.1	Execute the InLoader on InStreamPlm
8.2	Execute the InLoader on InStreamGrd
9	Execute the InManagerGrdPlm
10	Execute the OutManager1
11	Execute SDU State Machine
12	Execute the OutManager2
13	Execute SDU State Machine
14	Execute the Watchdog Reset Procedure

TABLE 4.2: Cyclical Activities (Christian Reimers, 2019b)

- TM(1,3) is sent when the command has cleared the command start step successfully.
- TM(1,4) is sent when an error occurs at the command start step. The `NOfStartFailedTc` counter is incremented by 1.
- TM(1,7) is sent when the command has cleared the command termination step successfully.
- TM(1,8) is sent when an error occurs at the command termination step. The `NOfTermFailedTc` counter is incremented by 1.
- No TM is sent when the command progresses successfully.
- No TM is sent when the command progresses unsuccessfully.

A comprehensive list of responses is given in Table 4.3. The classification of SSTs follows the basic pattern that odd numbers indicate the success of the singular steps necessary for accepting and executing a command. Even numbers, on the other hand, indicate failure.

TM[ST, SST]	meaning
TM[1,1]	successful acceptance verification report
TM[1,2]	failed acceptance verification report
TM[1,3]	successful start of execution verification report
TM[1,4]	failed start of execution verification report
TM[1,5]	successful progress of execution verification report
TM[1,6]	failed progress of execution verification report
TM[1,7]	successful completion of execution verification report
TM[1,8]	failed completion of execution verification report
TM[1,9]	successful command termination report
TM[1,10]	failed routing verification report

TABLE 4.3: All possible TM answer packets to a TC requesting a verification report. (ECSS, 2016a)

Chapter 5

Testing

5.1 The ECSS

Space projects in Europe are generally a collaboration of multiple entities. As mentioned in Section 3.3, multiple organizations and laboratories collaborate to create the SMILE mission. In order for this collaboration to be successful ESA has created a set of standards to which each participant in a ESA space mission has to adhere. The European Cooperation for Space Standardization (ECSS) manages this set of standards and is used in all European space activities.

The ECSS was established in 1993 with the goal to create coherent standards in all European space activities. The ECSS standards cover four main branches:

- engineering
- management
- product assurance
- space sustainability.

The standards are built upon requirements that have to be met within the project. To achieve their objectives effectively, the standards concern themselves with the needs that have to be fulfilled to meet the requirements. They do not concern themselves with the means to be used to meet the requirements. This emphasis of what over how and function over form gives contractors a large degree of freedom. It makes standards easy to adopt as they do not infringe on companies infrastructure and usual practices.
The ECSS are updated continuously to uphold international standards.

The objectives of the ECSS standards are as follows:

- reducing risks
- interoperability and interface compatibility between components
- clear communication between involved parties
- quality and safety of space projects
- cost-effectiveness of space programs and projects
- competitiveness of European space programs

The ECSS standards are divided into four main branches: The Project Management branch (M), the Product assurance branch (Q), the Engineering branch (E), and

the Space Sustainability branch (U). Overlaying these four main branches is a fifth instructional branch, the System branch S. The structure of these branches is depicted in Figure 5.1.
The M branch mostly contains useful templates and contains 6 standards.
The Q branch is relevant to engineers and concerns materials and the processes to work with them. It currently consists of 62 branches, 10 handbooks, and 4 technical memoranda.
The E branch defines the end product and verifies that the customer's technical requirements have been met and are compatible with regulation and company constraints. It consists of 66 standards, 46 handbooks, and 6 technical memoranda.
The branches themselves are subdivided into disciplines. For example there are 7 disciplines in the engineering branches: E-10 to E-70.
The structure of the ECSS are shown in Figure 5.1.

The ECSS provides a large document database with 3 basic types of documents:

Standards (ST) are normative documents that are limited to verifiable requirements. These documents state clearly what to do.

Handbooks (HB) are non normative documents. They provide guidelines and state how to do it.

Technological memoranda (TM) are nonnormative documents that provide useful information and data for the space community. The content of these memoranda is not mature enough to become a standard or a handbook.

The denomination of the ECSS documents follows strict rules:
The Header of the ECSS document name is always ECSS. This denomination is followed by the letter of the branch the document concerns itself with: **S**ystem, **M**anagement, **Q**uality, **E**ngineering, and **U** Sustainability.
This letter is followed by one of three two-letter combinations, telling the reader what document type they hold in their hands: **ST**andard, **H**and**B**ook, and **T**echnological **M**emoranda.
The document type is followed by the number of the document. There are two types of numbers: type 1, which consists of a two-digit number and is used for generic requirements; and type 2, which consists of two two-digit numbers and is used for specific requirements.
The last part of the document name consists of the document version, which is denoted with a letter starting with A and ending at Z.
The formula for giving an ECSS document its name is depicted in equation 5.1.

$$ECSS - \begin{bmatrix} S \\ M \\ Q \\ E \\ U \end{bmatrix} - \begin{bmatrix} ST \\ HB \\ TM \end{bmatrix} - < number >< version > \tag{5.1}$$

An example of ECSS standard names are the following:

- ECSS-E-ST-50 Communications (standard)
- ECSS-HB-50-A Communications (handbook)
- ECSS-ST-50-05C Radio frequency and modulation (standard)

As this thesis is concerned with the testing of flight software, it concerns itself with the dependability branch/product assurance branch.

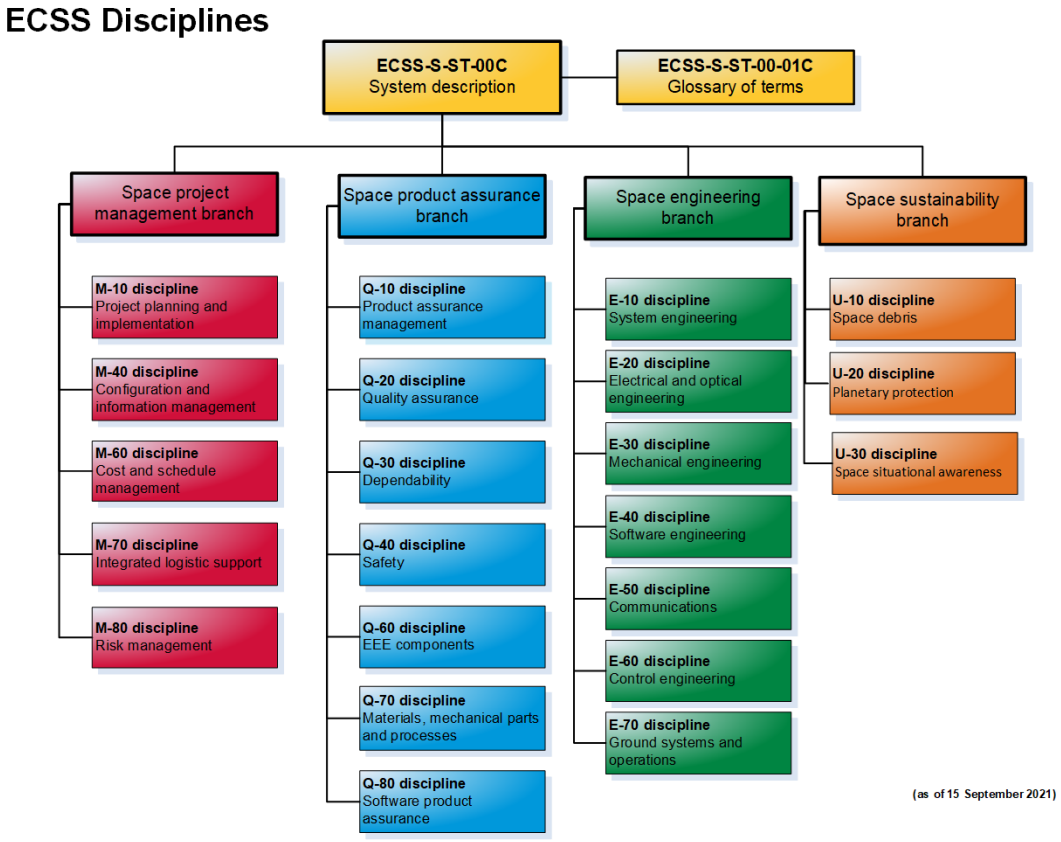

FIGURE 5.1: The main branches, subbranches and disciplines of the ECSS standard. (ESA, 2021)

5.1.1 Dependability

ECSS-Q-ST-30C is part of the space product assurance branch and concerns itself with the dependability of both hard- and software. The process of assuring dependability is continuous and iterative. It has to be repeated throughout the project life cycle.
The objectives of this standard are to identify all technical risks, ensure dependability targets are met, and to optimize cost and schedule.
The dependability standard identifies five levels of criticality for problems in hard- and software. These levels are depicted in Table 5.1. Depending upon the level of severity of the problem, different actions need to be taken.

Severity	Level	Dependability	Safety
Catastrophic	1	Failure propagation (Only for lower than system level analysis)	Loss of life, life-threatening or permanently disabling injury or occupational illness Loss of system Loss of an interfacing manned flight system Loss of launch site facilities Severe detrimental environmental effects
Critical	2	Loss of mission	Temporarily disabling but not life-threatening injury of temporary occupational illness Major damage to an interfacing flight system Major damage to ground facilities Major damage to public or private property Major detrimental environmental effects
Major	3	Major mission degradation	-
Minor or Negligible	4	Minor mission degradation or any other effect	-

TABLE 5.1: Severity classification. (ECSS, 2009)

A similar system exists specifically for the functions of software. This system is shown in Table 5.2.

Since a finished software product is only as good as its worst function, the finished software's rating is proportional to the severity rating of its most critical function. During the assignment of severity levels, the function of the software product within the overall system design is considered. For example, the existence of compensating provisions, preventing software-caused system failures or mitigating their consequences, can ease the severity rating. The levels of software criticality are shown in 5.3.

Severity	Function criticality	Criteria to assign criticality categories to functions
Catastrophic (Level 1)	I	A function that if not or incorrectly performed, or whose anomalous behavior can cause one or more feared events, resulting in catastrophic consequences
Critical (Level 2)	II	A function that if not or incorrectly performed, or whose anomalous behavior can cause one or more feared events, resulting in critical consequences
Major (Level 3)	III	A function that if not or incorrectly performed, or whose anomalous behavior can cause one or more feared events, resulting in major consequences
Minor or Negligible (Level 4)	IV	A function that if not or incorrectly performed, or whose anomalous behavior can cause one or more feared events, resulting in minor or negligible consequences

TABLE 5.2: Function severity classification. (ECSS, 2017)

Function criticality	Criticality category to be assigned to a software product
I	**Criticality category A if the software product is the sole means to implement the function** Criticality category B if, in addition, at least one of the following compensating provisions is available, meeting the requirements defined in clause 5.4.2: • A hardware implementation • A software implementation; this software implementation shall be classified as criticality A • An operational procedure
II	**Criticality category B if the software product is the sole means to implement the function** Criticality category C if, in addition, at least one of the following compensating provisions is available, meeting the requirements defined in clause 5.4.2: • A hardware implementation • A software implementation; this software implementation shall be classified as criticality B • An operational procedure
III	**Criticality category C if the software product is the sole means to implement the function** Criticality category D if, in addition, at least one of the following compensating provisions is available, meeting the requirements defined in clause 5.4.2: • A hardware implementation • A software implementation; this software implementation shall be classified as criticality C • An operational procedure
IV	**Criticality category D**

TABLE 5.3: Software severity classification. (ECSS, 2017)

5.2 The Packet Utilization Standard

Communication between Ground Control and the spacecraft is regulated by the Packet Utilization Standard (PUS) (ECSS, 2016b). According to this standard, ground control and satellite communication is always between two computers. One in ground control, the other on the satellite. The computer on the satellite runs various processes which can provide a multitude of services. Thus, the computer on the satellite is the service provider, while the computer in ground control is the service user. The exchange of information between service user and service provider occurs via packets, which are sent back and forth. There are two kinds of packets: requests, which carry commands to the service provider, which he then executes. These packets are also called commands. The second kind is called a report or a telemetry, and carries information from service provider to service user. A schema showing the architecture of the communication process between service user and service provider is depicted in Figure 5.2.
There are three different kinds of report (ESA, 2022):

- verification report
- event report
- data report

5.3 The Form of packets

TC and TM packets follow the form defined by the CCSDS Space Packet Protocol (CCSDS, 2023). The basic structure of these packets is shown in Figure 5.3. In general, the packets consist of two parts: the packet primary header and the packet data field. The primary header is filled with information to identify the packet. The packet type distinguishes TC and a TM packets: For a telemetry it is set to 0, for telecommand it is set to 1.

The distinguishing marks between different types of packets are the Service type (ST) and the Service sub-type (SST). The varying numbers tell about the purpose of the packet. The different purposes of packets are shown in Table 5.4 The actual data transported by a packet is found in the user data field after the primary and secondary headers.

5.3.1 TM packets

These are the packets sent by the service provider. While both TC and TM packets share the same basic structure, differences become apparent when the secondary header of both is examined. The structure of the secondary header is shown in Figure **??**. An important distinction between TM and TC packets is the existence of timestamps. Every TM packet is stamped with a time-signature of the spacecraft time. The service and subservice type ID are used to clearly identify the different types of TM packets as shown in Table 5.4.

5.3.2 TC packets

The packets sent by the service user do not have a timestamp but are still distinguished by ST and SST. Their structure is shown in Figure 5.5

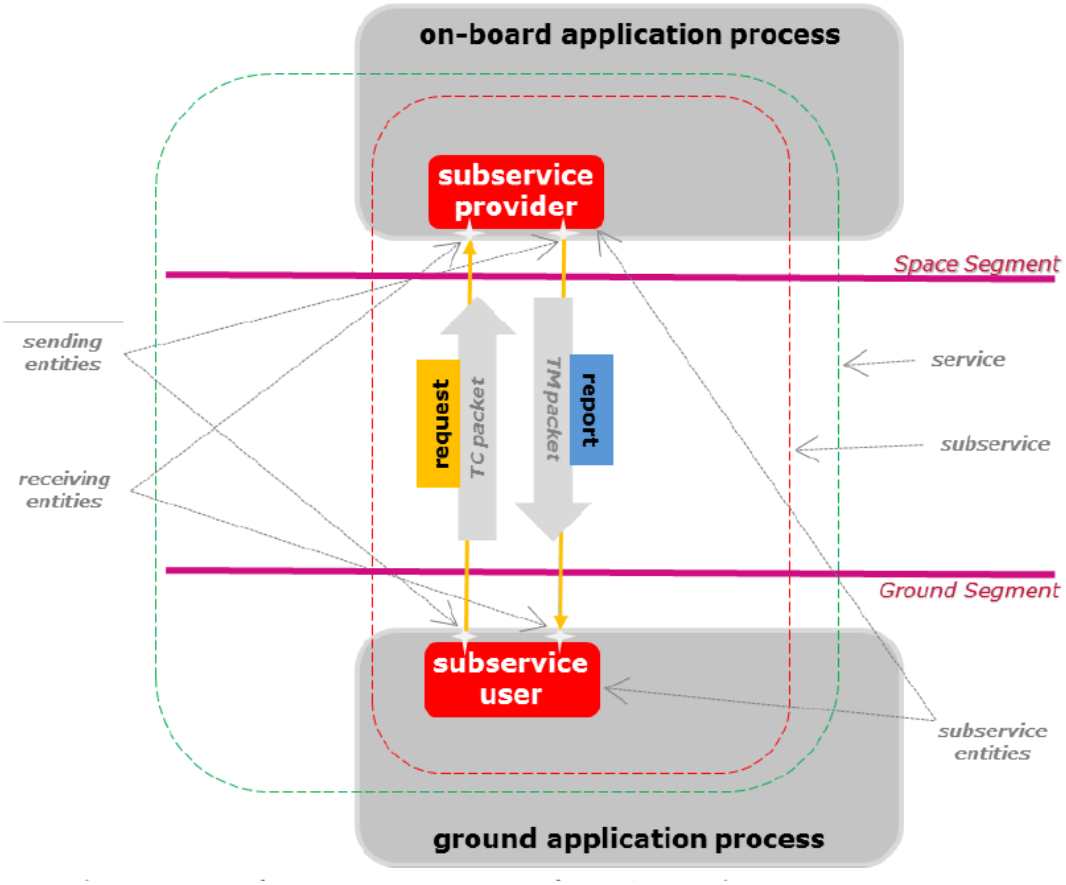

FIGURE 5.2: This figure shows the communication between service provider (the satellite) and service user (ground control). The communication happens via packets, designed in accordance with the PUS. Requests are executable commands that are transmitted via telecommand (TC) packets. Reports are packets filled with information for the service user. They are transmitted via telemetry (TM) packets. (ECSS, 2016b)

packet primary header							packet data field	
packet version number	packet ID			packet sequence control		packet data length	packet secondary header	user data field
	packet type	secondary header flag	application process ID	sequence flags	packet sequence count or packet name			
3 bits	1 bit	1 bit	11 bits	2 bits	14 bits	16 bits	variable	variable
2 octets				2 octets		2 octets	1 to 65536 octets	

FIGURE 5.3: The basic structure of TC and TM packets. (ECSS, 2016c)

Service Type Name	Service Type ID
Request verification	1
Device access	2
Housekeeping	3
Parameter statistics reporting	4
Event reporting	5
Memory management	6
(reserved)	7
Function management	8
Time management	9
(reserved)	10
Time-based scheduling	11
On-board storage and retrieval	12
Large packet transfer	13
Real-time forwarding control	14
On-board storage and retrieval	15
(reserved)	16
Test	17
On-board operations procedure	18
Event-action	19
On-board parameter management	20
Request sequencing	21
Position-based scheduling	22
File management	23

TABLE 5.4: The different Service Types of packets and their meanings (ECSS, 2016a).

5.3.3 Response to TC packets

Once a TC is received, the Service provider sends a number of response packets. These packets are all of Service Type 1. Depending on the outcome of the process, the response packet will have different Service Sub Types. The possible responses range from the TC packet being accepted to failing to execute the command in the packet. A detailed view of these responses is given in Subsection 4.1.2.

5.3.4 Service Types

The interaction between space, and ground capabilities takes place in the form of services. Every service type has a defined concept and related architecture, establishing the role it plays in the communication process.
In general two service-types exist:

- the standardized service type
- the mission-specific extensions

The mission-specific extensions can take two forms: they can be add-ons to the standard service types or new, mission-specific service types.
A mission does not need to use all standardized service types. Whenever a new mission starts, the standardized service types are filtered and the ones deemed necessary are implemented. Additionally, mission-specific services are created and additions to existing services can be made. The PUS standard service types and the

TM packet PUS version number	spacecraft time reference status	message type ID		message type counter	destination ID	time	spare
		service type ID	message subtype ID				
enumerated (4 bits)	enumerated (4 bits)	enumerated (8 bits)	enumerated (8 bits)	unsigned integer (16 bits)	enumerated (16 bits)	absolute time	fixed-size bit-string
							optional

FIGURE 5.4: The structure of the secondary header of TM packets. (ECSS, 2016a)

TC packet PUS version number	acknowledgement flags	message type ID		source ID	spare
		service type ID	message subtype ID		
enumerated (4 bits)	enumerated (4 bits)	enumerated (8 bits)	enumerated (8 bits)	enumerated (16 bits)	fixed-size bit-string
					optional

FIGURE 5.5: The structure of the secondary header of TM packets. (ECSS, 2016c)

mission-specific additions are then gathered in the mission-specific packet utilization definition document.
The choice, which standard service types and mission-specific additions make it into the mission-specific packet utilization definition document, depends on the mission objectives. (ECSS, 2016a)
The standardized service types are depicted in Table 4.1.

Sub-Service Types

The single subservices provide the means of communication between space and ground capabilities. Each TM or TC has to consist of a subservice type, characteristic of a specific service type. Each service type defines at least one subservice type. Subservice types without a service type do not exist.
Example: Service 6:
Service type 6 concerns memory management. Here memory refers to any physical or vertical memory area aboard the spacecraft. The memory service type consists of four subtypes, providing the capabilities to load, dump, and check the contents of these memories.
The four standardized subservices are (ECSS, 2016d):

- the raw data memory management subservice type;
- the structured data memory management subservice type;
- the common memory management subservice type;
- the memory configuration subservice type.

The raw data memory management subservice type allows the user to manage memories containing raw data (ECSS, 2016d).

The structured data memory management subservice type provides capabilities to manage memories that contain structured data (ECSS, 2016d).

The common memory management subservice type offers shared functions between raw and structured data memory management subservices. In this Standard, one such function is the "abort all memory dumps" capability, which works with both data memory management subservices to terminate all ongoing dump requests (ECSS, 2016d).

The memory configuration subservice type offers capabilities for managing memory as a whole, regardless of its content or specific addressing scheme. For instance, this subservice includes functions for enabling or disabling memory scrubbing and for applying write protection to memory (ECSS, 2016d).

Chapter 6

Framework for automated integration tests

6.1 The test process

The basic principle of testing the SXI DPU Software is as follows: First, a test script is generated. This script is then executed. The execution of the script sends a TC to the software, which responds in one of the ways described in Subsection 4.1.2. The responding TM is saved in the data pool, where it is looked up to check if the software's response was adequate.

To automate the process as much as possible, two Python scripts are generated per single test: the test script detailing the TC and the verification script looking up the corresponding TM in the Data Pool.

In addition to the Python scripts, the TST creates a LaTeX script to document the test process.

Tests usually do not consist of a single command. Often, a number of commands are sent to test a chain of processes in the software. In this case, a number of pre- and post-conditions might be applied. For example, the SXI's Science Mode can only start once the pre-science mode has been activated.

Two different kinds of tests are carried out. First there are the tests verifying that every process works a planned. Then there are the tests verifying that the software works even when something goes wrong. An example of this would be the command to open the radiation shutter before entering science mode. In this case, the software should send a TM that indicates a failed process. This indicates that the software works as intended, as this is something that should never happen.

6.2 Test Scripts

The Python test scripts are intended to be invoked by another script. Each test is a class that includes a method for every step, as well as a 'run()' method that executes all the steps in order. There are also functions for pre-conditions and post-conditions, which are designated areas for preparing the test setup, such as bringing the DPU up to a pre-determined state. The post-condition can be utilized to reset variables or return to a specific state.

A test class offers the following functionalities:

- Establishing the test's preconditions
- Executing individual steps by calling the step-function
- Running the complete test via the run function

- Providing feedback on the success of a function

When generating a test, the TST generates a JSON file. This file contains all relevant information about the test (name, description, pre-conditions, etc.). This file is the backbone of the test creation. The information for the other files created during the test process is obtained from this JSON file.

Return of test scripts

Every executed step causes a response in form of a TM. Depending on the TM details of the execution can be discerned. Originally the test-report contained only one Boolean value, indicating whether the step was successful or not. Later the outcome was saved into a dictionary, which contained more detailed information, according to the received TM. This made it possible to see if an exception had occurred and what type said exception had been.

6.2.1 Structure of a test script

A test comprises a class containing functions for each step, pre- and post-conditions, and an automated way of running the complete test. In addition to these functional features, each test contains a descriptive part for both the test script and the verification script. This description part is not executed while running the test and contains human-readable information about the test steps. It provides the user with information about the respective test step. When the class is instantiated, the __init__ (self) function sets information about the test, such as its name and description. The test designer is permitted to add variables and custom functions to the class, with test-specific variables having a consistent location within the __init__ (self) function. (Winkler, 2020)

6.2.2 Functions of a test script

__init__(self)

When a class is instantiated, the attributes of the class and their respective values are defined by a function that is invoked. In the case of test scripts, this implies that details about the test are established, and variables that need to be accessible across all methods are stored there. (Winkler, 2020)

establish_preconditions()

The function's purpose is to get the environment ready to conduct the test. The respective test setup is instructed to reach the intended states, if necessary. Housekeeping parameters are checked to verify if the preconditions have been satisfied. The function begins and concludes by obtaining and verifying the states of 'iaswState', 'FEEState', 'FEEOperState', and 'sdbState'. The test designer can effortlessly include additional parameters. If the parameters meet the expected criteria, the function is considered successful and returns True. (Winkler, 2020)

step_x

The function named step_x is generated for each step, and upon calling this function, information such as the step number, description, and comments are logged. This is

where the test designer adds the necessary commands or verifications. The success is logged at the end of the step, and a summary is returned. (Winkler, 2020)

postconditions

The postcondition function may revert services and parameters to their original values. The default-generated function resets housekeeping values (service 3) to their default frequency and enabled state. (Winkler, 2020)

run()

The run() method can be invoked to conduct the entire test case automatically. Its functions include:

- setting up the logger
- establishing the preconditions
- consecutively executing each step
- noting successful steps
- tracking any exceptions that occur
- checking integrity (optional)
- saving the data pool
- returning all details from the test run

6.3 Logging

A file is generated to comprehend what occurs when a test is run and store this information, aiding the tester in tracing the cause of a failed step. The primary objective of logging is to retain the necessary amount of information to quickly identify the cause of failure, striking a balance between too much and too little. The Python logging module is employed for this purpose, and testers can use the logging levels listed in Table 6.1 to alter the output verbosity.

The logging creates a unique file for each test in which the messages are saved. The logging level specifies which entries are written to the file. Additionally, a separate file is created for the manual-step scripts. It is also possible to enable a master log file that captures all messages. Furthermore, the data pool is saved for each test. An important aspect to consider is the module or code that generated the log entry. To address this, the module name is incorporated into the log message, with a simple statement ensuring that the module name dynamically changes.

```
log = logging.getLogger(__name__)
```

The logging appears as follows in the example snippet above, which displays how the 'establish_preconditions' method logs the Statemachines' state. (Winkler, 2020)

CRITICAL	Errors in the framework, preventing an ordinary execution of the test scripts.
WARNING	Important information indicates that a test does not run nominally. Such as bypassing the preconditions because the simulators are not able to reproduce the behavior of the hardware. Library functions where a not-expected behavior occurs.
INFO	All information that is logged while executing a test script nominal. Library functions which have the purpose to log information.
DEBUG	Information, which is not necessary to validate the test conditions. Primary for additional information, to examine more carefully.

TABLE 6.1: Logging levels and their purpose within the framework.

Chapter 7

Verification

A telemetry packet is sent back after a telecommand has been sent to the service provider. The package's content can be read if the telecommand has been executed successfully or if an error occurred. The goal of the TST is to give the tedious tasks of testing to the computer. Therefore, it falls to the TST to verify whether the service provider's output has the correct values. Nonetheless, human oversight is required to conclude if a test has been run successfully. In the window *Progress Viewer*, the conclusion of the testing process is displayed in a human-readable form.

The goal of the Testing process is to verify that the software works as it is supposed to. To this end, a series of telecommands is sent to the service provider. The telemetry packets sent back to the service user are then used to verify whether the software works as intended. The exchange process between service provider and service user is detailed in Section 5.2. The basic approach to verify the software is to send a TC and then check the response packages. Overall, nine response packages are listed in Table 4.3. The optimal outcome is to receive a combination of the two packages TM[1,1] and TM[1,7]. This means that the sent command has been executed without error and that the service provider finished the execution and moved on to other processes. If the response differs from TM[1,1] and TM[1,7], the received TMs are used to troubleshoot the problem, as the different numbers hint at the exact step of the process that went wrong.

7.1 The Data Pool

The data pool is the database in which all sent TCs and TMs are recorded. Whenever a new command or telemetry is sent, the database is updated. To keep track of all data sent in, the data pool creates different "sub-pools". Each of these "sub-pools" is identified by the variable pool_id. This variable increments by one for every connection to the data pool. This makes it easy to keep track of the data of one session.
The data pool keeps track of TCs and TMs on different counters.

7.1.1 The Pool Viewer

The information from the data pool is depicted in real-time in the pool viewer. The pool viewer during a test run is depicted in Figure 7.1.

The pool viewer depicts the following information about the incoming packets:

- #: The Packet number within the pool-viewer
- TM/TC: shows if the packet is a TM or a TC
- APID: APID of packet

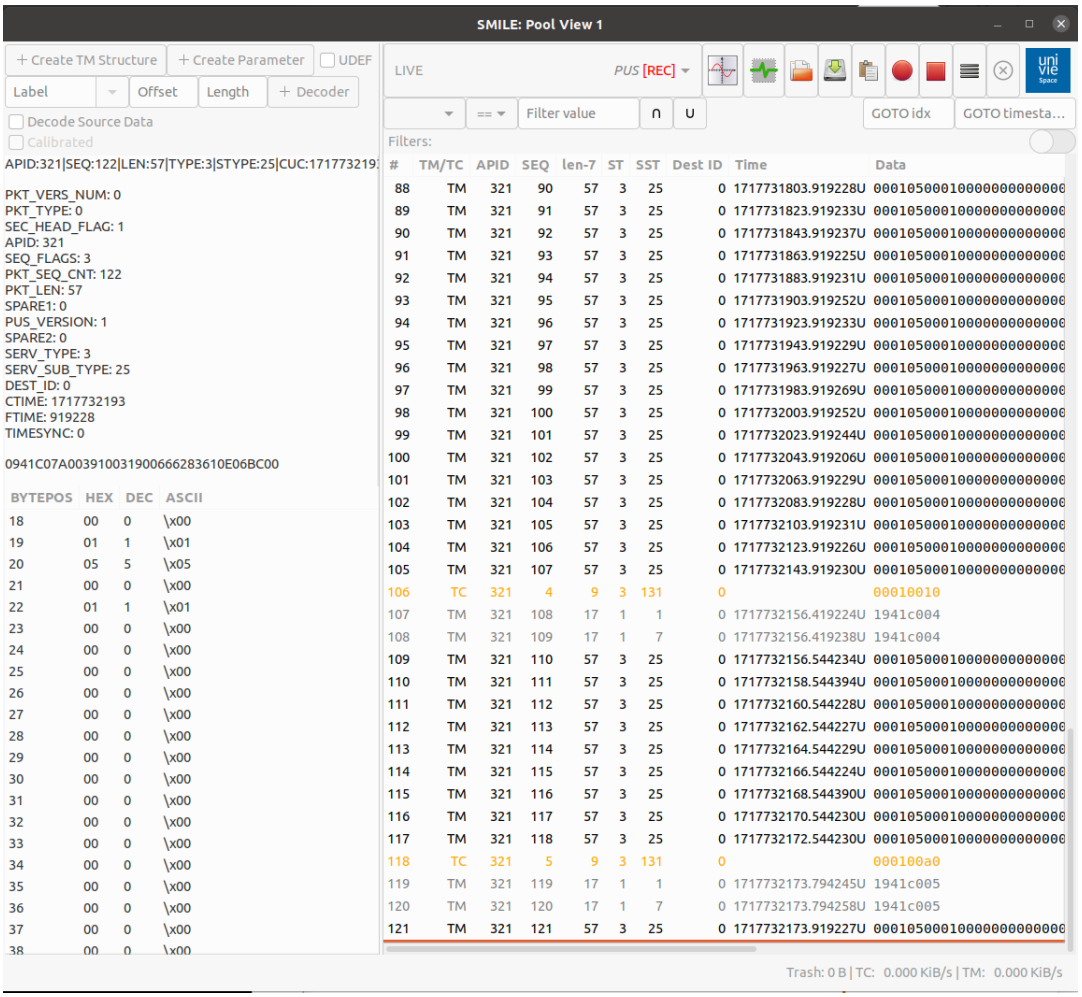

FIGURE 7.1: The running data pool. TMs and TCs are easily distinguished by their different colors. In this image, two TCs (yellow) are visible. The gray TMs following them are the response TMs. The black TMs is the Housekeeping. Before the first TC it arrives every 20 seconds. After the first TC it arrives every 2 seconds. # and SEQ differ because the pool-viewer was only started after a few TMs had already arrived in the data pool and because the data pool counts TMs and TCs on different lists, while the pool-viewer does not.

- SEQ: Packet number within the data pool
- len-7: length of packet
- ST: Service Type of packet
- SST: Subservice Type of packet
- Dest ID: Gives ID of subsystem, 0 stands for Ground-control
- Time: The time at which the simulator received the packet. Only exists for TM.
- Data: Complete Data of the packet

7.1.2 The Data Pool's role during Verification

The data pool plays a crucial role in the verification process. The basic procedure of Verification is to check if the software does what it is supposed to do. Do the cyclical activities take place as intended? Is every TC answered with a set of corresponding TMs?
To answer these questions, the Verification Code (as defined in the TST) searches the data pool and looks for corresponding packets.

7.2 The Progress Viewer

The Progress Viewer generates human-readable output of the testing process. To properly function, it needs three pre-existing files: the specification file of the respective test, the log file of the telecommand, and the log file of the verification.

7.2.1 The Files

The Test Specification log file

This file stores all the information about the test as specified in its single steps. All information given by the human user is found in this file. Among the variables stored in this file are:

- name
- description
- spec_version
- iasw_version
- primary_counter
- precon_name
- precon_code
- precon_descr
- postcon_name
- postcon_code
- postcon_descr
- comment

The log file of a simple test specification called *Change_HK_Period* is shown in Figure 7.2. The test itself does not do much and only consists of one step. The structure of the file is apparent. This file is generated whenever the save button is pressed. If no such file exists, a new one will be created. If a file already exists, a new one will be generated and saved, overwriting the old file.
Since the data saved into the log file is taken directly from the GTK Notebook in which the human user specifies the test and its steps, the Progress Viewer can only open a test and show its progress after the specification has been saved.
The test specification file is identified by its characteristic file name *<testname>-TS-.json*.

```
[
 "_name": "Change_HK_Period",
 "_description": "changes the period of HK",
 "_spec_version": "3",
 "_iasw_version": "0.2R",
 "_primary_counter_locked": false,
 "_precon_name": "No Precondition",
 "_precon_code": "success = True",
 "_precon_descr": "No Precondition",
 "_postcon_name": "No Postcondition",
 "_postcon_code": "success = True",
 "_postcon_descr": "No Postcondition\t",
 "_comment": "",
 "sequences": [
  {
   "_sequence": 0,
   "_name": "",
   "_description": "",
   "_spec_version": "",
   "_primary_counter_locked": false,
   "steps": [
    {
     "_primary_counter": 1,
     "_secondary_counter": 0,
     "_step_number": "1.0",
     "_description": "Change the the frequency of HK to 2 seconds",
     "_command_code": "SldNoCal = 1  # KSP50190\nPeriod = 16  # KSP50173\ncfl.Tcsend_DB('SASW ModHkPeriodCmd', SldNoCal, Period, pool_name='LIVE')",
     "_step_comment": "Change HK Period to 2 seconds",
     "_verification_code": "result = tm.get_frequency_of_hk(duration=60.0, frequency=2, SID=1, pool_name='LIVE')",
     "_verification_description": "Check if TMs arrive every 2 seconds",
     "_is_active": true,
     "_verified_item": [],
     "_start_sequence": null,
     "_stop_sequence": null
    }
   ]
  }
 ]
]
```


FIGURE 7.2: The log file of the test specification of a simple test called *Change_HK_Period*. All important data of the test specification is saved here. The test itself is very short, consisting of only one step.

The Command log file

The second external file needed by the Progress Viewer is the command.log file. Whenever a test with the same name is run, its results are appended to the file. This file shows the exact date and time of the testing: when the preconditions are started and finished and when the postconditions are finished and started.
This file keeps track of all the information that is gathered during the execution of the test scripts. These data comprise most of the points listed in Chapter 7.2.1. Additionally date and time of the command execution and date and time of the finished execution of every single test step is stored.

The Verification log file

In its function this third file is similar to the command log file. It contains all information about the verification code: which test step it belongs to, the time at which the verification code has been executed, which spec version is used, the verification description and the description of the entire step. After this initial information follows an entry about the finished execution of the verification code. This entry has information about the step, the time at which the verification code finished executing and the result of said execution. The result can have two outcomes: *true* or *false*.

7.2.2 Presentation in the Progress Viewer

The Progress Viewer is divided into two halves. In the upper half, the three files from which the Viewer takes its information are specified.
The bottom half shows said information. Test name and test description are taken from the test specification log file.
The step number and description are taken from the command.log file. Also taken from this log file are the timestamp, the Spec Version, the Status, and the Result.
The verification.log file is the source of the verification timestamp, the Spec. Version, the verification Status, and the Result.

Since the result is of special interest, it is color-coded. If the command was executed successfully the cell with the result is colored green. Similarly, if the verification passes with True, it is coded green. If either command or verification results in a False, the respective cell is colored red.
This makes the Progress Viewer easy to read. An example of both successful and unsuccessful tests is given in Figure 7.3.
This figure shows the progress view for three runs of the test *Neuer Test*. The first two runs were executed without error. The third run produced an error in the verification. The reason for this error is not apparent and has to be searched for.

7.3 The Library

7.3.1 tm.py

The telemetry library module comprises functions primarily used for developing verification code. The primary objective is to retrieve telemetry packets from the database, and various functions are available to accomplish this task. All functions that retrieve TM packets should verify whether it is a TM or a TC, with this information stored in the packet header. The framework should be functional both 'live' and 'offline.' When used 'live,' the functions retrieving packets may have to wait until the packet is sent, and in certain situations, establishing limits for these requests may be challenging. There are three distinct groups of functions:

- Functions that do not wait: get TM with specific parameters: st, sst, apid, event_ id, procedure_id within a time interval in the [past, now*]
- Function that waits for a specific TM packet (TM may be received in the future): get TM with specific parameters: st, sst, apid, event_id, procedure_id time intervals: [past, future], [now*, future]. There must always be a maximum waiting time to prevent the program from freezing.
- Functions that collect TMs over a duration: get TM with specific parameters: st, sst, apid, event_id, procedure_id time intervals: [past, future], [now*, future]. *: now refers to the CUC timestamp of the last TM packet in the pool. The packet's kind does not matter.

Functions for Housekeeping include:

- Get the last of a kind.
- Get the next of a kind (await housekeeping).
- Get all of a kind for a time interval.
- Get hk-entries of the last of a kind.
- Get hk-entries of the next of a kind (await housekeeping).
- Get hk-entries of all of a kind for a time interval.

Functions of the Events include:

- Get the last event of a kind.
- Get the next event of a kind (await event).

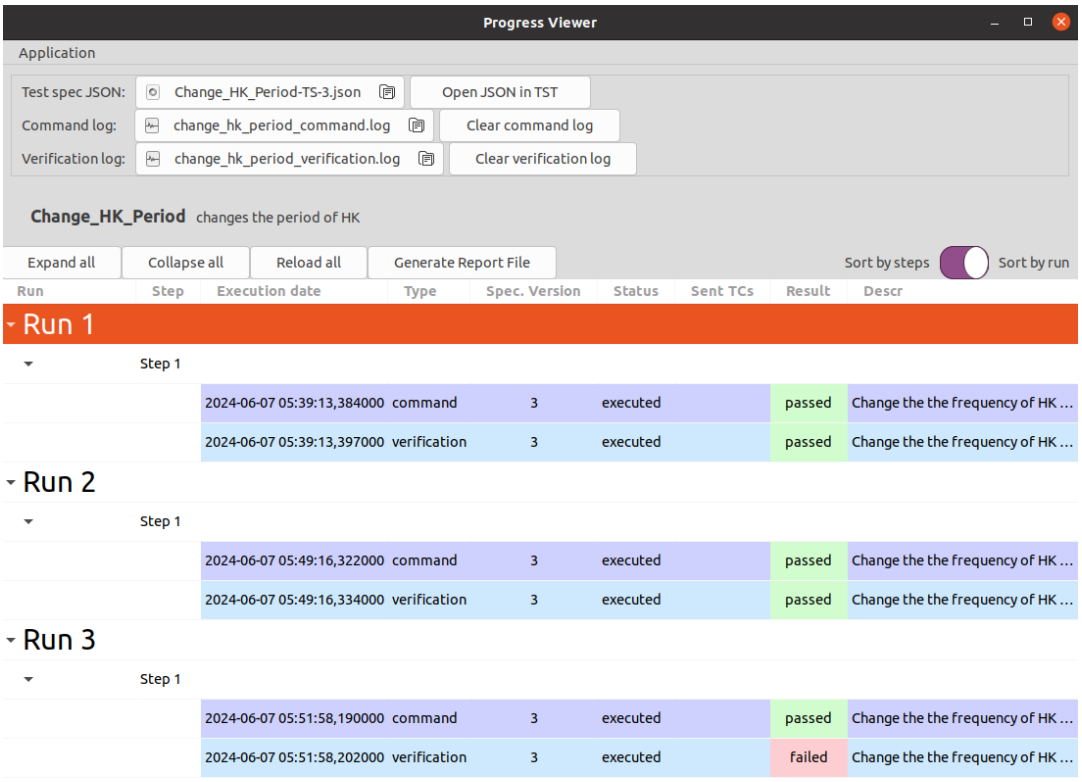

FIGURE 7.3: The Progress Viewer after the same one-step test has been run three times. In the upper part of the Viewer, the log files are specified. In the lower part, the three different runs are displayed. The steps' dark blue part is the command log information. The light blue part is the information from the verification log. The rightmost part shows if either command or verification have been successful. In all three cases the command was executed without error. But during the third run, the verification failed.

- Get all events of a kind for a time interval.
- Get event data entry.

Functions for acknowledgments include:

- Get the acknowledgment TM packets for a specific TC or TCs.
- If the TCs were sent a few seconds ago, the acknowledgment TMs may not be created yet because actions of IASW may take some time.

7.4 A simple test

The Figures 7.1, 7.2, and 7.3 show the workings of a single, simple test. The purpose of this test was to change the housekeeping frequency from 20 seconds (Period of 16 cycles), to 2 seconds (Period of 16 cycles).
The test with all its information is depicted in 7.2. The results are seen in both 7.1, and 7.3.

In the pool viewer (7.1), two commands (yellow) enter the data pool. The following two TMs are grayed out, marking them as direct responses to the yellow TCs.
A look at the timestamp (second to last column) shows that the regular TMs (black) came in every 20 seconds. After the first TC, they arrived every 2 seconds.
While not observable in the figure, the second TC set the HK frequency back to 20 seconds.

The Progress Viewer also shows the results (7.3). However, the data gained from it differs from that of the pool viewer.
First off, the data viewer does not show the frequency of the HK. It does not need to. When the verification was run, it checked, if the HK had the desired frequency. If so, the test was marked as successful. If not, it failed.
Therefore, it is not necessary for the Progress Viewer to show the exact ins and outs of a test. Only if the command was executed successfully and if the verification passed successfully.
While the pool viewer only shows a small part of the pool, the progress viewer structures the data associated with a test and shows it for each run.
The first TC in the pool viewer corresponds with the first run in the progress viewer. The resetting of the HK frequency is not visible in the progress viewer. This is because the test to reset to a frequency of 20 has a different name.
After the first successful test, another change in the HK report structure was made, which was also successful. The third run of the test failed.

Chapter 8

Automated Testing

8.1 The testing process

To complete the ECSS-prescribed testing process, a multitude of scripts need to be executed. These scripts are not detailed by the ECSS standards. Instead, they only specify what the test scripts need to accomplish. Each test script consists of at least one TC. The satellite software then needs to respond in the way prescribed by the standards. As soon as the test script is executed and the TC is sent, the software will first react in the number of ways detailed in Subsection 4.1.2.

All outcomes to commands are clearly defined both in cases of success and failure. To assure that the software works as intended it is therefore necessary to also create errors and check if the response is as expected. Tests that check these errors usually give commands that are impossible to execute in the respective state of the software. An example of this would be to start science mode without first preparing science mode. If this does not result in an error, the software must be revised.

The majority of test scripts, however, are designed to create executable commands. After an executable command is received, the IASW reacts to it with a variety of TMs which indicate if the single steps of the command have been successful or unsuccessful. In general, when a TM shows an odd-numbered sub-service type, the step has been cleared successfully. If the SST is even, the respective step has not been cleared successfully. Therefore TMs can be seen as diametrically opposed pairs: TM[1,1] and TM[1,2] indicate if the verification report was accepted successfully or not. TM[1,3] and TM[1,4] report if the verification report started its execution successfully or not. TM[1,5] and TM[1,6] show the success of the progress of the verification report. TM[1,7] and TM[1,8] indicate the completion of the execution of the verification report (ECSS, 2016a). (See also 4.3)

After successful execution, the consequences of the command are transmitted to the data pool. These consequences can take multiple forms, from changing the frequency of housekeeping to entering science mode. Each of these consequences is broadcast to the data pool.
In many cases to verify that the TCs were executed successfully, the (1,3), the (1,7) TM are sufficient. Additionally the consequences of execution need to be exactly as prescribed.
Therefore, another script needs to be created, the verification script. The function of this script is to check the data pool for the correct responses to the sent TC and give clearance once the correct output has been identified.
After the successful execution of both scripts, the executed TC is marked as working, and the testing continues to the next step.

8.2 The Test Specification Tool

In past projects, the test and verification scripts were written manually. This resulted in an approximate 1800 hours of work expended to thoroughly test each software function as described in the ECSS standards.
To reduce the amount of human labor involved in testing, the Test Specification Tool (TST) was created. Stefan Winkler created the first version of this tool (see Winkler, 2020) to help in the testing of the CHEOPS (CHaracterising ExOPlanets Satellite) software. In the course of this thesis the TST was updated and developed further.

8.3 The TST during CHEOPS

A lot of the footwork of the TST was done during CHEOPS. The structure of the test scripts was defined, script skeleton generation was implemented, and a logging tool was created. An extensive list can be found in Chapter 6 and a more detailed description in Winkler, 2020.

8.4 The TST during SMILE

The ultimate goal was to make the TST user-friendly while not sacrificing testing accuracy. A handful of useful features were identified to ease the testing process:
Entire tests, with all their steps, needed to be saved. This would greatly help if the software needs to be tested again after small changes were made. Test steps that are needed multiple times during tests should be able to be saved and used again with ease. The commands executed during the single steps should be readily available. Their implementation should consist of as few strokes of the keyboard as possible. For purposes of oversight, the information about telemetries should be easy to find and compare to the output of the tests.

All of these features were realized in the right-hand side of the TST. This part of the window serves as a toolbox and a reference table for the tester. Via connections to the database and log files, every information needed for testing is generated in tables, from which the information can be transferred to the actual test steps on the left-hand side.

The TST is depicted in Figure 8.2

8.5 The databases

A Management Information Base (MIB) is used to monitor the functioning of the satellite. It is used to store and retrieve data about the operational state of the devices in the network. The MIB therefore serves as a database for network devices. The SMIL MIB contains a collection of objects (like counters and configuration parameters) organized hierarchically. Each object represents a specific piece of data such as packet header, command sequences, and parameter sets (SCOS-2000-Team, 2010a) The MIB structure is similar to tables or schemas in a relational database, but is designed specifically for network device information rather than general-purpose data storage (Wittman, 2024).
To retrieve the necessary information each object in a MIB is identified by an Object Identifier (OID), which acts like a key or address to the specific data point. OIDs are used to request specific data from the MIB. (Wittman, 2024)

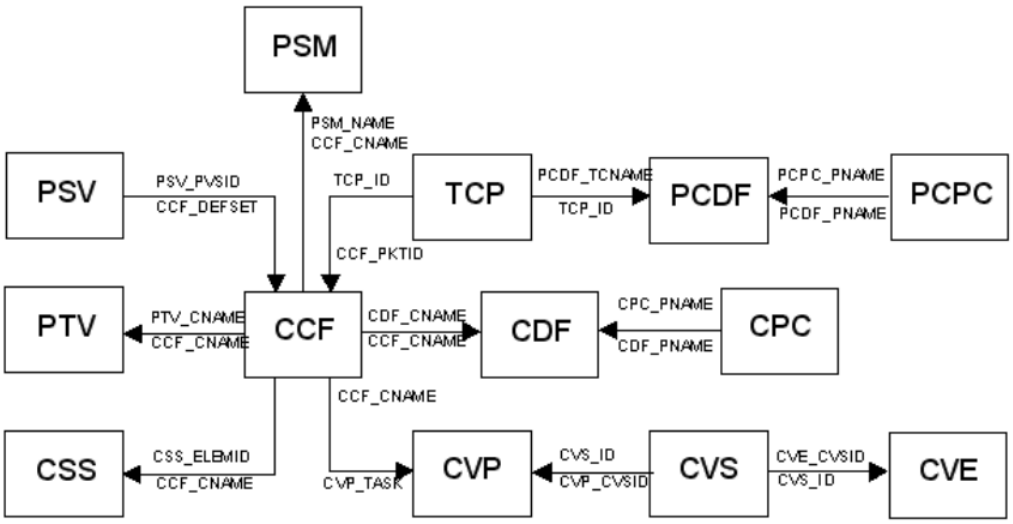

FIGURE 8.1: The relationships for the Command tables of the MIB. (SCOS-2000-Team, 2010b)

The data used for the TST comes from two separate SQL databases: the SMILE MIB and the SMILE DATA STORAGE. The SMILE MIB is used to generate the information depicted in the TST tables *TC Table*, *TM Table*, and *Data Pool*. Via various joins, the information in the SMILE MIB is used to generate the content of said tabs. The connections between some of the MIB tables are depicted in fig. 8.1. The TST is not permanently connected to the SMILE MIB. It only connects during the startup phase to generate the tables.

There are two ways for the running database to be changed: After the database is updated, the program is restarted, or the new database scheme is re-loaded while the TST is still running. The latter works via the “Select IDB” button.
For changes in the database currently running, the program needs to be restarted. Otherwise, to take effect, the program needs to be restarted.

The SMILE DATA STORAGE is the database where the poolviewer’s information is stored. All incoming and outgoing packages are stored here. The connection to this database ensures that the TST can access the communication data in real-time. The TST is permanently connected to this database. This allows the verification steps to access the telemetered data, among other things.

8.5.1 TC Table

This tab gives all necessary information to create a test command. To achieve this, the basic form of the table is a list of all commands as given in the SCOS-2000 database for SMILE. Every command and every piece of information depicted in this table is directly imported from this database. Therefore, new commands can be easily fit into the "TC Table" tab by adding new commands into the SCOS-2000 compliant database, in accordance with the ECSS-Standards.

The main command list, lists type and subtype of the command and shows both a short description and a long description of said command. Selecting one command gives further information about parameter and datatype.
To ease the testing process, additional information about every command is given.

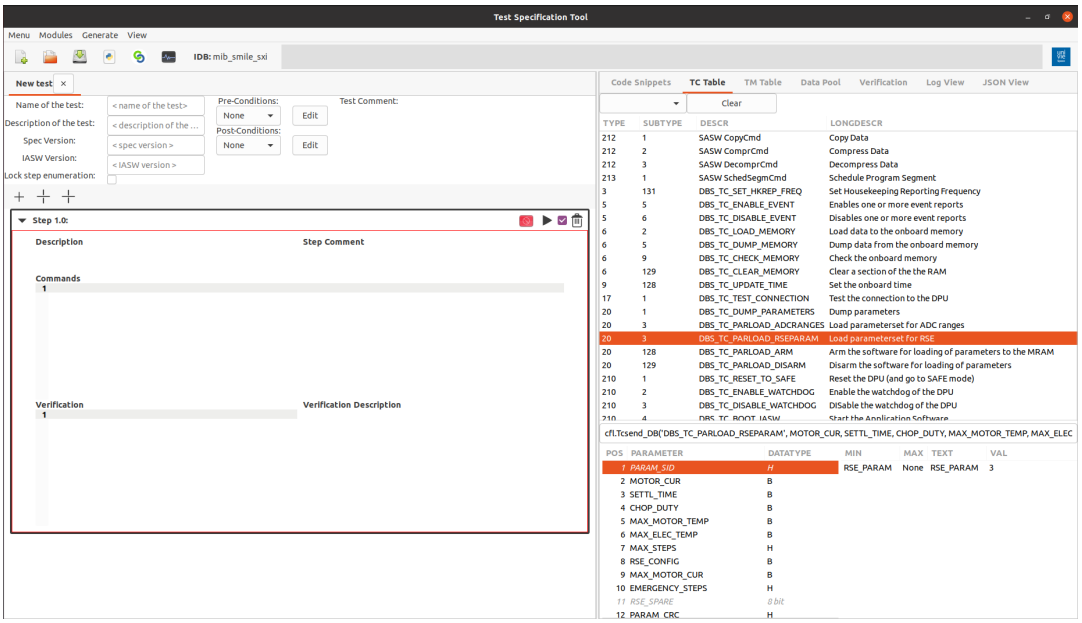

FIGURE 8.2: The TST, ready to create the first step of a new test. On the left-hand side, the structure of the test is built. Steps can be added and deleted with one click. The tests can be named and pre- and postconditions can be added. The right-hand side contains tabs that help create the test steps. The currently selected tab is the TC table, which lists all telecommands available.

This information consists of all the variables necessary to generate the command. It is specific to each TC and serves as a helpful reference when creating the commands.

The following pieces of information are given in the first treeview:

1. Type
2. Subtype
3. Description
4. Long Description

The following columns are given in the second treeview (command treeview):

1. Position: position in the treeview
2. Parameter: textual description of command parameter; position in database: CPC_DESCR (SCOS-2000-Team, 2010b)
3. Datatype: Parameter Format Code, identifies parameter length, 3 possible types: **I**nteger, **B**inary, **H**exadecimal; position in database: CPC_PFC (SCOS-2000-Team, 2010b)

The following columns are given in the third treeview (calibration treeview):

1. prv minval: shows the low limit value (or the allowed value if prv maxval is Null); position in database: PRV_MINVAL (SCOS-2000-Team, 2010b)
2. prv maxval: shows the high limit value; position in database: CPC_MAXVAL (SCOS-2000-Team, 2010b)

3. pas altxt: shows the text string used to provide an engineering interpretation to the raw value specified in pas alval; position in database: PAS_ALTXT (SCOS-2000-Team, 2010b)

4. pas alval: shows the raw value expressed (in decimal) in a format compatible with the corresponding PAF definition; position in database: PAS_ALVAL (SCOS-2000-Team, 2010b)

8.5.2 Data Pool

This table depicts the packet ID (PID) with the packet name and datatype. It also serves as a reference for the user. The information in the table can be inserted into other applications via drag-and-drop or right-click.

8.5.3 Verification Table

Similar to the Telecommand table this userface-element contains the commands necessary for the automated Verification. Via Drag and Drop, the commands are dropped into the *Verification Code* text field. When the test is executed, the Verification code is executed right after the telecommand. Depending on the process necessary to verify the working of the TC, different functions can be chosen to be executed. These functions vary in their **performance/realization**. There are functions to check if the TMs (1,1) and (1,7) are received and if a certain output is generated within a certain amount of time, ...
This modular aspect makes creating the required Verification for every TC possible.

Chapter 9

Conclusion

In this project, we successfully developed an automated testing software tailored for the flight software of satellites. The primary goal was to create a robust system capable of streamlining the execution and verification processes, ensuring that the software meets the ECSS standards. Throughout the development cycle, the automation of test cases progressed, allowing for continuous integration testing. The automated nature of the solution significantly reduces the time and effort needed for manual testing, improving overall efficiency.

The project's most notable outcome is the software's immediate readiness for practical use in the ARIEL, ATHENA, and ARRAKIHS projects. Outside the University of Vienna, the TST is used to test the software of the Planet Interceptor's CoCa and MANIaC instruments.

These successes notwithstanding, it is important to recognize that while the current implementation meets the expected requirements, future developments in satellite technology or mission complexity may introduce new challenges. Thus, regular reviews and updates should be considered to ensure continued alignment with evolving mission standards and requirements.

In conclusion, this project provides a robust, efficient, and scalable solution for automated flight software testing. It lays a solid foundation for future advancements and contributes to the overall reliability of satellite missions.

Appendix A

User Manual

A.1 Introduction

The TST is a tool for easily creating tests for missions such as CHEOPS and SMILE. The testing process is very simple: A command is sent to a running program, and the program responds to this command.

The command sent by the user is called a telecommand (TC), and the response of the program is called the telemetry (TM). Testing, therefore, is a process where TCs are sent and TMs are received.

Each TC has an expected TM as response. During testing, we want to check if the response to each TC is as expected or if errors occur. This is the job of the TST.

A.2 Setup of the TST

To set up the TST, follow the steps in the README file.

A.3 Connect the MIB

To setup the MIB enter the CCS directory. Enter the following path: CCS/Ccs/tools. In the tools directory execute the import_mib.py file: ./import_mib.py
This will show how to connect the MIB:
$./import\_mib.py < MIBDIR >< DBSCHEMA >< DBUSERNAME >$
For MBDIR enter the directory of the MIB, for DBSCHEMA enter the Schema you want to use and for DBUSERNAME enter the user of the database.

Once the MIB is connected it will be displayed in the top of the TST window next to the icons (see Figure A.1)

A.4 Create a single Test Step

The first step to create a test, is to click on the icon showing a white page and a green plus. This icon is located in the top left corner of the window, right below the Menu button. Hovering above the icon shows the text *New test specification*. Every time this button is clicked, a new test tab will open. The new test tab is shown in Figure A.1. It takes in the left part of the window.
In this new tab, the test can now be named, a short description can be written, and pre- and post-conditions can be filled in.
Below the area for the whole test, the single test steps can be defined. At the beginning, there will only be a single window for one undefined test step, called Step 1.0. Here, the test step can be described, and comments can be added.

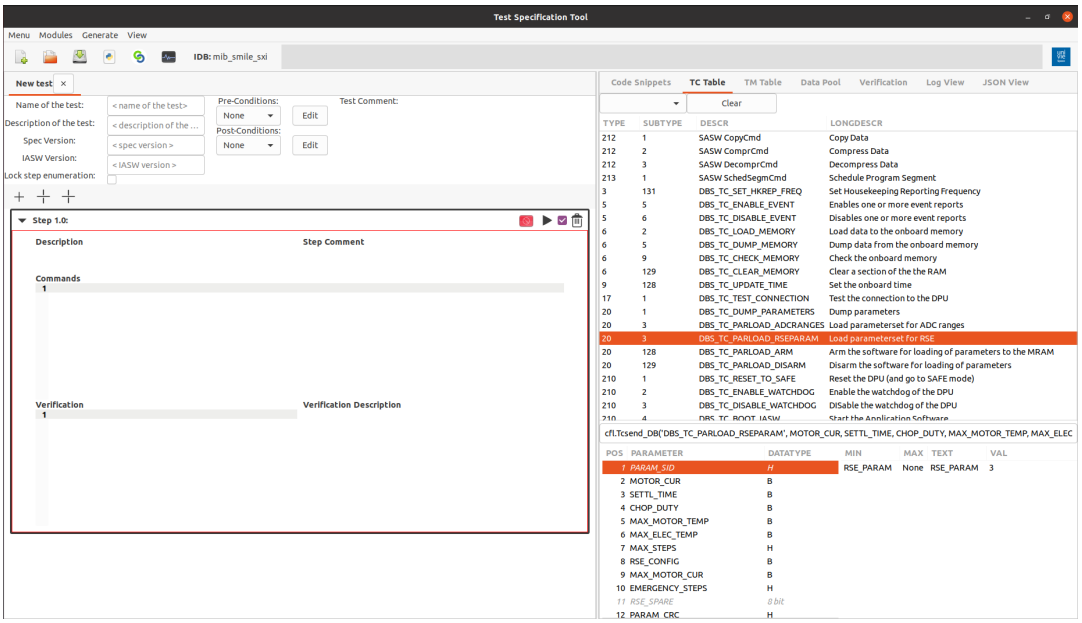

FIGURE A.1: The blank TST window

The two most important fields are the *Commands* field and the *Verification Code* field. Here the actual code has to be inserted. The *Commands* field takes in the code of the TC while the *Verification Code* takes in the code of the verification (see Section A.6.5) The fields with *Step Comment* and *Verification Description* are optional as the test will only need either the *Commands* field or the *Verification Code* field.
Pool Name: The Pool Name gives the connection to the poolviewer. A wrong name will lead to a faulty connection.

To simplify the process of filling the *Commands* or *Verification Code* fields, tools have been created. These tools are found on the right-hand side of the window and are organized in tabs.
The names of the tabs are the following: *Code Snippets*, *TC Table*, *TM Table*, *Data Pool*, *Verification*, *Log View*, *JSON View*.

To fill in the *Commands* field, navigate to the tab *Tc Table* (shown in A.1 on the right-hand side). Here all the TCs are listed. Selecting one of the TCs displays further information in the windows below. To insert the code of the Telecommand simply drag and drop the TC into the *Commands* field. The code and all its parameters will show and can be edited as needed.
If a PID is needed navigate to the tab *Data Pool*. Here all PIDs are listed with Name and Datatype. Once the correct PID is found it can also be dragged into the *Commands* field.

To create the Verification, select the *Verification* tab and similarly drag the needed verification into the *Verification Tab* field.

Side Note: The Verification Tab is currently under construction and might not be available on all releases.

After the fields have been filled in, the play button in the top right corner can be pressed, and the test step will execute. This needs to be done for each test step individually.

To add further test steps to the test, press the + sign on the left above the Test Step.

If test steps have been forgotten, just right-click on the name of the test step to insert a new test step above or below the selected one.

A.4.1 Defining time in a test step

The software does not measure time in seconds. Instead, it runs in cycles at a speed of 8 Hz (see Section 4.2). So one cycle has a runtime of 0.125 seconds. Therefore 1 second is defined as 8 cycles. **Example:** When modifying the frequency of Housekeeping (3,131) the period of the new HK needs to be defined via: *Period = 8*. This would result to a received HK report every second. To reset it to 20 seconds, the following is necessary: *Period = 160*.

A.5 Run a Test

To create a test consisting of multiple test steps repeat the process of Section A.4 multiple times.
This test can be executed either by pressing the play button for each step separately or, easier, by selecting the *Generate* option in the menu bar. Selecting the option *Generate Scripts* will open the CCS Editor with 4 tabs open. Each of the tabs contains a script, created from the test parameters.

The *prep_test_env.py* script will prepare the environment for running the test automatically. This script should be run first.
The *command.py* script contains all the steps and can be used to execute all steps.
The *run.py* script provides a way to execute every single step of the test.
The *verification.py* script contains the scripts of every step's verification.

A.6 The right-hand side Tabs

These tabs contain useful tools.

A.6.1 Code Snippets

Single code snippets can be reused here. If a single test step is used more than once, the step can be dragged into the *Code Snippets* tab. There it will be saved and displayed with its description. When the step needs to be used again, it can be dragged above or below an already existing step to be inserted there.

A.6.2 TC Table

This table shows all the necessary information about TCs. The information contained within the entries can be dragged and dropped into text fields.

A.6.3 TM Table

Like the TC Table, this tab shows all necessary information about TMs. The description of the TMs can be dragged and dropped into text fields.

A.6.4 Data Pool

A list of all PIDs with name and datatype. The PID can be dragged and dropped into text fields.

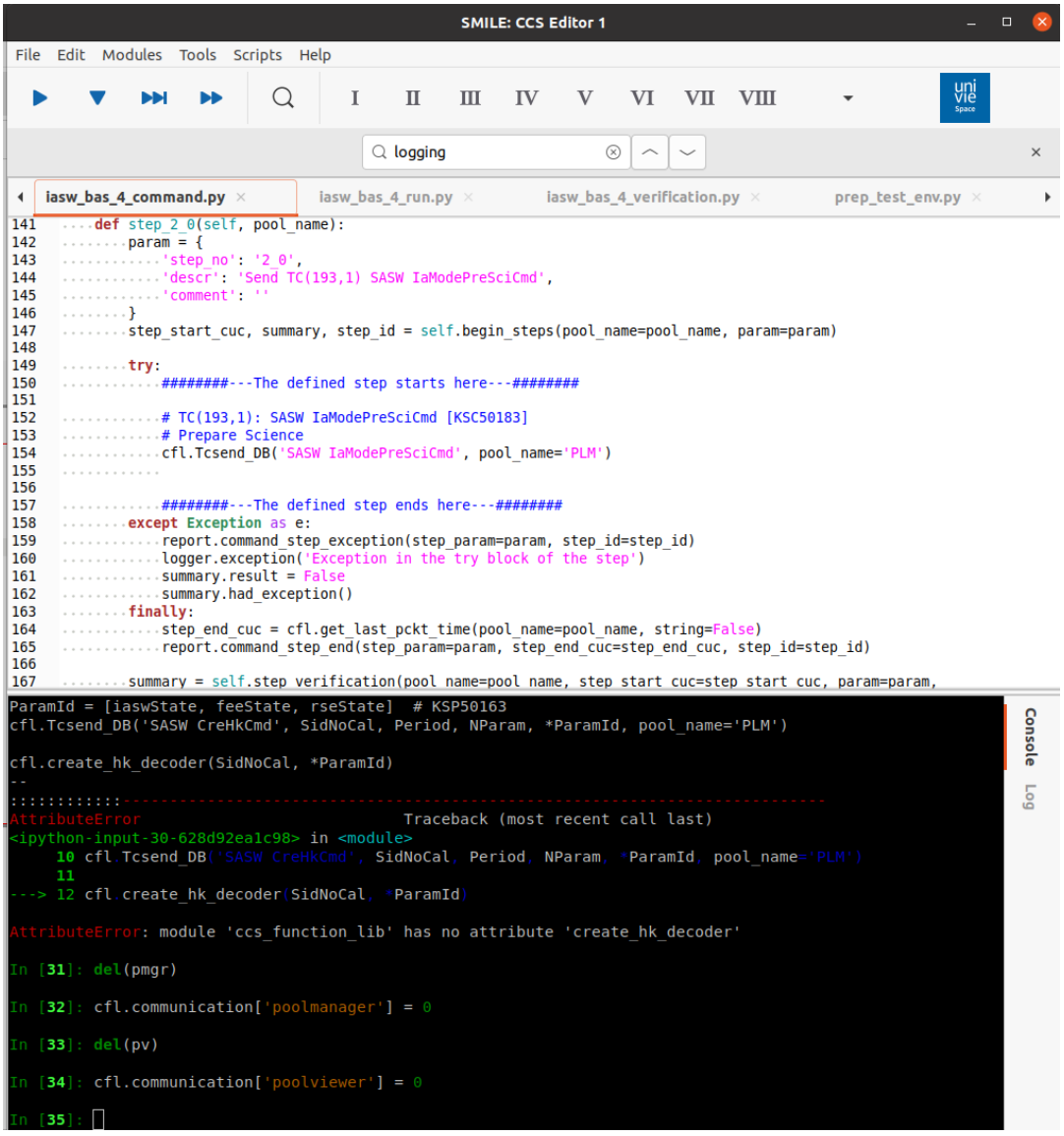

FIGURE A.2: The test scripts in the CCS.

A.6.5 Verification

This tab contains a list of common commands to check if the correct reactions to the TCs occur. The code of the verification can be dragged and dropped into text fields.

A.6.6 Log View

Gives an overview of the processes within the TST. This tab does not give information about the testing itself. To get information about the testing process the pool viewer is the correct tool.

A.6.7 JSON View

This tab shows the JSON file of the current test.

A.7 The Progress viewer

A.7.1 How to choose files

As the name suggests, the progress viewer shows the progress made in an individual test. The test can be run more than once. As long as the name stays the same its result and data will be saved in the same file. The data shown is extracted from three files: a $< testname > -TS - .json$ file, a $< testname >$ *_command.log* file, and a $<$ *testname* $>$ *_verification.log* file. These files can be chosen individually by clicking the Label right next to the text field describing which file should be chosen in the respective line. If the names of the files have not been changed since their generation by the TST, all three files can be chosen by clicking Application in the menu bar. This allows to select a .json file. The program will then choose the command and verification .log files automatically if they have the same <*testname*> as the .json file.

A.7.2 Presentation of Test Results

The progress viewer shows all test runs created under the same name and saved into the same files. These test runs are presented in two modes.

Sort by Steps

This is the default view. Every run is presented in a drawer that can be opened to show all steps within this run.

Sort by Run

By changing the switch button to "Sort by steps", the Progress Viewer shows the results sorted by steps. Every step has its drawer. When expanded, the drawer lists the result of every run for the chosen step.

Bibliography

Branduardi-Raymont, Graziella et al. (2018a). *SMILE Solar wind Magnetosphere Ionosphere Link Explorer*. ESA.

— (2018b). *SMILE Solar wind Magnetosphere Ionosphere Link Explorer*. ESA, pp. 11 –16.

— (2018c). *SMILE Solar wind Magnetosphere Ionosphere Link Explorer*. ESA, pp. 19 –20.

— (2018d). *SMILE Solar wind Magnetosphere Ionosphere Link Explorer*. ESA, pp. 31 –33.

— (2018e). *SMILE Solar wind Magnetosphere Ionosphere Link Explorer*. ESA, pp. 57 –63.

— (2018f). *SMILE Solar wind Magnetosphere Ionosphere Link Explorer*. ESA, pp. 56 –57.

— (2018g). *SMILE Solar wind Magnetosphere Ionosphere Link Explorer*. ESA, pp. 35 –39.

— (2018h). *SMILE Solar wind Magnetosphere Ionosphere Link Explorer*. ESA, pp. 40 –46.

— (2018i). *SMILE Solar wind Magnetosphere Ionosphere Link Explorer*. ESA, pp. 47 –51.

— (2018j). *SMILE Solar wind Magnetosphere Ionosphere Link Explorer*. ESA, pp. 51 –55.

Buggey, Thomas (2021). "Effects Of The Space Environment On The Performance Of SMILE SXI CCDs". Unpublished. URL: https://oro.open.ac.uk/81264/.

CAS (2022). *Overview*. URL: http://english.cssar.cas.cn/smile/overview/ (visited on 03/08/2023).

CCSDS (2023). *OVERVIEW OF SPACE COMMUNICATIONS PROTOCOLS*. CCSDS.

Cechticky, V., R. Ottensamer, and A. Pasetti (Sept. 2015). "Flight Software Development for the CHEOPS Instrument with the CORDET Framework". In: *DASIA 2015 - DAta Systems in Aerospace*. Ed. by L. Ouwehand. Vol. 732. ESA Special Publication, 43, p. 43.

Christian Reimers Roland Ottensamer, Franz Kerschbaum (2019a). *SMILE SXI IASW, Software Architectural Design*. Institut für Astrophysik, Universität Wien, pp. 9 –12.

— (2019b). *SMILE SXI IASW, Software Architectural Design*. Institut für Astrophysik, Universität Wien, pp. 17 –20.

— (2021). *SMILE SXI IASW User Manual*. Institut für Astrophysik, Universität Wien, p. 18.

ECSS (Mar. 2009). *ECSS-Q-ST-30C Dependability*. Tech. rep. ESA, pp. 14 –16.

— (Mar. 2016a). *ECSS-E-ST-70-41C Dependability*. Tech. rep. ESA, pp. 23 –24.

— (Mar. 2016b). *ECSS-E-ST-70-41C Dependability*. Tech. rep. ESA, pp. 18 –20.

— (Mar. 2016c). *ECSS-E-ST-70-41C Dependability*. Tech. rep. ESA, pp. 438 –444.

— (Mar. 2016d). *ECSS-E-ST-70-41C Dependability*. Tech. rep. ESA, pp. 127 –129.

— (Mar. 2017). *ECSS-Q-ST-30C Rev.1 Dependability*. Tech. rep. ESA, pp. 21 –23.

ESA (2019a). *SMILE Light Ion Analyser (LIA)*. URL: https://sci.esa.int/web/smile/-/59195-light-ion-analyser-lia (visited on 03/06/2023).

— (2019b). *SMILE'S OPERATIONAL ORBIT*. URL: https://sci.esa.int/web/smile/-/59199-smile-operational-orbit (visited on 02/15/2023).

— (2020). *Spacecraft*. URL: https://sci.esa.int/web/smile/-/59139-spacecraft (visited on 02/22/2023).

— (2021). *Document Tree*. URL: https://ecss.nl/standards/ecss-document-tree-and-status/ (visited on 04/26/2023).

ESA (2022). "A beginners guide to ECSS". URL: https://dec.dmrid.gov.cy/dmrid/dec/ws_dec.nsf/all/4290AA45DACE15BBC2258841002C15A0/$file/CIP-IPS%20ECSS%20presentation_final%2010.05.2022.pdf.

— (2024). *Smiles all round: Vega-C to launch ESA solar wind mission*. URL: https://www.esa.int/Science_Exploration/Space_Science/Smile/Smiles_all_round_Vega-C_to_launch_ESA_solar_wind_mission (visited on 09/25/2023).

Hanslmeier, Arnold (2004a). *The Sun and Space Weather*. Kluwer Academic Publishers, pp. 84 –91. ISBN: 1-4020-0684-5.

— (2004b). *The Sun and Space Weather*. Kluwer Academic Publishers, pp. 193 –195. ISBN: 1-4020-0684-5.

— (2004c). *The Sun and Space Weather*. Kluwer Academic Publishers, pp. 204 –207. ISBN: 1-4020-0684-5.

Hsu, Tung-Shin and Robert McPherron (July 2004). "Average characteristics of triggered and nontriggered substorms". In: *Journal of Geophysical Research* 109. DOI: 10.1029/2003JA009933.

Hubert, B. et al. (2009). "Statistical properties of flux closure induced by solar wind dynamic pressure fronts". In: *Journal of Geophysical Research: Space Physics* 114.A7. DOI: https://doi.org/10.1029/2008JA013813. eprint: https://agupubs.onlinelibrary.wiley.com/doi/pdf/10.1029/2008JA013813. URL: https://agupubs.onlinelibrary.wiley.com/doi/abs/10.1029/2008JA013813.

Liu, Xinyu et al. (Jan. 2024). "Orbit Transfer Design and Analysis for the SMILE Satellite". In: *Journal of Spacecraft and Rockets* 61, pp. 1–11. DOI: 10.2514/1.A35790.

Roland Ottensamer Christian Reimers, Franz Kerschbaum (2017). *SMILE/SXI APPLICATION SW SOFTWARE DEVELOPMENT PLAN*. Institut für Astrophysik, Universität Wien.

SCOS-2000-Team (2010a). *SCOS-2000 Database Import ICD*. Tech. rep. ESA, pp. 12 – 16.

— (2010b). *SCOS-2000 Database Import ICD*. Tech. rep. ESA, p. 207.

Vaclav Cechticky, Marcel Opprecht (Jan. 2022). *CORDET Framework*. https://github.com/pnp-software/cordetfw. Accessed: 2023-01-19.

Willy Benz David Ehrenreich, Christopher Broeg (2013). *CHEOPS CHaracterising ExOPlanet Satellite*. ESA.

Winkler, Stefan (2020). *Automated qualification test framework for the European Space Agency mission CHEOPS (Characterizing Exoplanet Satellite)*.

Wittman, Martina (2024). *SNMP Explained: What you Must Know About Monitoring via MIB and OIDs*. URL: https://blog.paessler.com/snmp-monitoring-via-oids-mibs (visited on 10/22/2024).