

POLITECNICO DI TORINO

Dipartimento di Ingegneria Meccanica ed Aerospaziale

Tesi di Laurea Magistrale

**CubeSat Control Centre for the
management of telemetry, telecommand
and operations based on the CCSDS
standards**



Relatori

Prof. Sabrina Corpino

Ing. Fabrizio Stesina

Candidato

Antonio Esposito

Matr. 243176

Ottobre 2019

Le vent se lève,
il faut tenter de vivre
Paul Valéry

Abstract

CubeSats are becoming an important reality in space exploration both in academia and industry. The increasing capabilities of this kind of system enable new kind of missions able to fulfill more diverse mission goals. Despite their reduced complexity, spacecraft operations do not scale down with the size. Hence, training future spacecraft operators via CubeSat operations would be an important method to increase the effectiveness of future operations with already trained experts.

To tackle these issues, the thesis presents Cubesat Control Centre (C3) is an innovative ground segment to support Cubesat Operations directly from Politecnico di Torino. It is composed by a ground station and a control centre operated by students and non-professional operators.

Control Centre's software for the unpacking of telemetries data and the scheduling of operations are implemented by Python. The aim of this software is to provide an interface that can read automatically telemetries data and using scheduled protocols for sending telecommands. The process is monitored by several consistency checks which identify the correct acquisition of the package, identify the type of package, extract the data and convert them into an engineering language.

The main component of this process is a database that helps software and consistency checks to perform their tasks. The operator can see an interface where the following are displayed: type of arriving packet, check of correct acquisition and validation of packets, binary code from satellite, engineering value of arriving data. From the interface the operator can also select the mission protocols to use for the mission, can simulate the command to send and sends command directly to satellite when it is in visibility.

C3 is one of the one of the first academic control centres in Italy using the ESA CCSDS's standards and it is a useful facility for educational and research purposes. Control Centre is divided into three areas: **Flight Operation**: Where CubeSats receives telecommands, sends telemetries and where the team evaluate follow up operations; **Payload Data Ground Segment**: where the management of

payload data happens; **User Segment:** where consumer could request a series of products by means of a formal request.

By providing to primary mission fast delivery information and high-quality data, sending scheduled commands and assisting in managing requests from stakeholders, C3's control centre is a valid resource and cheap support for operation, enabling great learning opportunities and effacement operations with an open source vision to support CubeSat community.

CONTENTS

| | |
|--|----|
| Contents | 5 |
| List of Figures | 7 |
| List of Tables..... | 10 |
| Abbreviations | 11 |
| 1. Introduction | 13 |
| 2. The C3 Project | 15 |
| 2.1 Design Approach: Functional Analysis | 15 |
| 2.2 Design Approach: State Analysis..... | 21 |
| 2.3 Design Approach: Risk Analysis and Management | 23 |
| 2.4 Design Approach: Baseline Proposal..... | 27 |
| 3. The Control Centre Design | 31 |
| 3.1 Control Centre Architecture..... | 32 |
| 3.2 Control Centre CCSDS Standards Overview | 37 |
| 3.2.1 CCSDS Overview: Telemetry Construction..... | 38 |
| 3.2.2 CCSDS Overview: Telecommand Construction | 43 |
| 3.2.3 CCSDS Overview: Image Compression Construction | 45 |
| 3.2.4 CCSDS Overview: Mission Planning & Scheduling..... | 50 |
| 4. Software Philosophy & Architecture..... | 52 |
| 4.1 Python Multithreading Approach | 53 |
| 4.2 SAT SIM Overview: Software Architecture | 56 |
| 4.3 SAT SIM Overview: Telemetry Branch | 57 |
| 4.4 SAT SIM Overview: Telecommand Branch..... | 62 |
| 4.5 SAT SIM Overview: Images Management Branch..... | 65 |
| 4.6 SAT SIM Overview: Interface Overview | 67 |
| 5. Test & Validation..... | 79 |
| 5.1 Objectives & Requirements | 80 |
| 5.2 Test Sessions | 82 |
| 5.2.1 Test Session: Debugging Process | 84 |
| 5.2.2 Test Session: Software Profiling..... | 85 |
| 5.2.3 Test Session: TM/TC Nominal Profile (Tc-001)..... | 89 |
| 5.2.4 Test Session: TM/TC Error Profile (Tc-002) | 91 |
| 5.2.5 Test Session: Image Profile (Tc-003) | 95 |

| | | |
|-------|---|-----|
| 5.2.6 | Test Session: Scheduler Test (Tc-004) | 98 |
| 5.3 | Results | 101 |
| 6. | Conclusions | 104 |
| | References..... | 106 |
| | Appendix | 107 |

LIST OF FIGURES

| | |
|--|----|
| Figure 1: A traditional CubeSat Mission Architecture, adapted from [2] | 14 |
| Figure 2: First Level of the Functional Tree | 17 |
| Figure 3: Product Tree (First Level) | 18 |
| Figure 4: RF System Block Diagram | 19 |
| Figure 5: Tracking System Block Diagram | 20 |
| Figure 6: Control Centre System Block Diagram | 20 |
| Figure 7: Risk Analysis Flowchart | 24 |
| Figure 8: Risk Index and Magnitude Scheme (Before Risk Reduction Action) | 25 |
| Figure 9: Index and Magnitude Scheme (After Risk Reduction Action) | 26 |
| Figure 10: Trade-Off Analysis Results | 28 |
| Figure 11: Cost Budget Diagram for Line | 29 |
| Figure 12: Cost Budget for Discipline | 30 |
| Figure 13: C3 Control Centre Architecture | 32 |
| Figure 14: Control Centre Acquisitions | 33 |
| Figure 15: Processing System Architecture | 34 |
| Figure 16: Control Centre sharing blocks | 36 |
| Figure 17: Overview of the Communication Protocol Core | 37 |
| Figure 18: TM Structure | 38 |
| Figure 19: CCSDS TM Packet Data System | 38 |
| Figure 20: Global Packet Structure | 39 |
| Figure 21: Source Packet Format [5] | 39 |
| Figure 22: TC Structure | 43 |
| Figure 23: TC Packet Format [8] | 43 |
| Figure 24: Structure Overview of the Decoding Process | 44 |
| Figure 25: CLTU Starting Sequence Pattern according to [7] | 45 |
| Figure 26: Functional Block Diagram [10] | 46 |
| Figure 27: Image Compression Model: Encoder and Decoder | 46 |
| Figure 28: Block and Group structure of DWT transformed data | 48 |
| Figure 29: Structure of an Image Packet | 48 |
| Figure 30: Bit Plane Encoder [10] | 49 |
| Figure 31: Example of Federated Planning for a Science Mission [15] | 50 |
| Figure 32: Planning Information Flow | 50 |
| Figure 33: Example of how Python can figure out the type at run-time | 53 |

| | |
|--|----|
| Figure 34: Different stages of a process [11] | 54 |
| Figure 35: Comparison between process with 1 thread (left) and process with multiple thread (right) [11] | 54 |
| Figure 36: Thread complete life cycle | 55 |
| Figure 37: SAT SIM Architecture | 56 |
| Figure 38: Example of SAT SIM Databases | 57 |
| Figure 39: SAT SIM - TM Branch | 58 |
| Figure 40: TM Gen Module | 58 |
| Figure 41: Packet Gen Module | 59 |
| Figure 42: TM Queue Gen Module | 60 |
| Figure 43: GS SIM Extraction Module | 60 |
| Figure 44: GS SIM Control Loop Module | 61 |
| Figure 45: SAT SIM - TC Branch | 62 |
| Figure 46: TC Gen Module | 63 |
| Figure 47: SAT SIM Extraction Module | 63 |
| Figure 48: New TM Gen after the command execution | 64 |
| Figure 49: TC Display module | 64 |
| Figure 50: IMG SIM Branch | 65 |
| Figure 51: IMAGE_SIM Module | 66 |
| Figure 52: Image Display Module | 67 |
| Figure 53: SAT SIM Interface architecture | 68 |
| Figure 54: SAT SIM Login/Register Interface | 68 |
| Figure 55: SAT SIM Login Interface | 69 |
| Figure 56: SAT SIM Registration Interface | 69 |
| Figure 57: SAT SIM main interface | 70 |
| Figure 58: SAT SIM TM Interface | 70 |
| Figure 59: Queue of the incoming packets | 71 |
| Figure 60: SAT SIM TM Display interface | 71 |
| Figure 61: SAT SIM TM Packet Visualization | 72 |
| Figure 62: Packet Archive | 72 |
| Figure 63: Display of a specific TM packet | 73 |
| Figure 64: TC Interface | 73 |
| Figure 65: TC Manage Interface | 74 |
| Figure 66: TC Packet Generation Module Interface | 74 |
| Figure 67: Correct execution of the command | 75 |

| | |
|---|-----|
| Figure 68: Command Structure | 75 |
| Figure 69: Scheduler Interface Command Choose | 77 |
| Figure 70: Command detail window | 77 |
| Figure 71: Scheduler Queue Interface | 77 |
| Figure 72: Example of a schedule execution | 78 |
| Figure 73: The V-model of the V&V process | 81 |
| Figure 74: Test evaluation flowchart | 82 |
| Figure 75: The Debugging Process | 84 |
| Figure 76: cProfiler output of SAT SIM main | 85 |
| Figure 77: cProfiler output of the Interface main | 86 |
| Figure 78: Call graph of the SAT SIM main | 87 |
| Figure 79: Call graph of Interface main | 88 |
| Figure 80: Tc Nominal TM generation flowchart | 89 |
| Figure 81: Tc-001 Nominal TM packets | 90 |
| Figure 82: Tc-001 Nominal TC interface | 91 |
| Figure 83: Tc-002 Error TM generation flowchart | 91 |
| Figure 84: Tc-002 Error TM packets | 92 |
| Figure 85: Alert Notification | 93 |
| Figure 86: Detail of the incoming message | 93 |
| Figure 87: TC packets generation | 94 |
| Figure 88: New Tm correction acquired | 94 |
| Figure 89: Incorrect Packet Acquisition | 95 |
| Figure 90: Tc-003 Image management flowchart | 95 |
| Figure 91: Tc-003 Image compression Interface | 96 |
| Figure 92: Tc-003 Frame Extraction | 96 |
| Figure 93: Tc-003 Images Display | 97 |
| Figure 94: Tc-004 Schedule test flowchart | 98 |
| Figure 95: Tc-004 Recognition and display of the pre-set commands | 98 |
| Figure 96: Tc-004 Generation of the Schedule queue | 99 |
| Figure 97: Tc-004 Schedule queue executed correctly | 99 |
| Figure 98: Tc-004 warning message for command with same priority number | 100 |
| Figure 99: Tc-006 Register Interface | 102 |
| Figure 100: Tc-006 Registration Success | 102 |
| Figure 101: Tc-006 Login Interface | 103 |
| Figure 102: Tc-006 Login Success | 103 |

| | |
|--------------------------------------|-----|
| Figure 103: SAT SIM Call Graph (1) | 107 |
| Figure 104: SAT SIM Call Graph (2) | 108 |
| Figure 105: SAT SIM Call Graph (3) | 109 |
| Figure 106: Interface Call Graph (1) | 110 |
| Figure 107: Interface Call Graph (2) | 111 |
| Figure 108: Interface Call Graph (3) | 112 |
| Figure 109: Interface Call Graph (4) | 113 |
| Figure 110: Interface Call Graph (5) | 114 |
| Figure 111: Interface Call Graph (6) | 115 |

LIST OF TABLES

| | |
|--|-----|
| Table 1: Mission Requirements | 16 |
| Table 2: Mission Planning | 21 |
| Table 3: Phase-Scenario Description | 23 |
| Table 4: Risk Magnitude and Proposed Action for Individual Risk according to [4] | 26 |
| Table 5: C3 Cost Budget for Line and Cost Budget for Discipline | 29 |
| Table 6: C3 Mass Budget | 30 |
| Table 7: Control Centre mission requirements | 31 |
| Table 8: Advantages and Disadvantages of using threads | 55 |
| Table 9: Software Requirements | 80 |
| Table 10: Test Classification | 83 |
| Table 11: Test Cases Results | 101 |

ABBREVIATIONS

ADCS: Attitude Determination and Control System
AOSTF: Advanced Orbiting System Transfer Frame
APID: Application Process ID
BATT: Batteries
BPE: Bit Plane Encoder
C3: CubeSat Control Centre
CADU: Channel Access Data Unit
CAM: Navigation camera
CCSDS: Consultative Committee for Space Data System
CLTU: Communication Link Transmission Unit
COTS: Commercial Off-the-Shelf component
DCT: Discrete Cosine Transform
DSP: Digital Signal Processor
DWT: Discrete Wavelet Transform
ECSS: European Cooperation for Space Standardization
EPS: Electrical Power System
FMECA: Failure Modes, Effects and Criticality Analysis
GS: Ground Station
HYPER: Hyperspectral
IDC: Image Data Compression
IMG PKT: Image Packet
KISS: Keep It Simple, Stupid
OBC: On-board Computer
PAY: Payload
PROP: Propulsion System
RF: Radio Frequency System
RW: Reaction Wheels
S\c: Spacecraft bus
SAT SIM: Satellite Simulator
SCHED: Scheduler
SE: System Engineering

TC PKT: Telecommand Packet

TCS: Thermal Control System

TCTF: Telecommand Transfer Frame

THR: Thrusters

TM PKT: Telemetry Packet

V&V: Validation and Verification process

1. INTRODUCTION

Communication between ground stations and CubeSats is a complicated endeavour. There are standardizations that have been set in place by organizations, like *Consultative Committee for Space Data System* (CCSDS), or the *European Cooperation for Space Standardization* (ECSS), to resolve this issue and to simplify the construction of communication systems and to promote their interoperability and uniformities.

The CCSDS is an organization with the aim to define and maintain standards for data systems and to provide communication between them in space. These standards cover a large number of fields for a specialized implementation to fit the need of the project. The ECSS is another organization that takes the implementation deriving from CCSDS recommendations and define the requirements that user must to follow. These requirements are used by space organization as a way to simplify the collaboration between them and organizations and companies in other countries. The ECCS takes some of CCSDS standards and consolidates them into more rigid requirements,[1].

According to these rules and recommendations, the thesis presents the CubeSat Control Centre (C3), an innovative ground segment to support Cubesat Operations directly from Politecnico di Torino following the CCSDS standards.

The major aim of this work of thesis is to develop a control centre to support CubeSat operations focused on CCSDS packet utilization for Telemetry acquisitions and Telecommand generation.

A ground system has two main purposes: to support space segment (spacecraft bus and payloads) and to transmit missions data derived from on-board computer to the mission stakeholders. The same concept is applied to both large and small satellites, such as CubeSats. Then, what are the real needs and the real benefits in developing a Ground Control Centre totally focused on CubeSats? A first answer could be to reduce the costs, but even is important, the expenses related to ground station operations are not the main concern. The most important benefit in developing a CubeSat control centre is the exploitation of university facilities and the national autonomy.

In this perspective, the C3 project was created to offer students and non-professional operators a facility where is possible to manage and control CubeSat missions in complete freedom. Being free to explore different design possibilities allows to extrapolate the better project according to the national infrastructure and needs.

The first step to know the system of interest is to identify the conventional ground station operations taking into account that the ground station is a part of the space mission architecture. Figure 1 shows a typical mission architecture adapted to a CubeSat mission. The architecture is the same for small and large satellites, the only main difference is that the mission control, ground segment and communication control architecture could join into a single segment.

To achieve the main functions (send telecommand, receive telemetry, track the CubeSats and process their data) having only one ground station is generally enough, but the project design must follow special requirements and characteristics to work effectively like a ground segment.

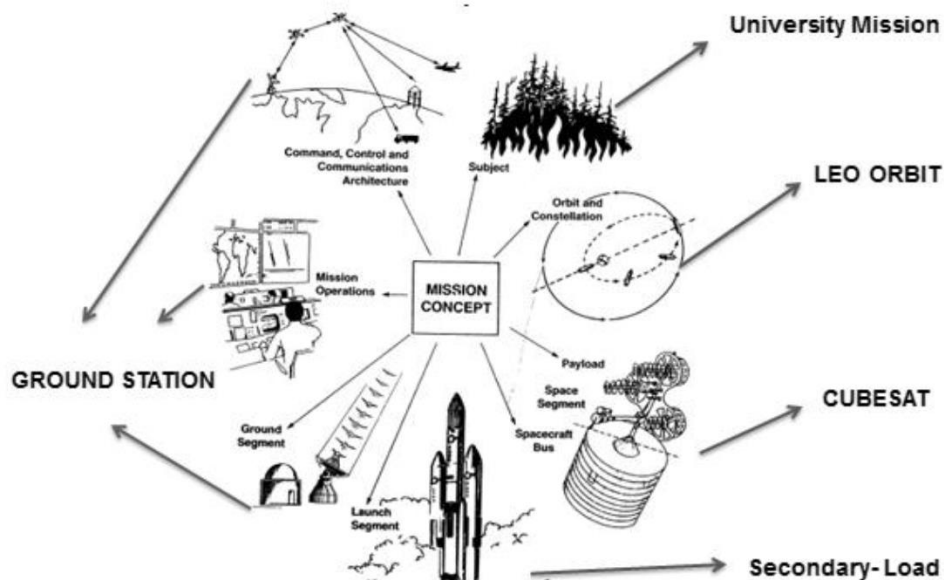


Figure 1: A traditional CubeSat Mission Architecture, adapted from [2]

For the design project of the ground segment is necessary to analyze the space segment parameters to support them and to derive some functional requirements.

2. THE C3 PROJECT

The objectives pursued by the team on the C3 project are the following:

- To have a Control Centre at Politecnico di Torino for the communication between the space segment, ground operators and users
- The CubeSat Control Centre (C3) aims to support CubeSats mission operations with capabilities and performance beyond traditional radio amateur C3s, but at a lower cost than professionally driven ground segment (control centre and ground station)
- C3 is the mission, spacecraft bus and payload control centre of the SROC mission and manages its payload while interfacing with other stations/centres involved in the Space Rider Mission.

2.1 DESIGN APPROACH: FUNCTIONAL ANALYSIS

According to the system engineering approach, the project followed a list of process steps that helped to better understand the real needs of the design.

The first step of the System Engineering analysis consists basically of the need statement. Once they are established, the identification of stakeholders and requirements definition is conducted through iterative processes and generation of goal and mission objectives. This stage is concluded by generating the concept of operations, which shows the system behavior into their operational work.

The second step is related to the requirement analysis where the stakeholders' needs are converted into requirements and they are analyzed qualitatively and quantitatively to achieve the better design of the project.

In the last steps, the functional analysis and the life cycle analysis are conducted to conceptualize all the systems behaviors and their functions, and to evaluate the operating environment in order to specify all the systems in more details. At the end, the result of this analysis is the project baseline proposal in which all the systems are assigned to their physical components.

The result of the mission needs led the team to the first requirements analysis. In this phase the C3's requirements identify the functions, physical

characteristics or quality factors that limit the needs of the product or process for which a solution is pursued. Therefore, Table 1 shows some of the mission requirements identified for the C3 project according to the division indicated in the ECSS standards [3].

| MISSION REQUIREMENTS | |
|----------------------|---|
| ID | Requirement Text |
| C3-MIS--1 | <i>C3 shall support Cubesats mission operations from ground when they are in LEO orbit</i> |
| C3-MIS--2 | <i>C3 shall guarantee the management of the operations for PolITO/Cubesat Team missions</i> |
| C3-MIS--3 | <i>C3 shall be located at Politecnico di Torino in TBD location</i> |
| C3-MIS--4 | <i>C3 shall manage mission data from CubeSats payload</i> |
| C3-MIS--7 | <i>C3 shall manage CubeSats housekeeping data</i> |
| C3-MIS--8 | <i>C3 shall manage telecommands to CubeSats</i> |
| C3-MIS--9 | <i>C3 shall guarantee the management of the planning activities on the CubeSats</i> |
| C3-MIS--10 | <i>C3 shall be operated by students and non-professional operators</i> |
| C3-MIS--11 | <i>C3 shall cost less than 30K</i> |
| C3-MIS--12 | <i>C3 shall implement at least E2 level of autonomy</i> |
| C3-MIS--13 | <i>C3 shall be designed manufactured, integrated and tested in 35 months from the KOM</i> |
| C3-MIS--14 | <i>C3 shall be flexible with respect to the protocols, the frequency bands, the type of signals</i> |
| C3-MIS--15 | <i>C3 shall operate in UHF,VHF, S-band and X-band</i> |
| C3-MIS--16 | <i>C3 shall manage data,voice, image and video</i> |
| C3-MIS--17 | <i>C3 shall satisfy applicable emission regulations (ITU, Ministry of Communications, ...)</i> |

Table 1: Mission Requirements

To fill the different categories of requirements a functional analysis is mandatory to better identify the correct functions of the project and to understand what kind of systems could achieve those functions.

For the ground station, the blocks describe its main function with shallow details. However, it is important to say that at this stage of the system development, the entire operation of the planned C3 is already covered.

According to the functional tree shown in Figure 2, it is possible to build the operational mode and state diagrams for the station, where the states are the operating levels of the systems characterizing the ground segment and the modes are the functions that run the system under these levels (e.g. operative status and data acquisition mode). This kind of analysis increases system knowledge. This is helpful for a better assignment of functions to physical components for the creation of the product tree.

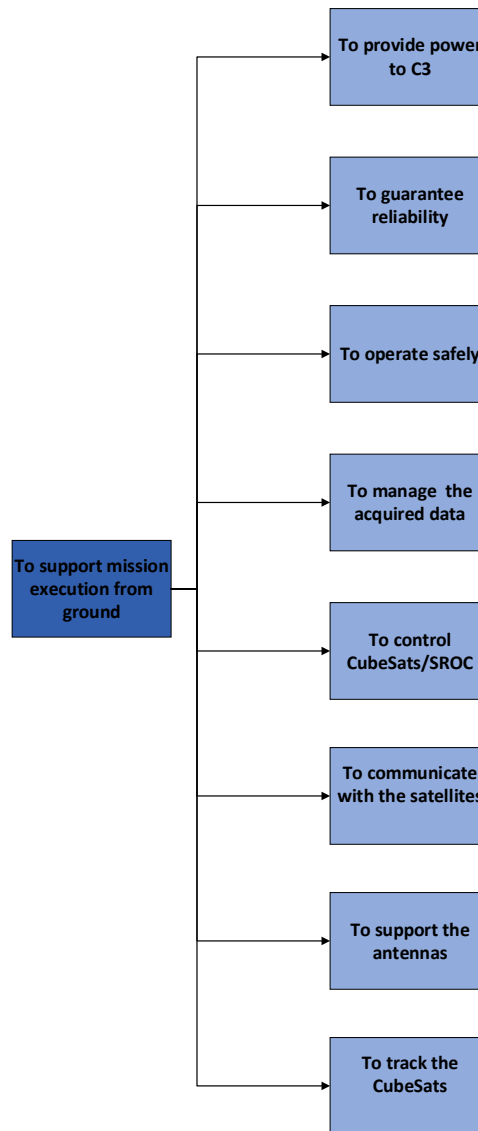


Figure 2: First Level of the Functional Tree

The functional tree, in particular the last level, made it possible to identify the subsystems that make up the ground segment. These elements, as seen in the product tree in Figure 3, determine the characteristics of the system of interest and therefore, the better architectural solution for the ground station (Figures 4-6).

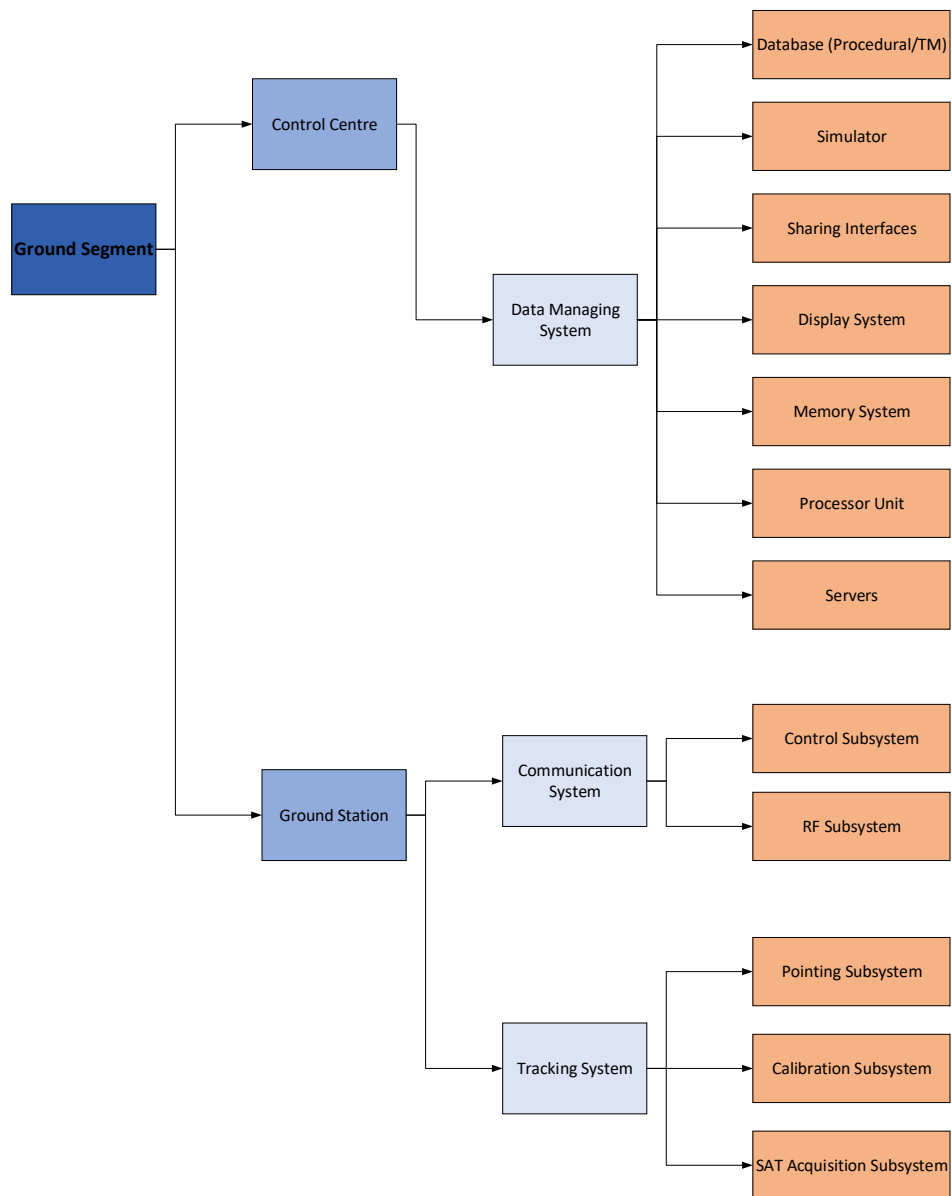


Figure 3: Product Tree (First Level)

Ground Segment

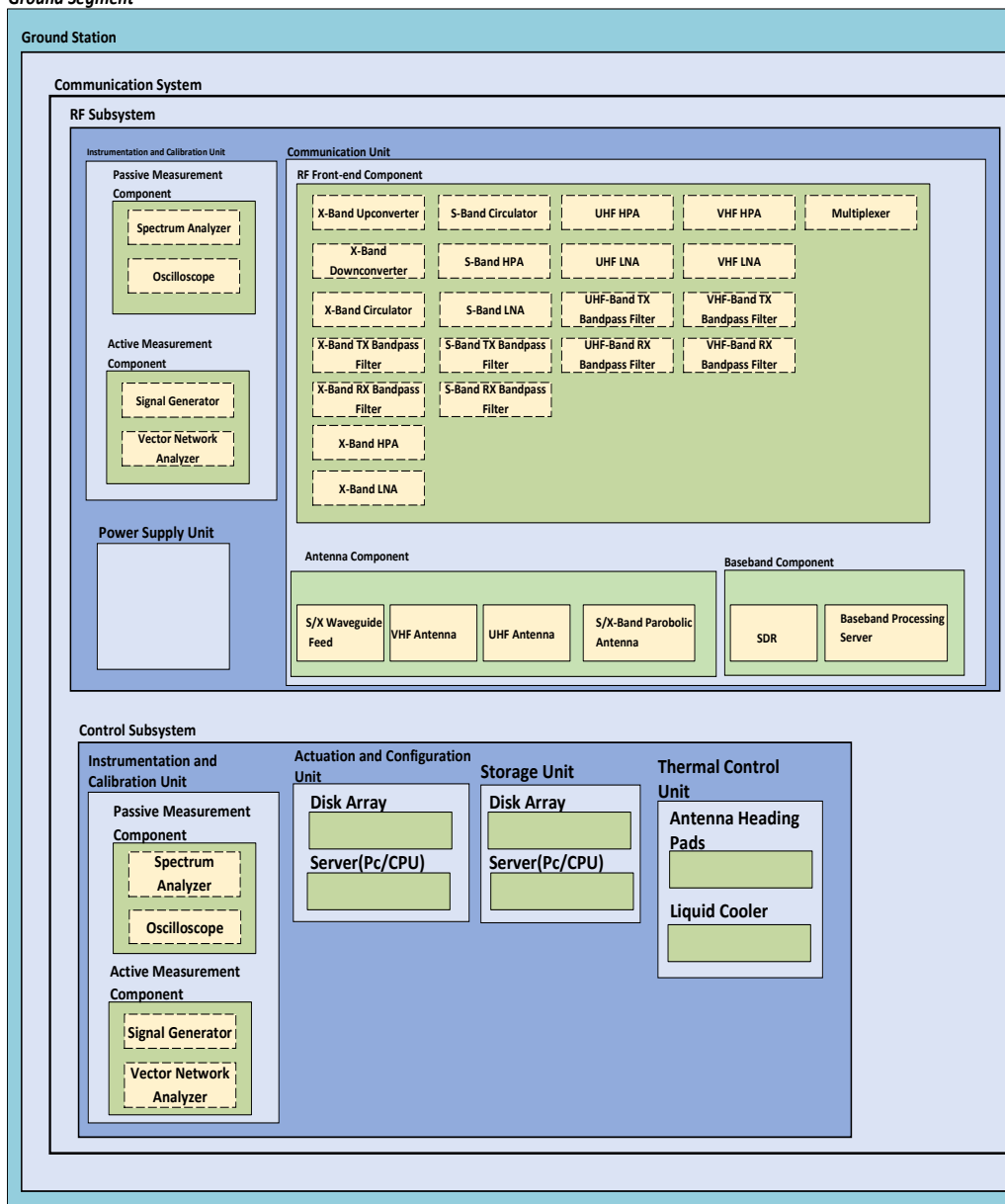


Figure 4: RF System Block Diagram

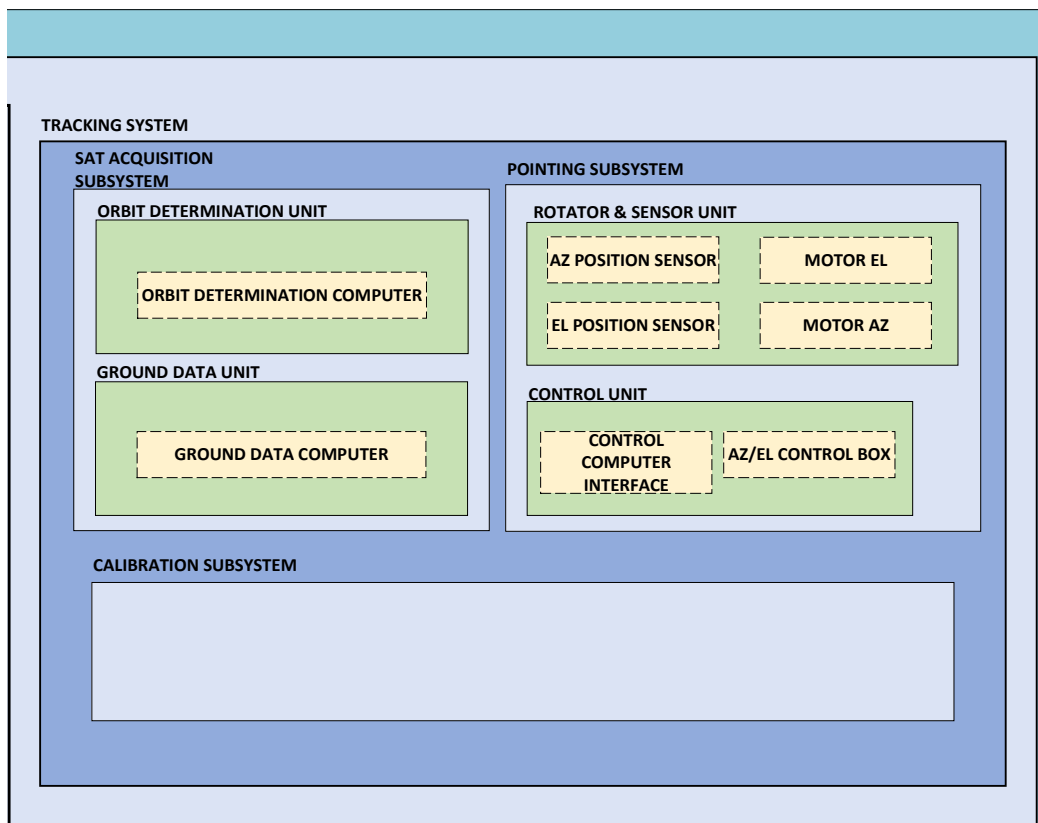


Figure 5: Tracking System Block Diagram

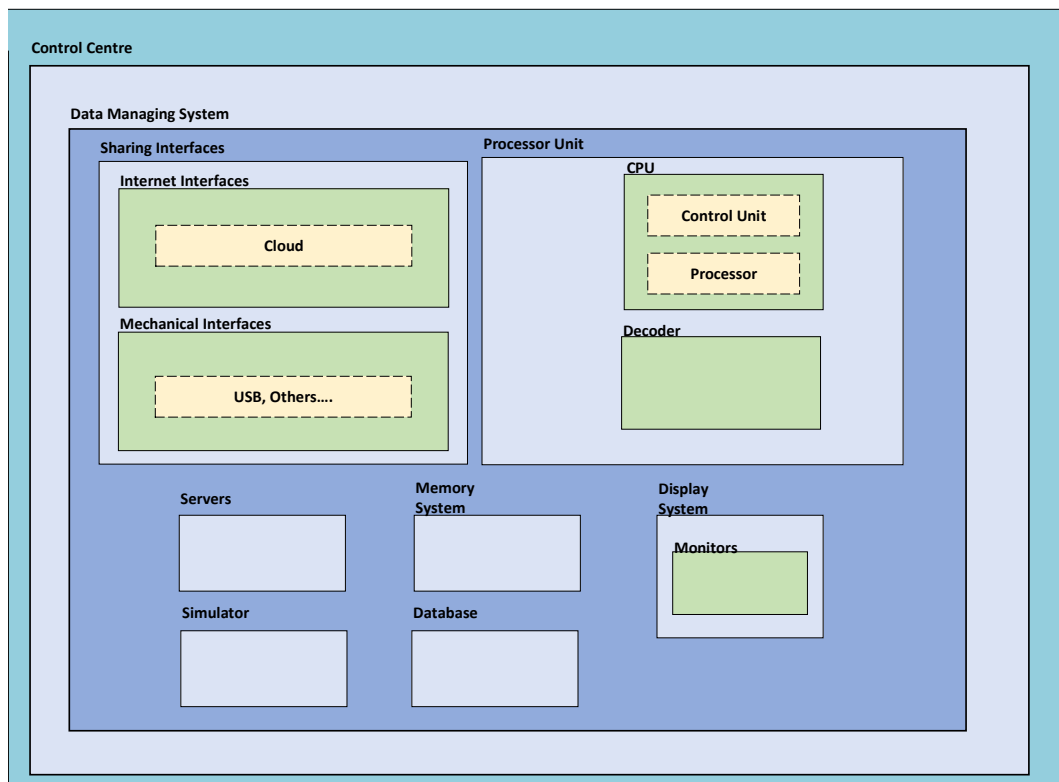


Figure 6: Control Centre System Block Diagram

2.2 DESIGN APPROACH: STATE ANALYSIS

The operative modes of the ground segment are also identified, during the design process, to better estimate the project mission plan and all the scenarios in which the station will operate. During this analysis, as seen in Table 2, four phases are identified; each phase is characterized by scenarios that describe in detail the phase, and each scenario is described by objectives, constraints, and duration.

| Phase | Scenario |
|-------------------------|---|
| 1)Visibility On | Real time operation |
| 2)Visibility Off | Setup - Preparation for the satellite visibility |
| | Post processing activity |
| | Checkout communication channel (internal and with other ground station) |
| | Post processing analysis |
| | Check of off nominal events |
| | Mission planning activity |
| | Post processing data sharing |
| 3)LEOP | Check out |
| | Telemetry Link |
| | Command Link |
| 4)Safe Mode | Safe scenario |

Table 2: Mission Planning

Four macro operative mode have been identified:

- **Visibility On** represents the phase in which the satellite is in visibility of the ground station and real-time operations are carried out such as receiving telemetry by the satellite and sending remote control from the station.
- **Visibility Off** is the phase in which the satellite is not in visibility of the ground station. This phase includes the preparation of the subsystems for the next passage of the satellite, communication with other stations involved in the mission for the exchange of information, and post-processing activities.

- **LEOP (Launch and Early Orbit Phase):** this phase is one of the most critical phases of a mission. Spacecraft operations engineers take control of the satellite after it separates from the launch vehicle up to the time when the satellite is safely positioned in its final orbit. During this period, the operators work 24 hours a day to activate, monitor and control the various subsystems of the satellite, including the deployment of any satellite appendages (antennas, solar array, reflector, etc.), and undertake critical orbit and attitude control manoeuvres.
- **Safe Mode:** represents the security status of the ground segment in the presence of the catastrophic, critical or off-nominal events. It is the ability of the ground station, in the presence of a failure, to secure operators and all subsystems and to correctly protect the data.

Each phase is characterized by different scenarios and each of them, as seen in Table 3, is described by:

- **General Description:** a description of the scenario.
- **Initial Condition:** a description of the condition for the start of the scenario.
- **Final Condition:** a description of the condition for the end of the scenario.
- **Environment:** a description of the environment in which the scenario is executed.
- **Top Level Objectives:** a list of the high level objectives characterizing the scenario.
- **Required I/F with other systems:** a description of all the interfaces required for the correct execution of the scenario.
- **Duration:** duration of the scenario
- **Constraints:** a list of all the constraints and requirements that describe the scenario.
- **Potential Off-Nominal Events:** a description of the possible off-nominal events related to that specific scenario.

| Real time operation | |
|---------------------------------|---|
| Characteristics | Description |
| General description | Uplink and Downlink operations in satellite visibility |
| Initial Conditions | Satellite comes in visibility |
| Final Conditions | Satellite comes out visibility |
| Environment | Earth environment |
| Top Level Objectives | Establish communication link Up/Down link operations Track satellite passage To reduce space loss |
| Required I/F with other systems | Communication system I/F Tracking system (Move the Antenna) Communication system I/F Control Centre Tracking System I/F Control Centre |
| Duration | 8/10 min (LEO Orbit) |
| Constraints | All operation must have a duration less than 10 minutes |
| Potential Off-Nominal Events | Loss of communication link Down/Up link operation failure |

Table 3: Phase-Scenario Description

The identification of the operative mode and the relative operational requirements will allow to establish the schedule generation for the telecommand (TC) to communicate with CubeSats even when these are not in visibility of the ground station.

2.3 DESIGN APPROACH: RISK ANALYSIS AND MANAGEMENT

During the design process and mission analysis the risk analysis and management is essential because it allows to identify which are the several possible failures during the mission, in order to prevent them. Although it is almost not possible to avoid a risk, one of the aims of the study is to try to limit any possible damage in order to complete successfully the mission. Therefore, the study has been conducted in reference to the possible failures of the C3's system. Furthermore, this allows to compare them and to highlight which risks would lead to compromise the mission's feasibility or the achievement of mission aim. A study of catastrophic hazards has therefore been carried out; it has enabled the identification of possible project corrections.

The risk analysis is based on **Failure Modes, Effects and Criticality Analysis (FMECA)** that is probably the most widely used and the most effective design reliability analysis method. It is a bottom-up analysis of all possible ways in which a component may fail, considering every failure mode one by one. This analysis is performed according to the following steps:

- Identify each possible component in the system;
- Determine all possible failures for the component;
- Determine all the credible causes for each failure;
- Determine the worst effect on the system considering every mission phase;
- Determine severity and likelihood of each failure;
- Determine criticality of each failure (criticality matrix);

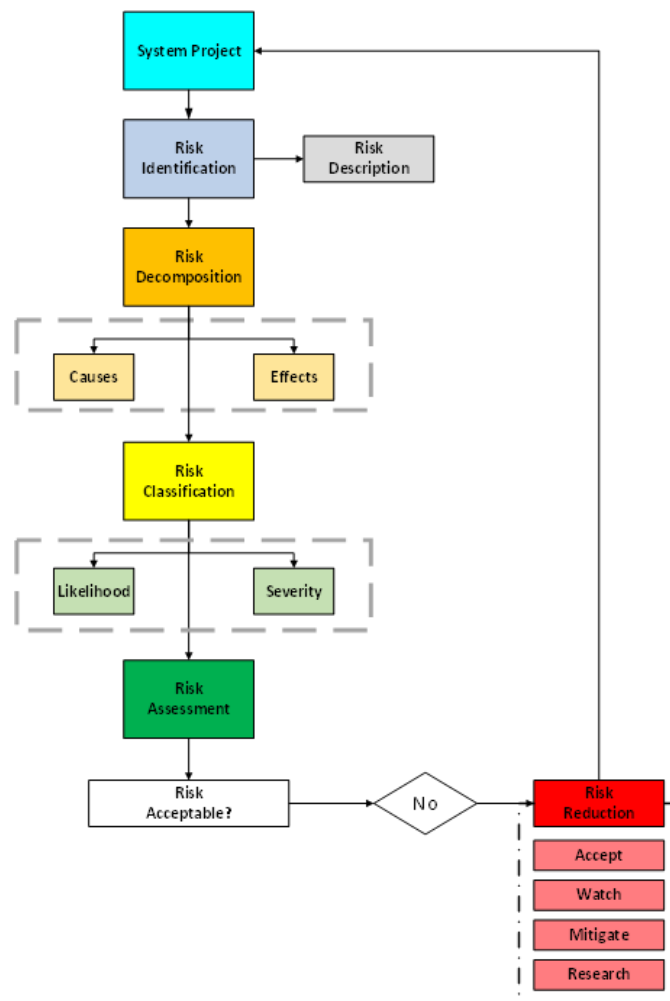


Figure 7: Risk Analysis Flowchart

The fundamental actions performed could be summarised in the following way: **Risk Identification, Risk Decomposition, Risk Classification and Risk Reduction.**

During the **Risk Identification**, any risk that may arise is identified and divided into categories lifetime, schedule, costs and component.

In the **Risk Decomposition**, each risk has therefore been divided into cause and effect to know which are the failures that could lead to the worst consequences. An inductive method is used so that, from the causes of the possible failures, it has been possible to move on to the effects.

The **Risk Classification** is the most fundamental part of the risk analysis. It consists of a classification and assessment of risks by assigning to each one its probability of occurrence and its severity of consequences which are respectively the frequency of its occurrence and the importance of the consequences of an event on the mission purpose. The likelihood and severity index, established by [4], has been assigned to every risk.

| | | SEVERITY | | | | |
|--|---|----------------|--------------------------------|--|---|---|
| | | Negligible (1) | Significant (2) | Major (3) | Critical (4) | Catastrophic (5) |
| L I K E L I H O O D | E | | | | | |
| | D | | | COM-03; | COM-02; | |
| | C | COM-17; | LT-01; SCH-06; | LT-06; SCH-03; SCH-08; COS-02; COM-05; | COM-01; COM-06; COM-38; COM-39; COM-40; COM-41; COM-42; | LT-03; COM-12; COM-19; COM-20; |
| | B | SCH-04; | LT-02; LT-04; LT-10; | SCH-02; SCH-05; COS-01; | SCH-01; SCH-07; SCH-09; COS-03; COM-07 | LT-07; LT-09; COM-04; COM-08; COM-09; COM-14; COM-18; COM-22; COM-23; COM-24; COM-25; COM-35; COM-36; |
| | A | COM-37; | LT-05; COM-11; COM-16; COM-34; | LT-08; LT-11; | COM-13; COM-43; COM-44; COM-45; COM-46; | COM-10; COM-15; COM-21; COM-26; COM-27; COM-28; COM-29; COM-30; COM-31; COM-32; COM-33; |

Figure 8: Risk Index and Magnitude Scheme (Before Risk Reduction Action)

| Risk magnitude | Proposed actions |
|----------------|---|
| Very High risk | Unacceptable risk: implement new team process or change baseline – seek project management attention at appropriate high management level as defined in the risk management plan. |
| High risk | Unacceptable risk: see above. |
| Medium risk | Unacceptable risk: aggressively manage, consider alternative team process or baseline – seek attention at appropriate management level as defined in the risk management plan. |
| Low risk | Acceptable risk: control, monitor – seek responsible work package management attention. |
| Very Low risk | Acceptable risk: see above. |

Table 4: Risk Magnitude and Proposed Action for Individual Risk according to [4]

The actions proposed to mitigate the risks are **Accept**, **Watch**, **Mitigate** and **Research**: they give the risk a trend that allows it to be downgraded according to the standards [4]. **Accept** is a partially preventive action that consists in accepting a problem that cannot be solved in any way. **Watch** is a partially reactive action which foresees the system monitoring and the research solutions for issues that could arise and have not been otherwise prevented. **Mitigate** is a preventive action that provides corrective strategies in order to improve mission project and prevent any possible problem. **Research** is a reactive action that gives possible solutions, even if the problem has not been estimated.

| | | SEVERITY | | | | |
|------------|---|-----------------|--|--|---|---|
| | | Negligible(1) | Significant (2) | Major (3) | Critical (4) | Catastrophic (5) |
| LIKELIHOOD | E | | | | | |
| | D | | | COM-03; | | |
| | C | COM-17; COM-18; | LT-01; SCH-06;COM-02; | LT-03; LT-06; SCH-03; SCH-08; COS-02; COM-05; COM-20; | COM-01; COM-06; COM-38; COM-39; COM-40; COM-41; COM-42; | |
| | B | SCH-04 | LT-02; LT-04; LT-10; | LT-07; SCH-02; SCH-05; COS-01; COM-23; COM-24; COM-25; | SCH-01; SCH-07; SCH-09; COS-03; COM-07 | LT-09; COM-04; COM-08; COM-09; COM-19; |
| | A | COM-37 | LT-05; COM-12; COM-11; COM-16; COM-34; COM-35; | LT-08; LT-11;LT-12 | COM-13; COM-14; COM-43; COM-44; COM-45; COM-46; | COM-10; COM-15; COM-21; COM-22; COM-26; COM-27; COM-28; COM-29; COM-30; COM-31; COM-32; COM-33; COM-36; |

Figure 9: Index and Magnitude Scheme (After Risk Reduction Action)

The risk analysis made it possible to identify the redundancies to be implemented and to arrive at a new physical architecture.

2.4 DESIGN APPROACH: BASELINE PROPOSAL

Three possible proposals of ground architectures are carried out to achieve the better configuration according to requirements and mission objectives. All the designs follow the **KISS (Keep It Simple, Stupid)** approach in order to work best if systems are kept simple rather than made complicated; therefore, simplicity is a key goal in design, and unnecessary complexity will be avoided. The three proposals are the following:

- **Compact Architecture:** S-band and VHF/UHF-band together on the same rotator. X-band on a different rotator.
- **Large Architecture:** S-band, X-band, VHF/UHF-band on three different rotators.
- **Compact Single Feed Architecture:** S-band and X-band on the same rotator. VHF/UHF-band on a different rotator.

In the definition of the system, a trade-off study consists of comparing the characteristics of each system element (figures of merit) for each candidate proposal architecture to determine the best solution that could better balances the choose criteria. For the three proposal the figures of merit are the following:

- **Cost:** In order to satisfy all requirements with a budget of about 30 k€, COTS components are considered to try to find, adapt and acquire items already available on the market while minimizing custom-made designs. This philosophy is a great incentive for the project because using these components could may increase the complexity of the ground station but with a lower cost.
- **Radio Frequency (RF) Performance:** It refers to parameters like full duplex operation, bandwidth, losses, gain, link budget, efficiency, error rate and other specific RF attributes.
- **Tracking Performance:** It refers to parameters like angular resolution, rotating speed, vertical load, breaking and turning torque.
- **Ground Station Performance:** It refers to global parameters like number of satellites with which the station can communicate at the same time, and the quality of the visibility window.

- **Architectural Reliability:** It aim is to minimize the probability of failures and their severity and criticality to achieve high reliability. To achieve this important goal, possible solutions could be fewer components, redundant components (whenever possible), low complexity components, components protection and distributing the capabilities of the architecture to lower criticality of faults (separate rotators for example).
- **Footprint:** In order to install the antennas on a roof, this figure of merit is fundamental for the trade-off analysis.
- **Mass**
- **RF Flexibility:** It refers to the ability of the ground station to operate at various microwave frequencies without sacrificing much performance, and it refers to the capability to move to other frequency while replacing the minimum number of components.
- **Tracking Flexibility:** It refers to the high resolution of rotators to move to higher frequencies, which require high pointing accuracy.
- **Simplicity:** the ability of the design project to remain in the KISS approach.

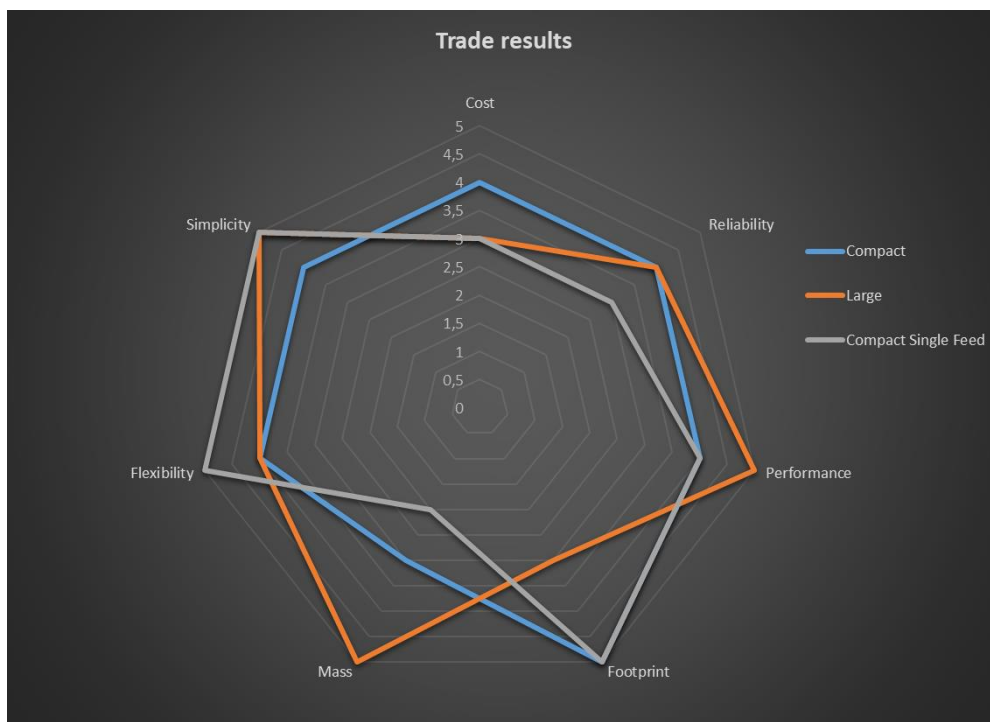


Figure 10: Trade-Off Analysis Results

As seen in Figure 10, the best proposal for the project is the **Compact Architecture** with S-band and VHF/UHF-band on the same rotator and the X-band on a different rotator.

In conclusion the Compact Architecture proposal present the follow characterizes as show in Tables 5-6.

| Cost Budget | |
|------------------|----------|
| X-Line (€) | 14620,94 |
| S-Line (€) | 8014,303 |
| VHF/UHF-Line (€) | 5910,03 |
| | |
| Total (€) | 28545,27 |

| Cost Budget | |
|--------------|----------|
| Support (€) | 3600 |
| RF (€) | 18581,27 |
| Tracking (€) | 6364 |

Table 5: C3 Cost Budget for Line and Cost Budget for Discipline

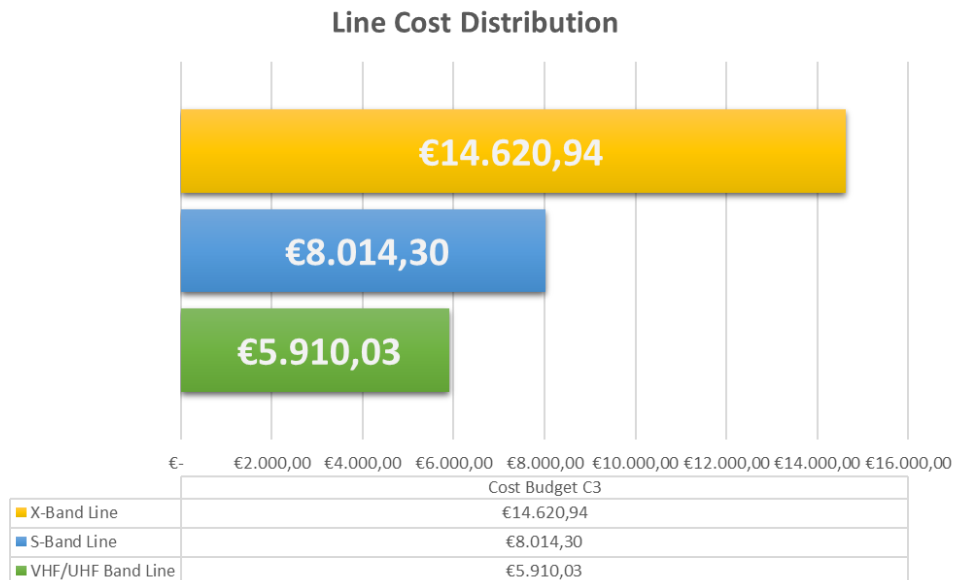


Figure 11: Cost Budget Diagram for Line

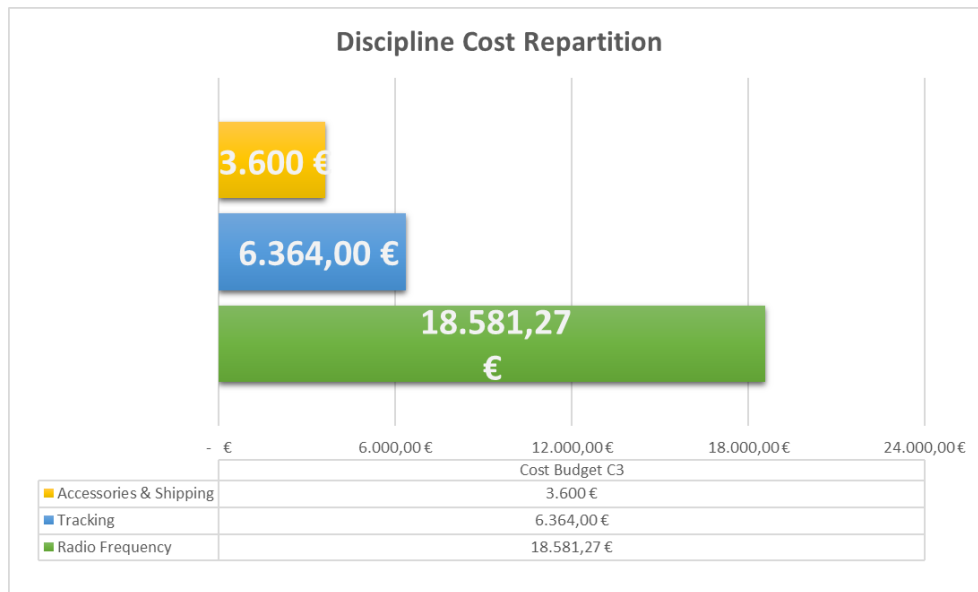


Figure 12: Cost Budget for Discipline

As seen in Table 5 the total cost of the CubeSat Control Centre (C3) is about 29 k€ according to the mission requirement that limit the total cost of the ground segment to 30 k€.

| Mass Budget | |
|----------------------------|-----|
| X-Line (Kg) | 83 |
| VHF/UHF-Line + S-Line (kg) | 124 |
| | |
| Total (Kg) | 207 |

Table 6: C3 Mass Budget

According to Table 6, the total weight of the station (antennas and rotators) is less than 210 Kg. This number is important to respect the security standard for the future installation on a roof.

In conclusion the total consumption of the station in Watt is less than 2 kW. These characteristics are important for the management of the project, but also for the developing of the control centre that has the aim to manage and control the entire station and to communicate with the satellites.

3. THE CONTROL CENTRE DESIGN

The objective of this thesis is to develop a full software to manage telemetries from CubeSats and telecommands from ground, that non-professional operators and students could use without issues. This software will be integrated in the C3's control centre environment and will be the core of the design architecture. In this section the design of control centre functional architecture will be described in detail, as well as the details of CCSDS standards that helped to uniform the software to the European requirements. The mission objectives that led to the actual architecture are shown in Table 7.

| MISSION REQUIREMENTS | |
|----------------------|---|
| ID | Requirement Text |
| C3-MIS--1 | <i>C3 shall support Cubesats mission operations from ground when they are in LEO orbit</i> |
| C3-MIS--2 | <i>C3 shall guarantee the management of the operations for PoliTO/Cubesat Team missions</i> |
| C3-MIS--4 | <i>C3 shall manage mission data from CubeSats payload</i> |
| C3-MIS--7 | <i>C3 shall manage CubeSats housekeeping data</i> |
| C3-MIS--8 | <i>C3 shall manage telecommands to CubeSats</i> |
| C3-MIS--9 | <i>C3 shall guarantee the management of the planning activities on the CubeSats</i> |
| C3-MIS--10 | <i>C3 shall be operated by students and non-professional operators</i> |

Table 7: Control Centre mission requirements

To achieve these requirements, at the beginning, a functional analysis was conducted, and the thesis subsequently focused on the software implementation that will be discussed in Chapter 4.

3.1 CONTROL CENTRE ARCHITECTURE

Figure 13 shows the global architecture of the control centre, in particular the input to processor and all the interface required by the software.

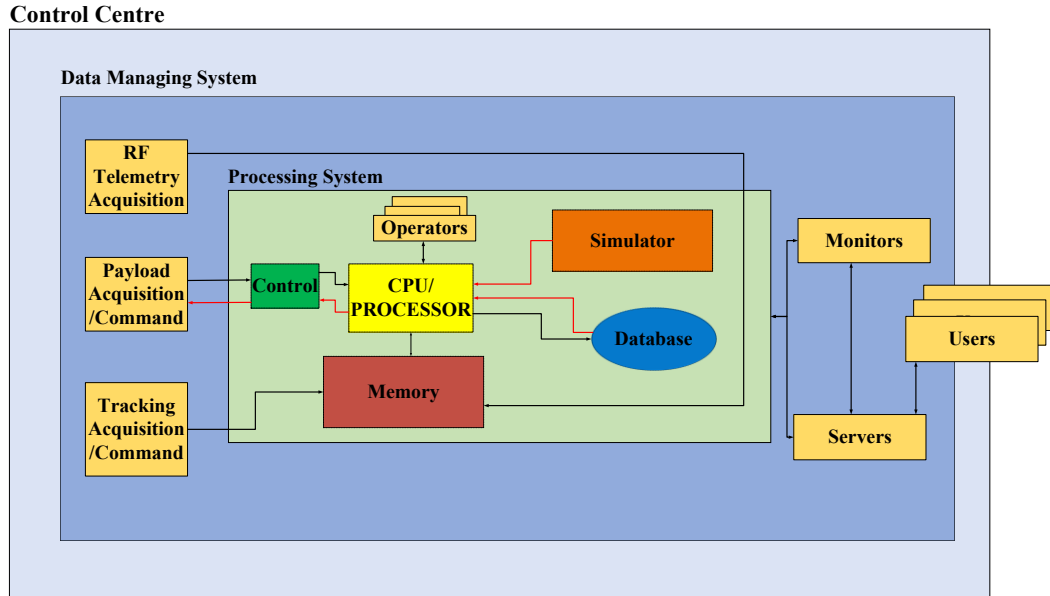


Figure 13: C3 Control Centre Architecture

The first step for the design of the C3 Control Centre is the definition of the inputs (S\c telemetry and ground station subsystems telemetry) and outputs (S\c telecommand and information sharing with mission users). The three main inputs that were identified, shown in Figure 14, in details, are:

- **RF Telemetry Acquisition:** this block concerns all the telemetry from the radiofrequency subsystems like the antenna status, SDR signals and other hardware telemetry. In this environment the control centre could be able to interface itself with the software that manage the RF functions to control them or only to manage them.
- **Payload and S\c Acquisition:** this block concerns all the telemetry (TM) packets sent from the satellite to ground, black line (input), and all the telecommand (TC) packets sent from the ground to the satellite, red line (output). In the input phase, TM packet from the satellite are acquired and checked for a correct acquisition, therefore the useful data, contained within the packets, are extracted, converted and brought to the attention of the operators for the post processing activities. In the output phase, commands from ground

are organized in TC packets and then they are sent to the satellite when it is in visibility of the ground station, or like a schedule organization when the satellite is not in visibility. The satellite acquires the TC packets and executes the command and returns to ground another TM packet as result of the correct execution of the command.

- **Tracking Acquisition:** this block concerns all the telemetry from the tracking subsystems like the antenna rotators, TLE software and other hardware telemetry. In this environment the control centre could be able to interface itself with the software that manage the tracking functions to control them or only to manage them.

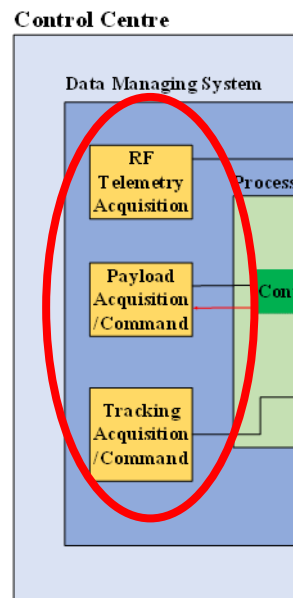


Figure 14: Control Centre Acquisitions

After the acquisitions, the analysis and the processing phase are the major aim of the *Processing Systems* that is a computer composed by different blocks with different functions as shown in Figure 13. In other terms the main objectives of the processing systems are:

- Check the correct acquisition of the TM and TC packets.
- In input, identify the correct useful data (metadata) in the TM packets, and correctly extract that metadata.

- In input, convert the extracted metadata in engineering language and display that information to the operators for the post-processing activities.
- In output, identify the correct command, put them in TC packets and send them to the satellite and monitor the correct execution of the command by the S\c.
- Save and track all actions and activities of the operators.
- Display and share the information with the major mission stakeholders or with the public.

To perform these objectives in Figure 15, the blocks that compose the processing system are shown.

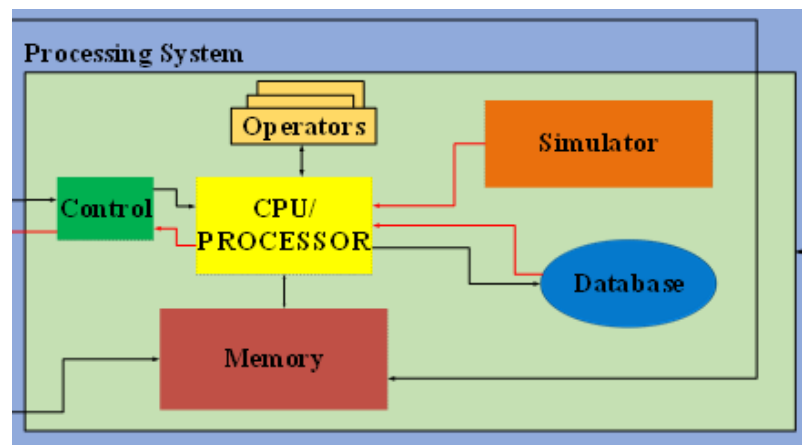


Figure 15: Processing System Architecture

Figure 15 shows in detail:

- **Control Software Block:** The principal aim of this block is to acquire the packets and control them for a correct acquisition. This block is the first step for packet filtering: in this way the correctly acquired packets pass through the data extraction phase, while those that present errors or incorrect acquisition are discarded and requested again by the satellite.
- **CPU/Processor Block:** This block is the core of the control centre. Its major aims are to extract metadata, convert them and perform all the processing and post processing activities of all ground segment.
- **Memory Block:** This block has the function of archiving all the packets and metadata that arriving to the ground station. The

memory block also has the objective of tracking all the activities and actions performed by operators.

- **Database Block:** This block helps the processor unit to extract metadata from TM packets and to convert them into engineering language. In the control centre software, the database is composed by roots and dictionaries in which are expressed the TM/TC packets structures for packets construction and packets extraction, conversion methods, packet rows and packets check loops to control the correct acquisition of the packets, the correct extraction of the metadata, the correct conversion and the control of the conversion value.
- **Simulator Block:** This block is fundamental in the command construction phase. This environment can simulate the execution of the command and to display the correct result of this execution. It is able to simulate the spacecraft OBC, housekeeping and science telemetry and merge them to generate a realistic simulated data stream. The command packet is sent in input to the simulation environment and it is tested for the correct execution and to be secure that the command create by the operators is correct. Once the TC packet passed the simulation, it is ready to be sent to the real S/c.
- **Operators:** This block represents the operators working on the platform interface. It is important to say that the station is managed by non-professional operators and by students, so a training period for the operators is mandatory in the design process.

In conclusion, in Figure 16, the last part of the control centre architecture is shown. In this part the following block are considered:

- **Monitors Block:** This block has the function of displaying telemetries and telecommands to the operators through an interface and monitors.
- **Server Block:** This block represents the archive where the operators can save all the information about the mission (telemetries, images).

This is an open archive where the major mission stakeholders can take the access and use the information saved.

- **Users Block:** This block represents the major mission stakeholders that can require to the operators or to the servers of the ground segment information about a specific mission.

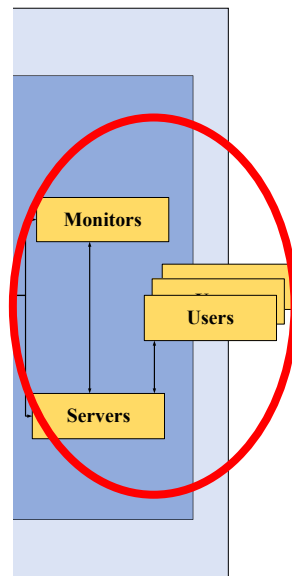


Figure 16: Control Centre sharing blocks

3.2 CONTROL CENTRE CCSDS STANDARDS OVERVIEW

The **Consultative Committee for Space Data Systems** (CCSDS) is an organisation officially established by the management of member space agencies. The committee meets periodically to address data systems problems that are common to all participants, and to formulate some technical solutions to these problems. Insofar as participation in the CCSDS is completely voluntary, the result of Committee actions are termed *recommendations* and are not considered binding on any Agency [5].

At the start of this master thesis there was a period of information gathering. This entailed a prolonged study of recommendation documents from CCSDS and ECSS. These documents covered information about how the TM packets, TC packets and image packets (IMG) are to be structured, about how to encode and decode the communications to ensure error-free transmission and how to determine when data has been lost. From all standards and recommendations, a rough idea of a structure of the packets, transfer protocol and their implementation could be formed. In Figure 17 a description of all the communication protocol according to the CCSDS standards is shown.

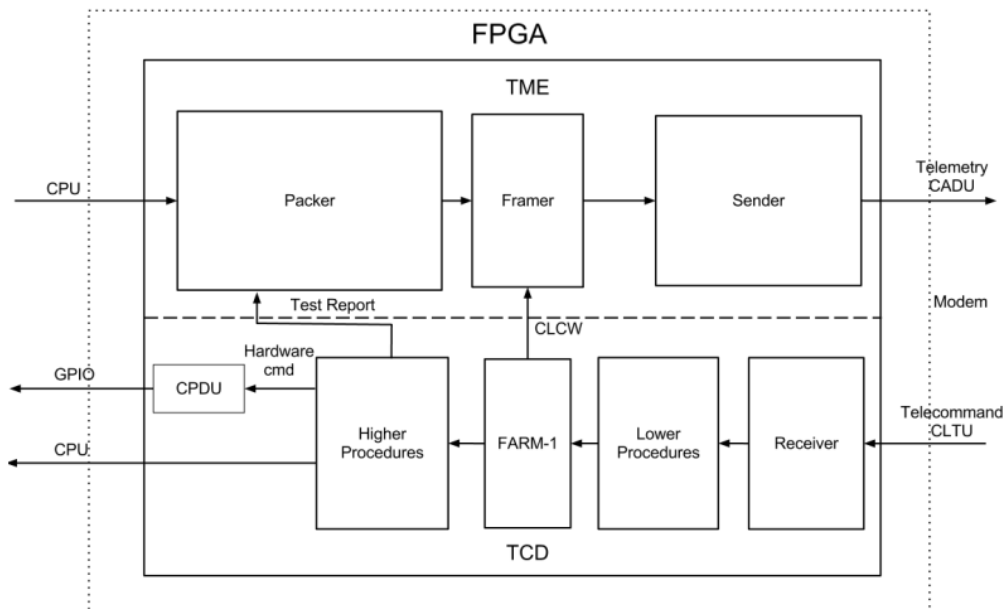


Figure 17: Overview of the Communication Protocol Core

In Figure 17, it is possible to see the different modules, depicted as squares, which make up the core of communications. Each module fills a specific function

in the communication with the ground station. The top row encodes the telemetry and the bottom row decodes telecommands. In the next section, it is described in detail how telemetry branch, telecommand branch and image branch are structured and how they are encoded and decoded.

3.2.1 CCSDS OVERVIEW: TELEMETRY CONSTRUCTION

According to [5], TM is constructed as shown in Figure 18 with the Space Packet placed inside an Advanced Orbiting System Transfer Frame (AOSTF) which is in turn inside a Channel Access Data Unit (CADU). The telemetry construction process is from the inside out.

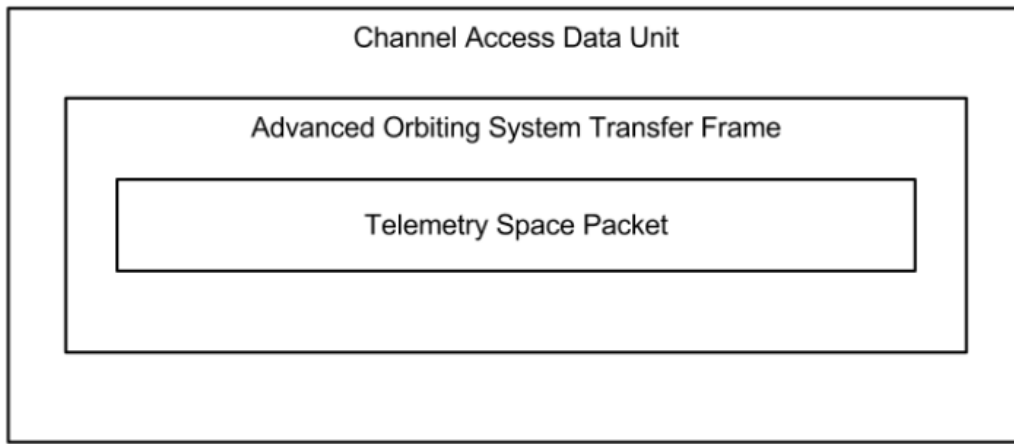


Figure 18: TM Structure

In specific terms, Figure 19 describes data structures used to transfer metadata from on board systems to ground systems.



Figure 19: CCSDS TM Packet Data System

The aim of the packet telemetry concept is to permit multiple application process running in on-board sources to create units of data as best suits each data source, and then to allow the on-board systems to transmit these data over a space-to-ground communication channel in a way that enables the ground systems to receive the data with efficiency and reliability and provide them to the operators. To achieve these functions, the CCSDS Recommendation defines

different structure such as: **Source Packets**, and **Source Packets** from various **Application Processes** (APIDs).

Source Packet (PKT), which is also termed *packet*, is a data structure generated by an on-board APID in a way that is responsive to the needs of that process. It could be generated at fixed or variable intervals and may be fixed or variable length. The packet structure is composed by a header that identifies the source and the characteristics of the incoming packet and identifies the APID that controls the internal data content of the packet. Each packet is defined by a fixed frame called header, at the beginning of the packet, and a tail, at the end of the packet. The useful field of the packet contains the telemetries generated on-board and it is characterized by parameters of variable length.

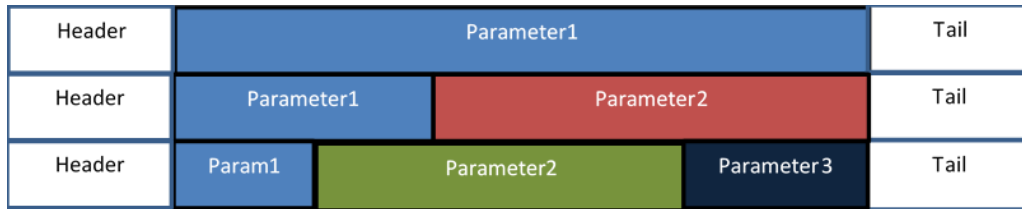


Figure 20: Global Packet Structure

The Packet will encapsulate all the information and the data application which are to be transmitted from a specific APID in space to one or several channels on the ground. As aforementioned, the source packet will consist in two major fields, positioned contiguously, in the following sequence: **PKT Primary Header** (mandatory) and **PKT Data Field** (mandatory).

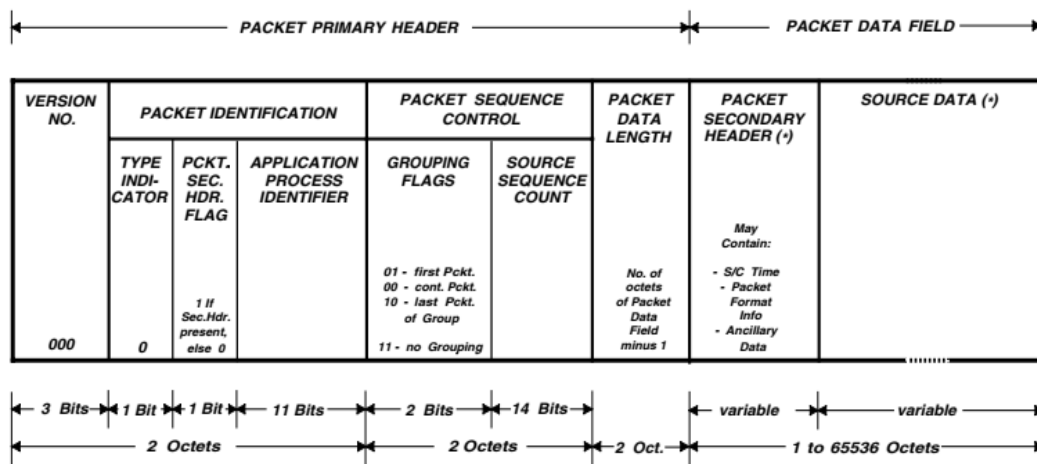


Figure 21: Source Packet Format [5]

As shown in Figure 21, the packet will consist of a least 7 and at most 65542 octets.

The **PKT Primary Header** is mandatory and will consist of the four fixed fields positioned contiguously, in the following order:

- **Version Number** (3 bits length)
- **Packet Identification** (13 bits)
- **Packet Sequence Control** (16 bits)
- **Packet Data Length** (16 bits)

Each field contains a different information.

- **VERSION NUMBER:** will be contained within the bits 0-2 of the PKT Primary Header, and will identify the data unit as a source packet and shall be set to “000”
- **PKT IDENTIFICATION FIELD:** will be contained within the bits 3-15 of the PKT primary Header. It is divided into three sub-fields:
 - **TYPE INDICATOR** (1 bit): it will identify the type of packet. Because the CCSDS TC packet uses a similar structure, the type indicator distinguishes between telemetry and telecommand data units. For TM packet will be set to “0”, instead for TC packet will be set to “1”.
 - **PACKET SECONDARY HEADER FLAG** (1 bit): it will indicate the presence or the absence of the PKT Secondary Header within this packet. It will be set to “1” if a PKT Secondary Header is present, it will be set to “0” if a PKT Secondary Header is not present. This flag will be static with respect to the APID throughout a mission phase.
 - **APID** (11 bits): it will be different for different application processes on the same transmission channel. The Application Process defines the context of the data field and control all the useful data of the packet for the correct construction on-board and the correct extraction on ground.

- **PKT SEQUENCE CONTROL FIELD:** it will be contained within bits 16-31 of the PKT Primary Header. It is divided into two sub-groups as follows:
 - **GROUPING FLAGS** (2 bits): expresses the segmentation of the packet groups. It will be set to “01” for the first packet of the group, to “00” for a continuing packet of the group and to “10” for a last packet of the group. If there is no segmentation, it will be set to “11”. All packets belonging to a specific group of packets will be identified by a unique APID.
 - **SOURCE SEQUENCE COUNT** (14 bits): it will provide the sequential binary count of each packet generated by an APID. The purpose of this field is to order a specific packet with other packets generated by the same APID, even though their natural order may have been disturbed during the transmission to the operators on the ground. This field is normally associated to a Time Code [6] (its insertion is, however, not mandatory) to provide unambiguous ordering.
- **PKT DATA LENGTH FIELD:** it will be contained within bits 32-47 of the PKT Primary Header. This 16 bit field will contain a binary number equal to the number of the octets in the PKT Data Field minus 1. The value contained in this field may be variable and it is in the range of 0 to 65535, corresponding to 1 to 65536 octets.

The **PKT Data Field** follows, without gap, the PKT Primary Header. This field is mandatory, and it is divided in two field with a variable length, positioned contiguously, as follows:

- **PKT SECONDARY HEADER:** follows, without gap, the PKT DATA LENGTH FIELD and it is mandatory if there is not Data Field, otherwise it is optional. In any case the presence or the absence of the PKT Secondary Header will be signalled by the PKT SECONDARY HEADER FLAG. If present, the **PKT SECONDARY HEADER DATA FIELD**, consists of an integral

number of octets. This field contains the CCSDS time codes formats defined in [6]. In this field is defined the time of packet construction and transmission according to the CCSDS recommendation. The time code defined in [6] generally consists of an optional P-Field (Preamble Field) which identifies the time code choice and its characteristics like period, epoch, length and resolution, and a mandatory T-Field (Time Field). The time code selected must be static for a given APID throughout all mission phases. All the field associated to the PKT Secondary Header depending on what time code is selected for the packet construction.

- **SOURCE DATA FIELD:** If this field is present, it will follow, without gap, the PKT Secondary Header. This field is mandatory in the case of absence of PKT Secondary Header, otherwise it is optional. The field contains the source data (metadata) from a specific APID and the length of this field may be variable: it will contain an integral number of octets.

All the fields described are fundamental for a correct construction of the packets. When the packets are constructed, they are ready to be sent to the ground through a space-to-ground channel. This channel allows to transfer these packets to the ground and, in addition this aim, consents to control and check the correct construction (and on the ground the correct extraction) of the packets using several consistency checks like CRC (Cyclic Redundancy Check) loops or other consistency checks to monitor the correct acquisition of the bits in space and on the ground. It is important to say that, on the ground, the extraction process uses the same packet structures to make simple the metadata research, the metadata extraction and the metadata conversion.

3.2.2 CCSDS OVERVIEW: TELECOMMAND CONSTRUCTION

According to [7]-[9], TC construction is shown in Figure 22 with the Space Packet placed inside a **Telecommand Transfer Frame** (TCTF) which is in turn inside a **Communication Link Transmission Unit** (CLTU). Telecommands are decoded from the outside in.

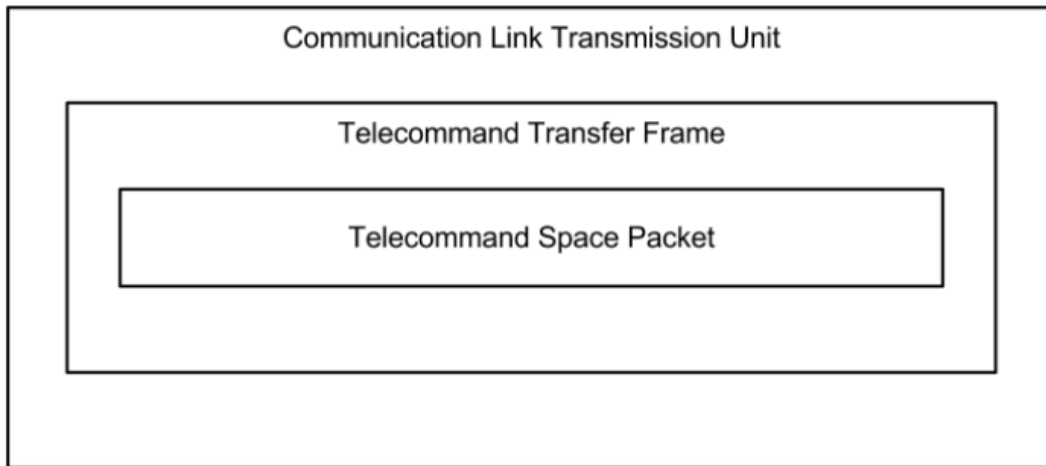


Figure 22: TC Structure

The Space Packet is the common standard for the structure of packets that are sent as telemetry or telecommand. The structure for TM and TC packets is almost identical, but there are some differences. The structure of the TC packets is shown in Figure 23.

| Packet Header (48 Bits) | | | | | | | Packet Data Field (Variable) | | | |
|-------------------------|-----------|------------------------|------------------------|-------------------------|----------------|---------------|---|------------------|----------|-----------------------------------|
| Packet ID | | | | Packet Sequence Control | | Packet Length | Data Field Header (Optional) (see Note 1) | Application Data | Spare | Packet Error Control (see Note 2) |
| Version Number (=0) | Type (=1) | Data Field Header Flag | Application Process ID | Sequence Flags | Sequence Count | | | | | |
| 3 | 1 | 1 | 11 | 2 | 14 | | | | | |
| 16 | | | | 16 | | 16 | Variable | Variable | Variable | 16 |

Figure 23: TC Packet Format [8]

The TC decoding process can be split into three main parts shown in Figure 24. The first part is the channel coding and synchronization (the receiver) and the second part is the TC data link protocol which in-turn can be split into the lower procedures. Each part is coded and tested separately before being integrated into one cohesive piece of code.

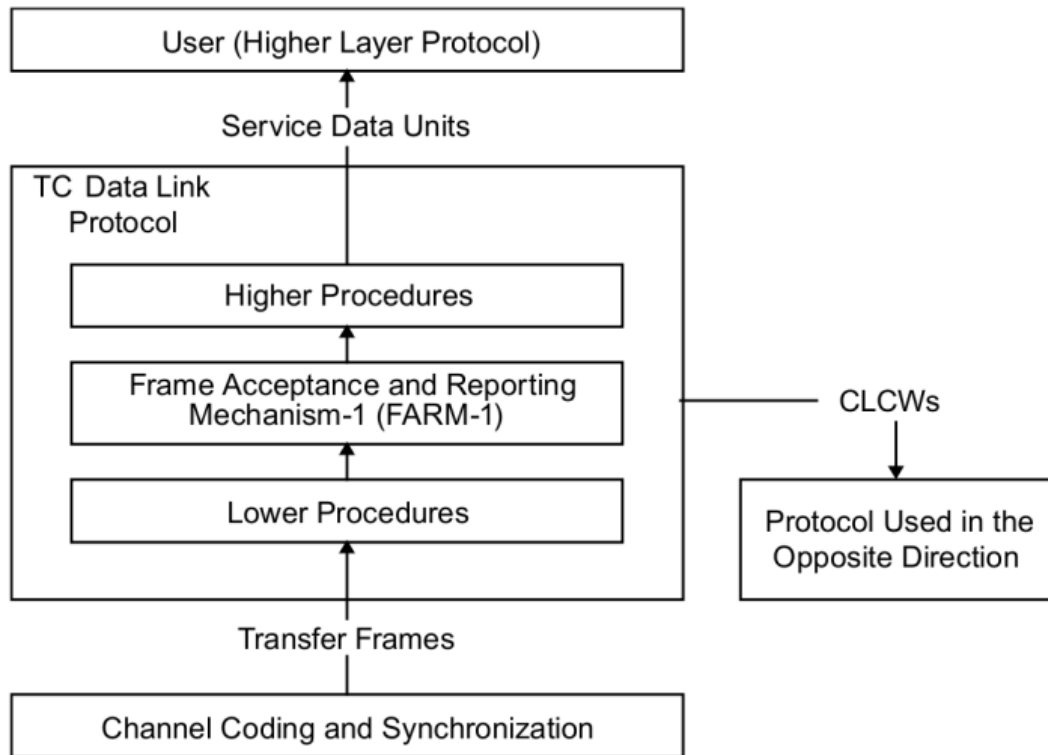


Figure 24: Structure Overview of the Decoding Process

The standard data structures are the Acquisition Sequence, CLTU, and the Idle Sequence. They are used to provide synchronization of the symbol stream and are described below.

- **Acquisition Sequence:** it is a data structure forming an introduction which provides for initial symbol synchronization within the incoming stream of detected symbols. The length of the Acquisition Sequence will be selected according to the mission telecommand link performance requirements, but the preferred minimum length is 16 octets. The length is not necessary to be an integral multiple of octets. The pattern will be alternating “ones” and “zeros”, starting with either a “one” or a “zero”.
- **CLTU:** it contains the data symbol that are to be transmitted to the S/c. Each code block within the CLTU provides at least 2 data transitions. The CLTU as delivered to the physical layer must have a random component to guarantee sufficiently frequent transitions for adequate symbol synchronization [7].

modules i.e. the discrete wavelet transform (DWT) and the bit plane encoder (BPE) modules. It can perform both lossy and lossless compression and has very low complexity so that it can be implemented with minimum power and processing resources requirements.

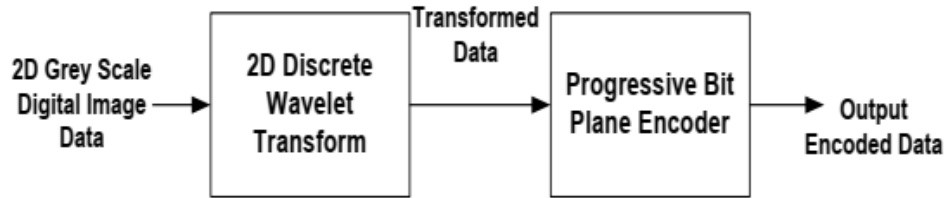


Figure 26: Functional Block Diagram [10]

As aforementioned, image compression system consists of the following two main blocks:

- **Encoder:** Figure 27 shows the basic building blocks of a source encoder. Mapper module maps the input image pixels performing the de-correlation using transform. The quantize block limits the accuracy of the mapper output values. This is the step where major compression takes place.
- **Decoder:** The decoder performs the reverse function of that of the encoder. However, quantization is generally irreversible hence the quantization block is excluded from the decoder as shown in Figure 27.

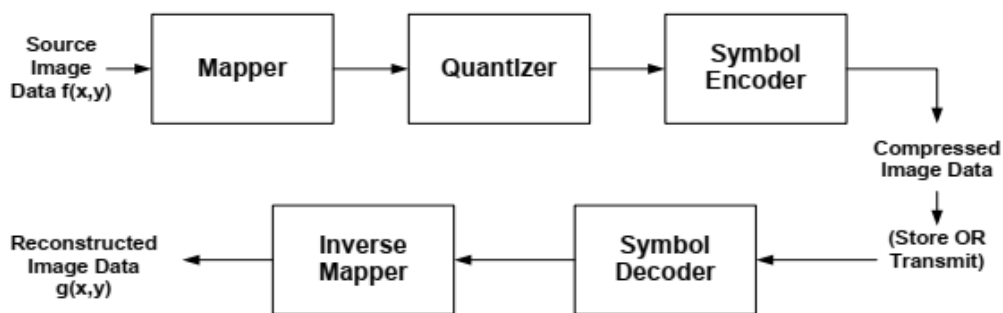


Figure 27: Image Compression Model: Encoder and Decoder

According to the CCSDS standards, the algorithm works on the two dimensional digital image data i.e. grey scale images from panchromatic single channel image sensors in space imaging systems. The de-correlation module consists of discrete wavelet transform which is followed by the progressive BPE

(Bit Plane Encoder) module. The BPE produces encoded output bit-stream in the form of a single segment or a series of segments. Each segment has a segment header which is followed by the encoded data.

All the processing steps are based on the DCT (Discrete Cosine Transform). Source image samples are grouped into 8x8 blocks, shifted from unsigned integers to signed integers and input to the DCT. The following equation is the idealized mathematical definition of the 8x8 DCT:

$$F(u, v) = \frac{1}{4} C(u) C(v) \left[\sum_{x=0}^7 \sum_{y=0}^7 f(x, y) * \cos \frac{(2x+1)u\pi}{16} \cos \frac{(2y+1)v\pi}{16} \right]$$

The DCT takes such a signal as its input and decomposes of the 64 unique two-dimensional “spatial frequencies” which comprise the input signal’s “spectrum”. The output of the DCT is the set of 64 basis-signal amplitudes (DCT coefficients) whose values can be regarded as the relative amount of the 2D spatial frequencies contained in the 64-pint input signal. The DCT coefficients are divided into “DC coefficient” and “AC coefficients”. DC coefficient is the coefficient with zero frequency in both dimensions, and AC coefficients are remaining 63 coefficients with non-zero frequencies. The DCT step can concentrate most of the signal in the lower spatial frequencies. In other words, most of the spatial frequencies have zero or near-zero amplitude and need not be encoded.

The BPE encodes the bit planes starting from most significant bit plane to the least significant bit plane.

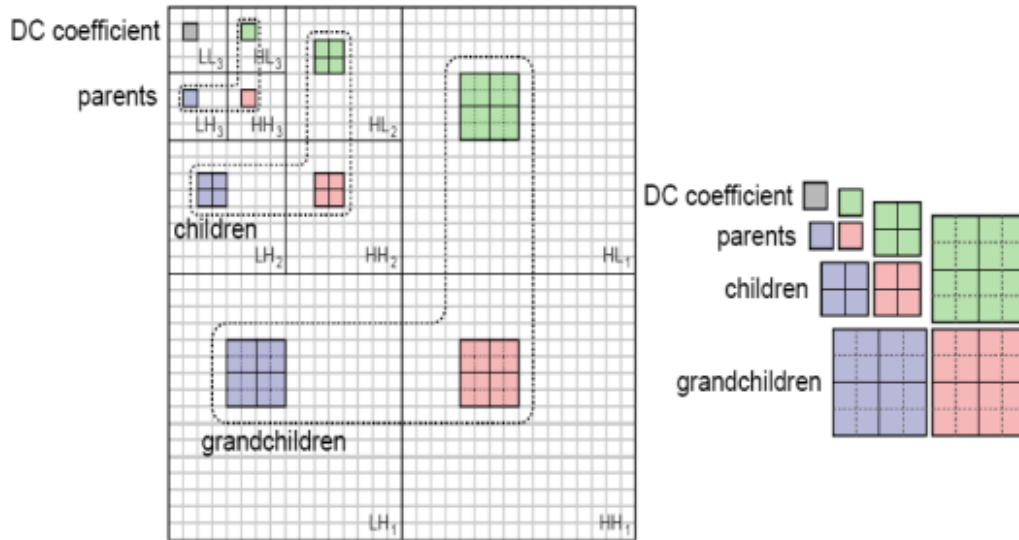


Figure 28: Block and Group structure of DWT transformed data

Within a single coded segment (or image packet), the segment header is coded first. After the quantized DC coefficients are coded, the AC coefficient bit depths are implemented and then the bit planes of the DWT coefficient block are coded as shown in Figure 29.

| Segment Header |
|---|
| Initial coding of DC coefficients |
| AC coefficients bit depths |
| Encoded bit plane $b = \text{bitDepthAC} - 1$ |
| Encoded bit plane $b = \text{bitDepthAC} - 2$ |
| |
| Encoded bit plane $b = 0$ |

Figure 29: Structure of an Image Packet

In details, there are:

- **Segment Header:** it consists of the follow four parts:
 - **Part I:** (3 or 4 bytes - Compulsory)
 - **Part II:** (5 bytes – Optional)
 - **Part III:** (3 bytes – Optional)
 - **Part IV:** (8 bytes – Optional)
- **Initial Coding of DC coefficient:** it is performed into two stages:
 - Coding quantized DC coefficients
 - Coding additional bit planes of DC coefficients

- Stages of Bit Plane Encoding:** each bit plane is encoded in multiple stages from 0 to 4 as shown in Figure 30. Stage 0 coding for each block is the most significant bit of each DC coefficient. Stage 1 encodes the bit planes containing magnitudes of parent coefficients in a segment. Stage 2 encodes children coefficients and Stage 3 encodes bit planes containing magnitudes of grand-children coefficients in a segment. Stage 4, in conclusion, encodes the remaining bits of each AC coefficient.

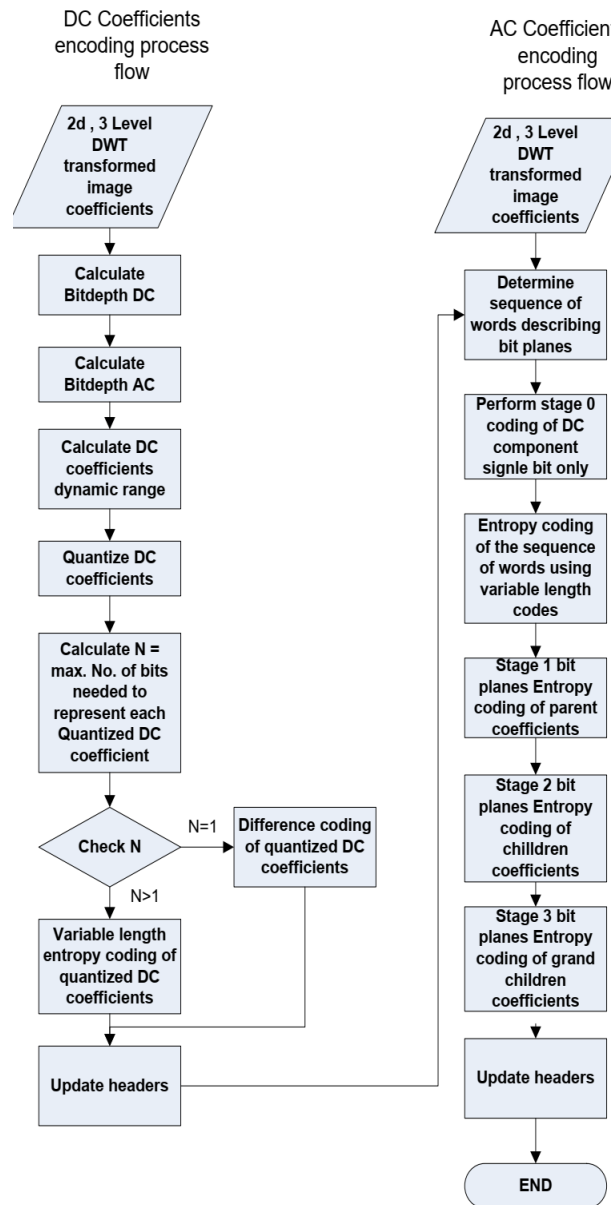


Figure 30: Bit Plane Encoder [10]

3.2.4 CCSDS OVERVIEW: MISSION PLANNING & SCHEDULING

Mission planning and Scheduling are integral parts of Mission Operations and closely related to the other aspects of the overall monitoring and control space missions. In some space mission, in particular for CubeSat missions, the planning may be centralized in a single function. The distribution of functions over different entities depends by a number of factors such as the availability of facilities with unique capabilities, the existence of groups of experts with specific knowledge and availability of planning experts [15].

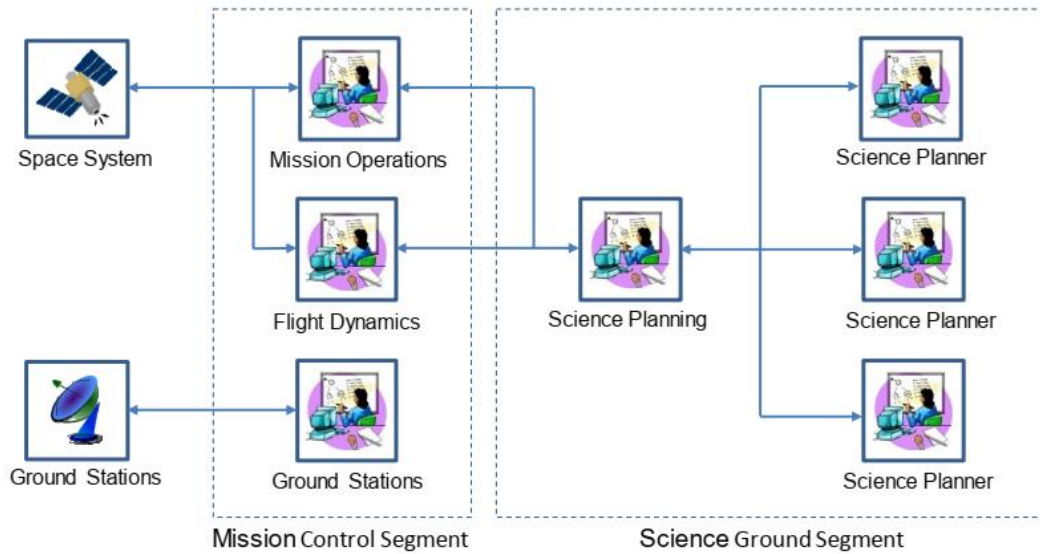


Figure 31: Example of Federated Planning for a Science Mission [15]

According to the recommendation plan a space mission requires the collaboration of different elements that have a flow of information at different levels. For example, the output of a planning function could be the input for a new planning function at the same level or in a lower level as show in Figure 32.

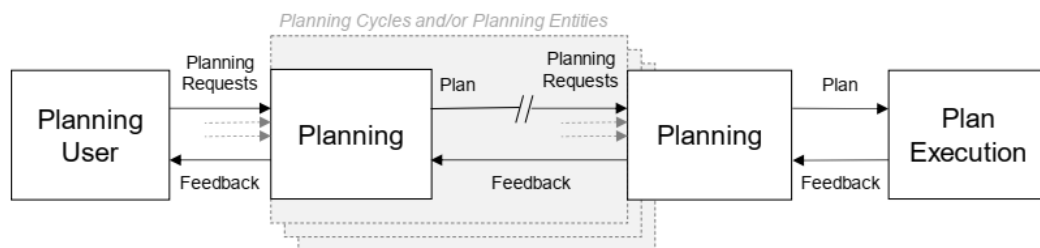


Figure 32: Planning Information Flow

As shown in Figure 32, a Planning flow has the following steps:

- **Planning User:** a generic function that is responsible for submitting request to the planning function and control. It also receives feedbacks on the status of Planning Requests, the generated plans, and the status of the planning process. It is not a Planning function itself, but it is a user of Planning data and service. In the specific case of SAT SIM software, the user plan is the operator that generate the schedule.
- **Planning:** this is the function responsible for performing Mission Planning. The output of the Planning function is the plan that is retrieved by the Planning Users and distributed to Plan Execution functions.
- **Plan Execution:** this is the function responsible for executing a Plan (or part of one). It is possible to have multiple plan execution functions distributed between space and ground segments. It is not a Planning function itself, but it does support a common model of the plan in its interface with Planning [15].

In conclusion, the generation and the execution of a mission plan is a monitored process from the start to the end of its protocols and it provide an important and autonomous way for the execution of all the phases of a mission.

4. SOFTWARE PHILOSOPHY & ARCHITECTURE

The thesis follows the Python Multithreading Approach. In this section the overview of the code philosophy and the software architecture will be explained in detail.

Python is a computer programming language designed for readability and functionality. One of Python's design goals is that the aims of the code are easily understood because of the very clear syntax of the language. The Python language has a specific form (syntax) and semantics which are able to express computations and data manipulations which can be performed by a computer. Python's implementation was started in 1989 by Guido van Rossum at CWI (Centrum Wiskunde & Informatica, research institute in the Netherlands) as an update and a successor to the ABC programming language.

Python is an interpreted language, meaning a programming language whose programs are not directly executed by the host CPU but rather executed (or as said "*interpreted*") by a program known as an interpreter. The source code of a Python program is partially compiled to a bytecode form of a Python "*process virtual machine*" language. This is one of the major distinctions with the C codes which are compiled to CPU-machine code before the run-time.

Another characteristic of Python is that it is "*dynamically typed*", that means that most of its type checking is performed at run-time as opposed to at compile-time. Other dynamically typed languages are JavaScript, Ruby and Objective-C.

The data which a Python program deals must be described precisely. The description of variables is referred to as the data type. In the case of Python, the fact that it is dynamically typed means that the interpreter will figure out what type a variable is at run-time, so the programmer doesn't have to declare variable types himself. Python is "*strongly typed*", meaning that it will raise a run-time type error when the programmer has violated the Python syntax rule as to how types can be used together in a statement. Of course, all these facts do not mean that the programmer can be neglecting and hoping Python to figure out things.

```
>>> a = 2012
>>> type(a)
<type 'int'>
```

Figure 33: Example of how Python can figure out the type at run-time

4.1 PYTHON MULTITHREADING APPROACH

To better explain the processes occurring in the master thesis architecture, it is important to have an overview about the Python's concurrency, multiprocessing and multithreading philosophy.

The concurrency, in Python, is the occurrence of two or more events at the same time. In terms of programming language, concurrency is the overlapping of two tasks in execution. With concurrent programming, the performance of software systems can be improved because it can concurrently deal with the requests rather than waiting for a previous one to be completed.

Thread is a small unit of execution that can be performed in an operating system. It is not itself a program but runs within a program; it means that threads are not independent from one other. Each thread shares code section, data section, etc. with other threads. A thread has the following components:

- Program counter which consist of the address of the next executable instruction.
- Stack.
- Set of registers.
- Unique ID thread.

Instead, multithreading is the ability of the CPU to manage and control the use of operating systems by executing multiple threads concurrently. The main concept of the multithreading philosophy is to achieve parallelism by dividing a process into multiple threads [11].

A process is defined as an entity, which represents the basic unit of the code implemented in the system. In other words, the programmer writes his program and when he executes it, it becomes a process that performs all the tasks in the program. During the process execution, the code passes through the stages shown in Figure 34.

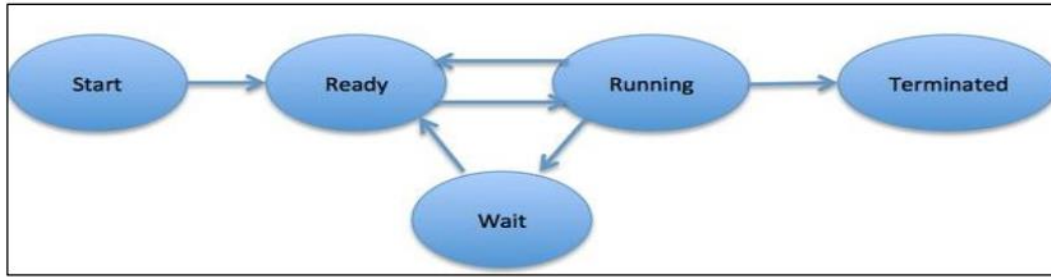


Figure 34: Different stages of a process [11]

A process can only have a thread (primary thread) or multiple threads where each of them have their own set of registers, program counter and stack as shown in Figure 35.

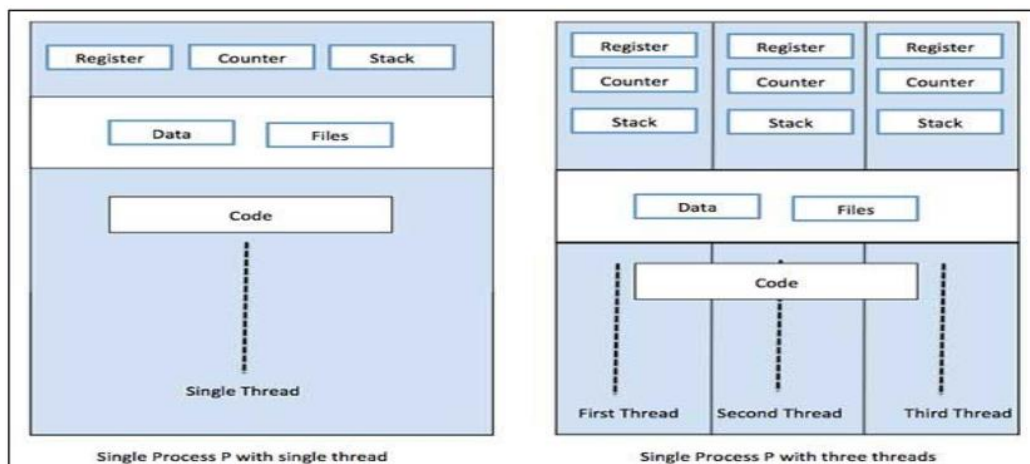


Figure 35: Comparison between process with 1 thread (left) and process with multiple thread (right) [11]

Typically, a thread can exist in five different states:

- **New Thread:** a new thread begins its life cycle in the new state. At this stage, it has not yet started, and it has not been allocated any resources (it is only an instance of an object).
- **Runnable:** the thread is started, and it is waiting to run. At this time, the thread has all the resources but still task scheduler has not scheduled it to run.
- **Running:** the thread makes progress and executes the task, which are running in the task scheduler. At this moment, the thread can go to either the dead state or the non-runnable/waiting state.
- **Non-running/waiting:** the thread is paused because it is waiting for the response of some I/O request or waiting for the completion of the execution of other thread.

- **Dead:** the thread enters the terminated state when it completes its task (the thread is terminated).

The complete thread life cycle is shown in Figure 36.

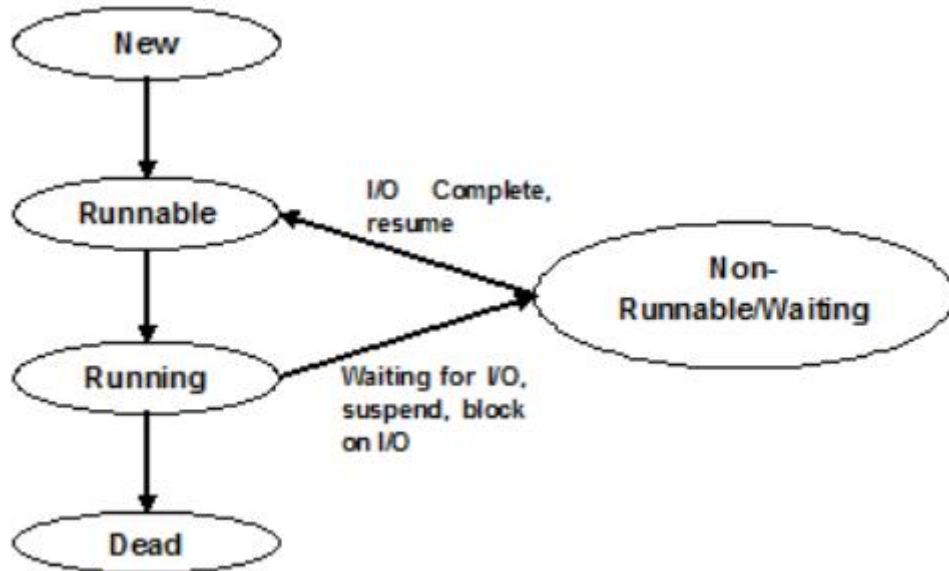


Figure 36: Thread complete life cycle

In conclusion, in table 8 an overview of advantages and disadvantage of using threads is shown.

| Advantages | Disadvantage |
|---|---|
| Thread library contains code for creating and destroying threads, for passing message and data between threads, for scheduling thread execution and for saving and restoring thread contexts. | In a typical operating system, most system call are blocking |
| Thread can run on any operating system | Multithreading application cannot take advantage of multiprocessing |
| Scheduling can be application specific in the thread | |
| Threads are fast to create and manage | |

Table 8: Advantages and Disadvantages of using threads

4.2 SAT SIM OVERVIEW: SOFTWARE ARCHITECTURE

Figure 37 shows the global architecture of the SAT SIM library.

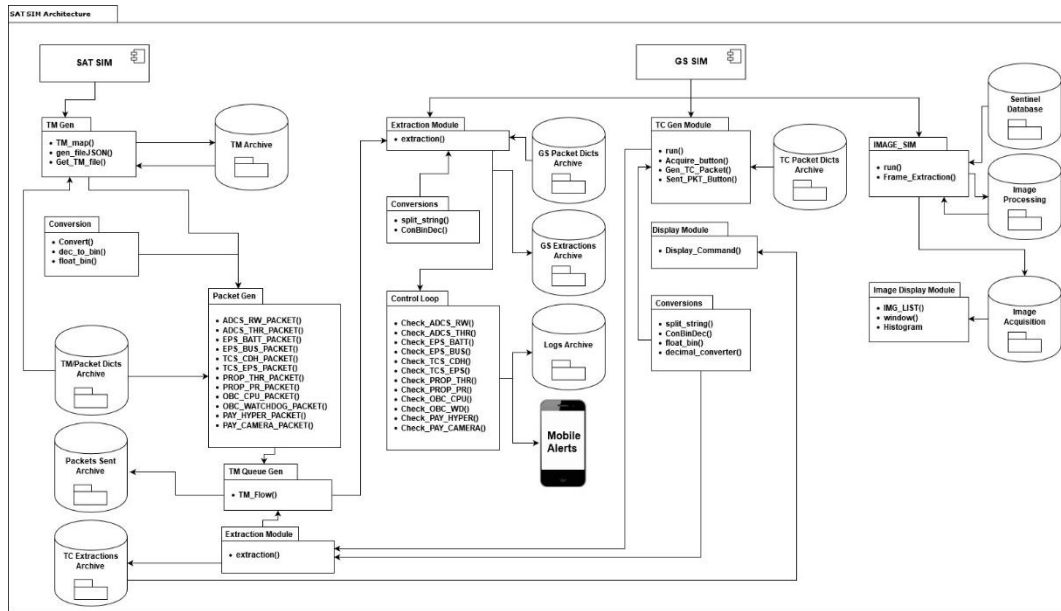


Figure 37: SAT SIM Architecture

In this architecture the main software blocks are shown, and for each block all the methods selected to achieve the blocks' functions are highlighted.

From the global overview it is possible to see two main threads SAT SIM and GS SIM.

- **SAT SIM Thread:** this thread (*Satellite Simulator*) has the main function of generating telemetries, pack them and send the CCSDS standard packets to the GS SIM thread. The SAT SIM thread has also the aim of receiving telecommands from the GS SIM thread, recognizing them and executing the telecommand and, in conclusion, sending new telemetry as proof of the correct execution of sent telecommand.
- **GS SIM Thread:** this thread (*Ground Station Simulator*) has the main function of receiving telemetries, recognizing them and extracting the useful data from packets. The GS SIM has also the aim of generating telecommand packets and sending them to the SAT SIM thread. In the GS SIM there is also an additional branch to manage the images from the satellite. The main function of this

branch is to receive the images, recognize them and to display images to the operators.

In the next section, the details of the branches will be examined to better explain all the methods that compose the software architecture of the master thesis work.

It is important to say that all the architecture uses different databases to store data and to read the structure of the packets to build them.

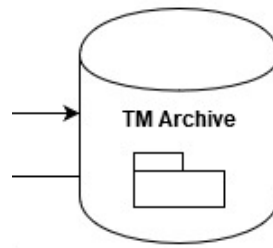


Figure 38: Example of SAT SIM Databases

4.3 SAT SIM OVERVIEW: TELEMETRY BRANCH

In the previous section it was explained the main block of the SAT SIM architecture. In this section, the telemetry branch will be analyzed. The objectives that led to the construction of this branch are the following:

- To simulate a CubeSat architecture to generate realistic telemetries.
- To generate packets with the CCSDS standards.
- To generate packet strings to send to the GS SIM.
- To take track of all the action done during the generation of telemetries.

In Figure 39 the TM branch is shown with the detail of all the methods and database that led to the CCSDS packets generation.

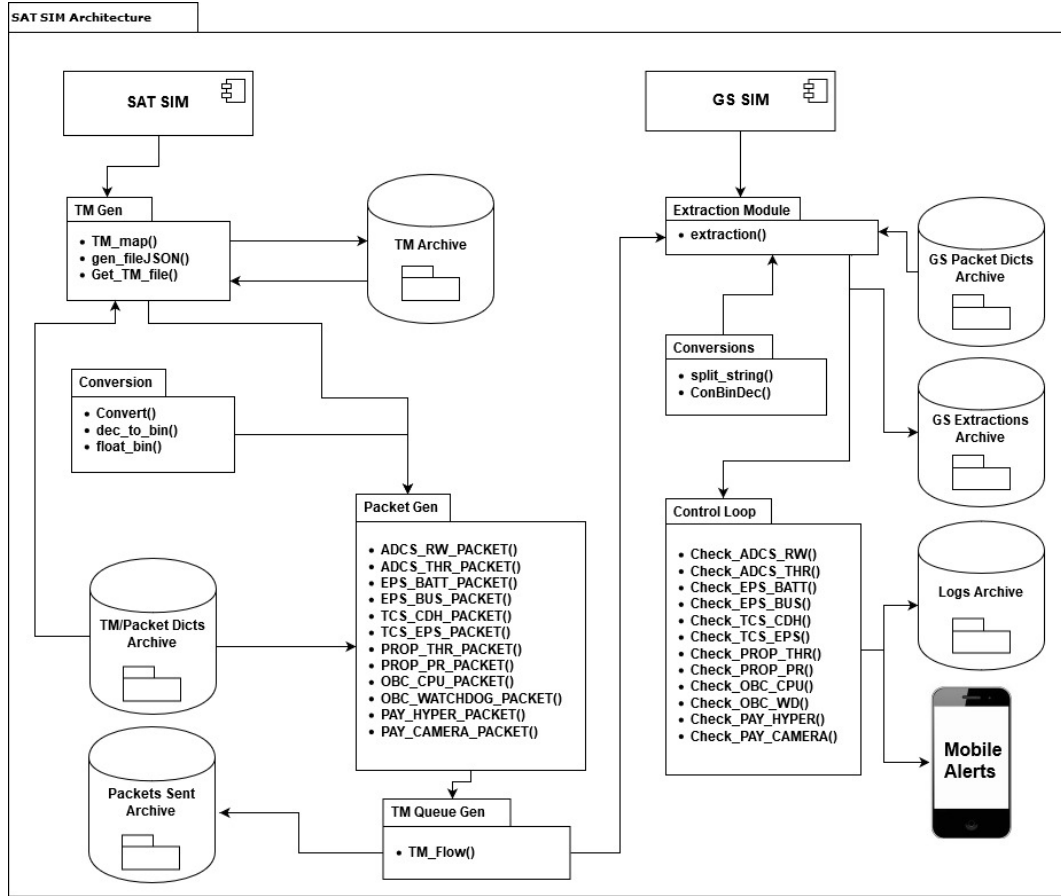


Figure 39: SAT SIM - TM Branch

The generation of telemetry is dedicated to the Python class *SAT SIM*. This thread is organized into different module with different functions (methods). The **TM Gen** module has the purpose to simulate the CubeSat architecture and to generate realistic telemetries.

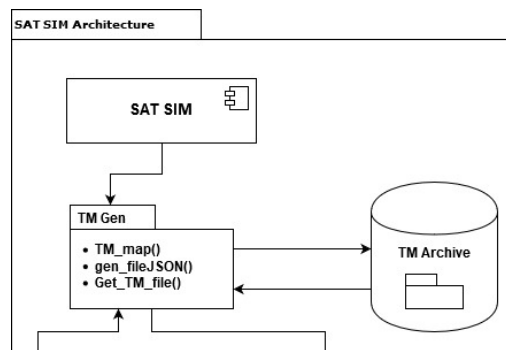


Figure 40: TM Gen Module

In TM Gen has different methods, as follows:

- **TM_map ():** is the method that generates the telemetry when it is called. The telemetry is generated in form of dictionary and the

gen_fileJSON() method saves it in the *TM Archive*. In the archive, there are the telemetries generated saved in form of “Sat_Telemetry_date and time of generation.json” (i.e. “*Sat_Telemetry_2019_06_01-22_33_18.json*”). The structure of the TM dictionary is taken from the *TM/Packet Archive*, where the structure of dictionaries that the thread needs are saved.

- **gen_fileJSON ()**: this method saves the generated telemetries in the *TM Archive* in form of JSON file when it is called. These files are dictionaries that, when they are called in the software, make it easy to unpack data and search different fields faster.
- **Get_TM_file ()**: this method takes the saved files from the *TM Archive* and read them to generate new packets. The TM dictionary structure is created as object in the code, and, when a TM profile is generated, this structure is filled and saved as json file in the archive.

The output of this module is the TM dictionary where there are the useful data to packetize according to the CCSDS standards. This dictionary is the input for the **Packet Gen** module.

The **Packet Gen** module has the aim to use the telemetries data, divide them for the different subsystems, and to generate the CCSDS packet.

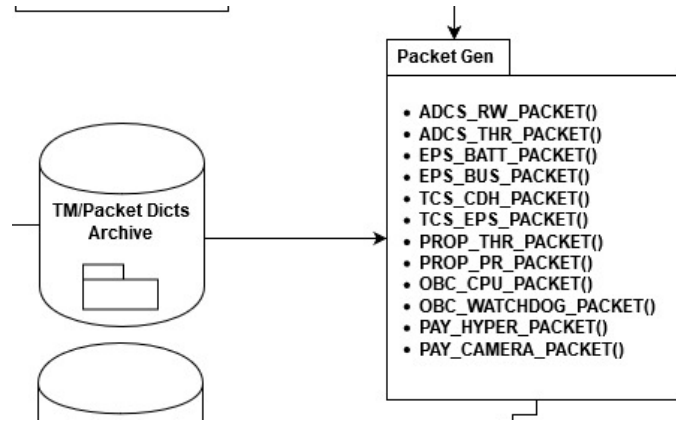


Figure 41: Packet Gen Module

The methods in this module according to the specific on-board system, take the useful data and generate the packets according to the standards specified in the section 3.2.1. To build the packets this module uses the packet structures saved in the *TM/Packet Dicts Archive* where there are all the dictionary structures, divided by sub-systems, to fill to build a packet. The output of this module is, for each

system, a binary string that represents all the packet fields. These strings are the input to the **TM Queue Gen** module as shown in Figure 42.

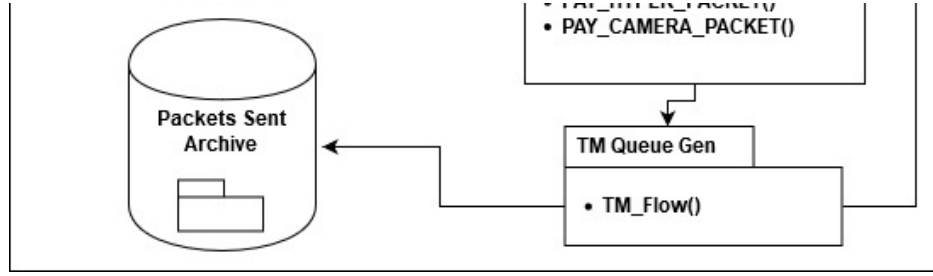


Figure 42: TM Queue Gen Module

The **TM Queue Gen** module has the main purpose of generating a sequence of strings that represent the incoming packets from the satellite. The **TM_Flow ()** method generates the queue, saves it in the *Packet Sent Archive* in the form of “*TM Queue_2019_07_12-21_25_48.txt*” and, in conclusion, sends it to the GS SIM thread for the ground station packet extraction.

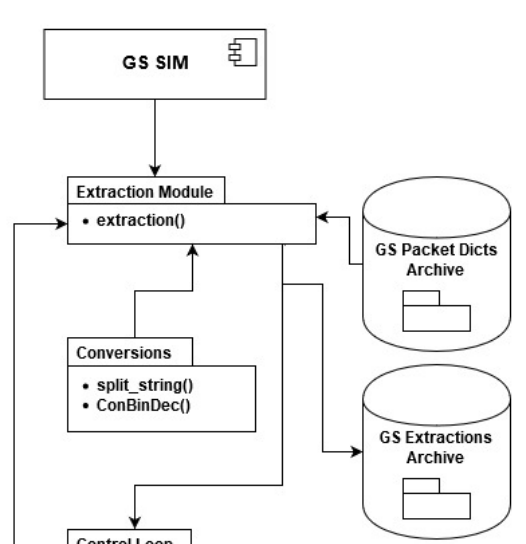


Figure 43: GS SIM Extraction Module

The TM Queue generated is sent to the GS SIM thread’s **extraction** module. The method **extraction ()** receives in input the telemetry queue and the packets dictionary structure from the *GS Packet Dicts Archive* and extracts the useful data from the packets. These data are saved in the *GS Extractions Archive* in form of dictionary divided by on-board system (for example *ADCS_RW.json* is the ground extraction related to the reaction wheels), and then they are sent to the **Control Loop** module.

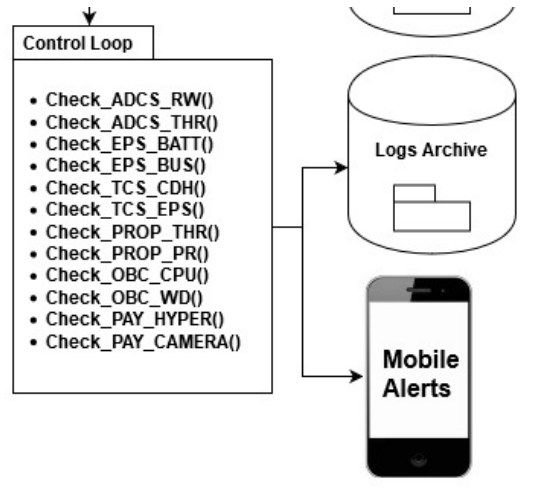


Figure 44: GS SIM Control Loop Module

The **Control Loop** module receives in input the extracted data and check if it is acceptable or if it is not acceptable. According to [1], the criteria used to evaluate the data are the following:

- **Checked:** the data is acceptable.
- **Alarm:** the data is not acceptable and not tolerable.
- **Tolerance:** the data is not acceptable, but it is tolerable.
- **OOL:** the data is out of the prefixed limits.

The results of the **Control Loop** module are saved in the *Logs Archive* and, if it happened, a message of alert is sent to the operator's mobile. This is an important skill for the control centre because in this way the operator can check the incoming telemetries directly from his mobile phone and he is not forced to be constantly present in the control center office.

4.4 SAT SIM OVERVIEW: TELECOMMAND BRANCH

The telecommand generation has a path similar to the telemetry generation. The process starts from the GS SIM thread that generate the TC packets and send them to the SAT SIM thread that extract the command from the packet and execute it. The architecture of the TC branch is shown in Figure 45.

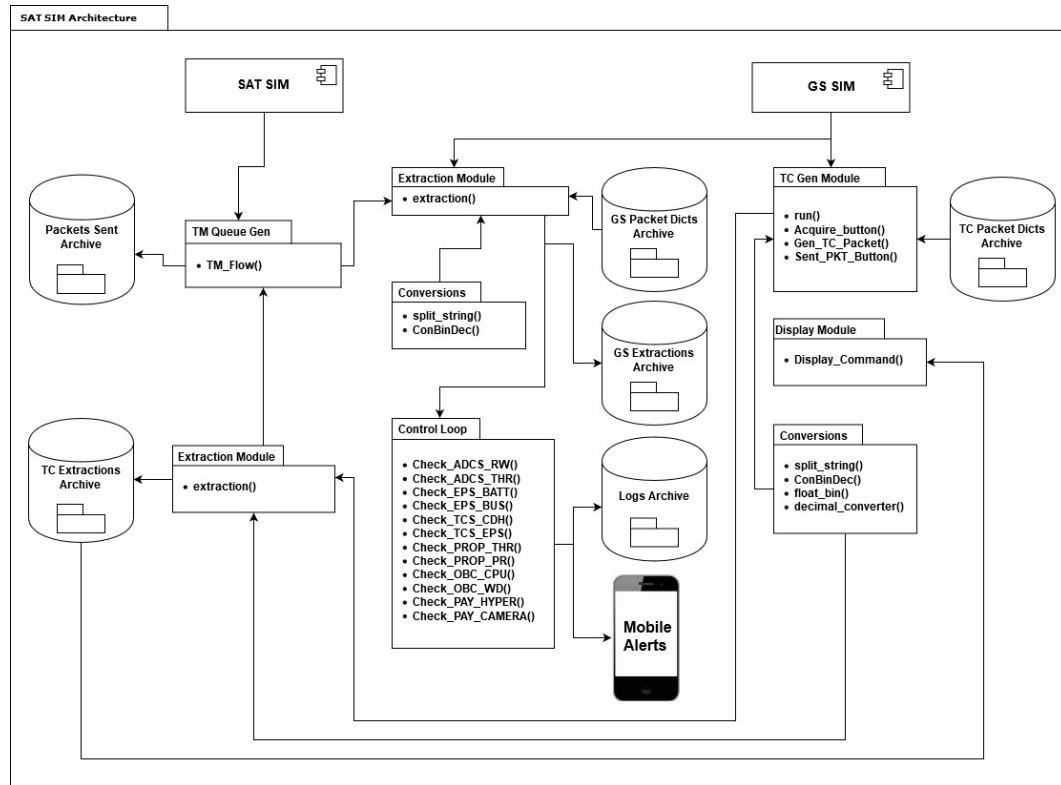


Figure 45: SAT SIM - TC Branch

The generation of telecommand is dedicated to the Python class *GS SIM*. This thread is organized into different module with different functions (methods). The **TC Gen** module has the purpose to generate the command packet for each on-board system; from the TC interface, the operator can manage the on-board equipment parameters and generate the packets to send to the SAT SIM extraction module.

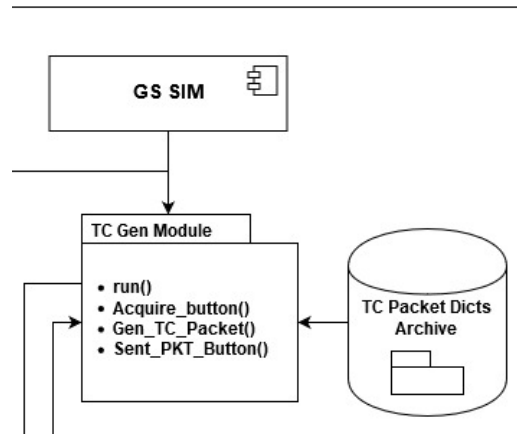


Figure 46: TC Gen Module

The TC Gen module has different modules:

- **run ()**: this method starts the thread, calls all the method in the Python class and generates the interface for the generation of the telecommand.
- **Acquire_button ()**: for each on-board system, this method has the aim to check that all the parameters changed by the operator are correct and able to be sent to the satellite. The criteria to establish if the input parameter is acceptable are the same exposed in the section 4.3.
- **Gen_TC_Packet ()**: for each on-board system, this method generates the packets according the CCSDS standards expressed in the section 3.2.2. To build the packets, this method refers to the packet dictionary structures saved in the *TC Packet Dicts Archive*.
- **Sent_PKT_Button ()**: this method generates the packet string for each on-board system and send it to the SAT SIM **extraction** module.

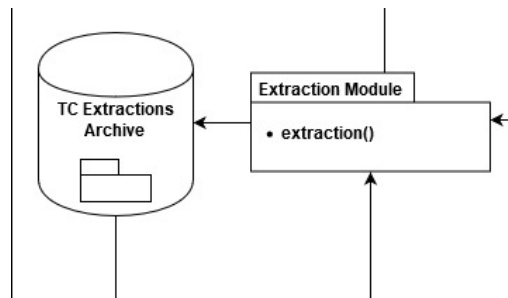


Figure 47: SAT SIM Extraction Module

The SAT SIM **extraction** module has objectives similar to the GS SIM **extraction** module. This method receives in input the packet strings and extracts all the packet fields from the strings. When the data is extracted, the extraction module executes the command updating the on-board equipment parameters and saving the data in the *TC Extractions Archive*. The useful data (in this case the command to execute) is sent to the GS SIM **TM Queue Gen** to generate new telemetry as proof of the correct execution of the command; from this point on, the path is the same path of the TM branch. In the case that the command is not executed, a message of error is sent to the operator to warn him on the incorrect event happened.

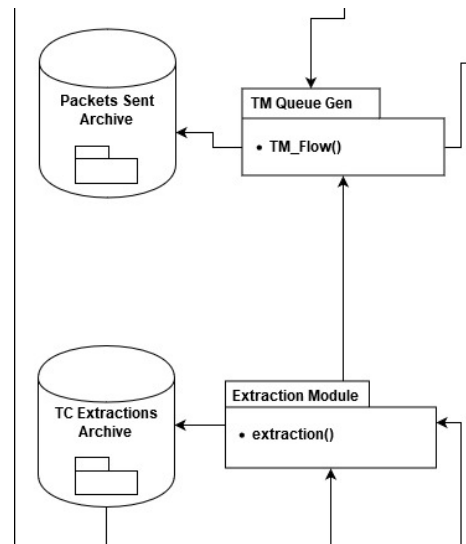


Figure 48: New TM Gen after the command execution

In conclusion, all the TC logs and data are sent to the **TC Display** module.

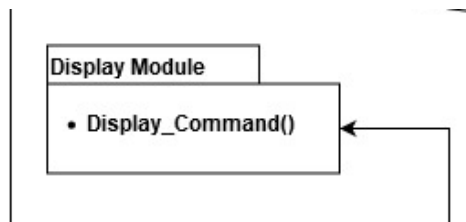


Figure 49: TC Display module

The **Display_Command ()** module has the purpose of displaying the commands sent and all the messages of correct/incorrect command execution, correct/incorrect command sending and the new telemetry as proof of the correct command execution.

4.5 SAT SIM OVERVIEW: IMAGES MANAGEMENT BRANCH

The control centre also presents the possibility to manage the incoming images sent from a payload camera. To achieve these functions, it was considered an image branch in the SAT SIM architecture called *IMG SIM* (Image Simulator).

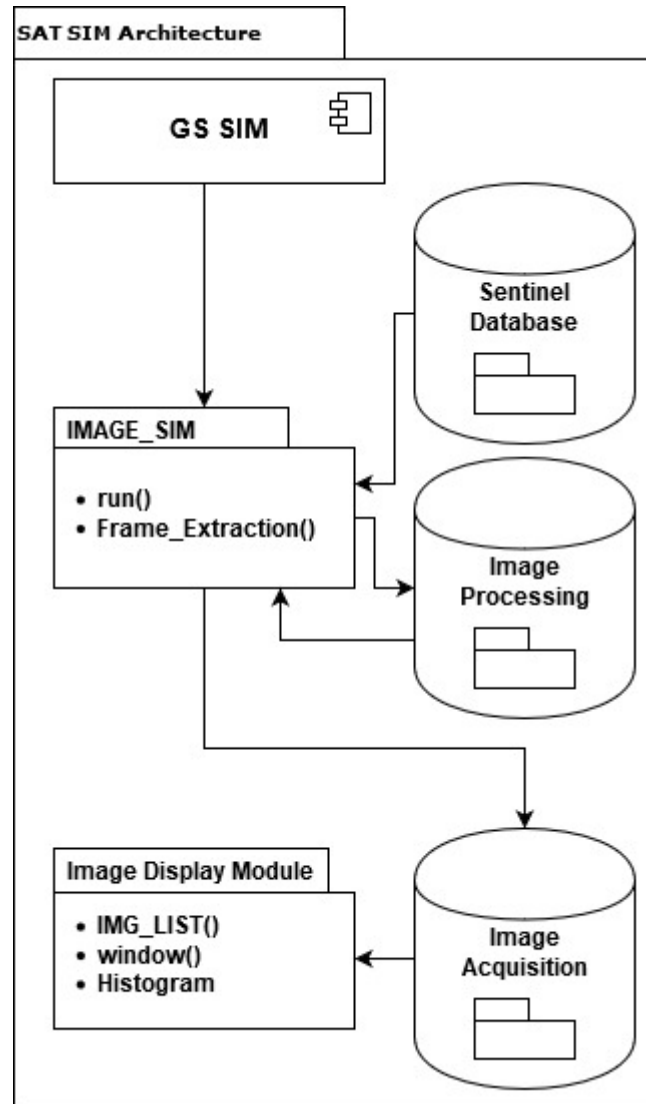


Figure 50: IMG SIM Branch

To simulate the payload camera, the SAT SIM software refers to a dataset of images taken from the *Sentinel Hub* a database of multispectral images. The master thesis work takes in exams some spectral bands and image of Turin city [12]. This database is the input to the **IMAGE_SIM** module as shown in Figure 51.

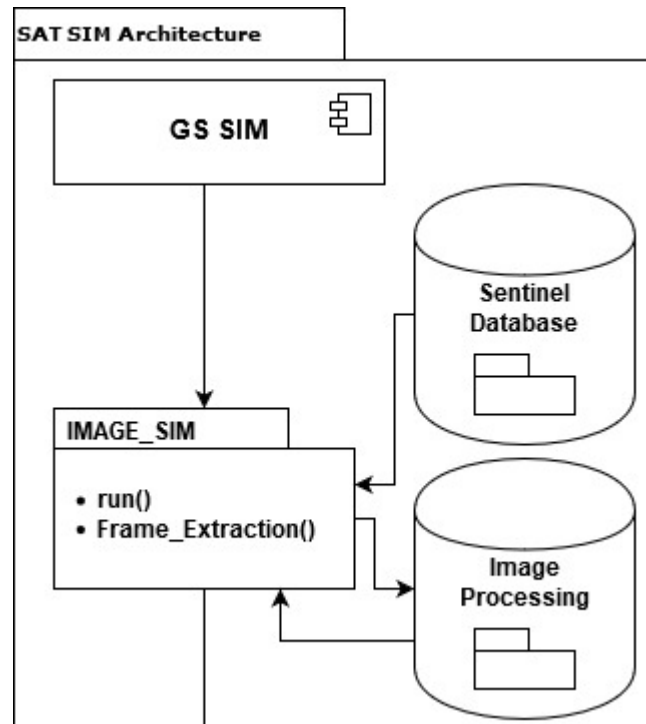


Figure 51: IMAGE_SIM Module

The IMAGE_SIM module has different methods:

- **run ():** this method starts the IMAGE_SIM thread and receive in input the files in the *Sentinel Database*. This function takes the files and compresses them into the CCSDS packets according the standards presented in the section 3.2.3. These packets are saved in the *Image Processing Archive* in form of files “Acquisition_2019_08_23-19_14_26_1.txt” and sent to the method **Frame_Extraction ()**.
- **Frame_Extraction ():** this method receives in input the image packets and extracts the useful data from them. When the extraction is ended, the frame extracted is decompressed and converted in the effective images and sent to the **Image Display** module.

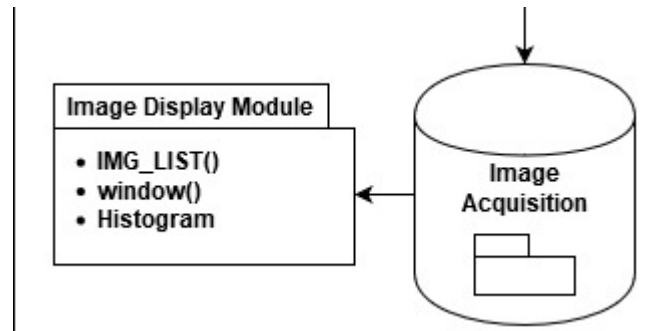


Figure 52: Image Display Module

The converted images coming from the IMAGE_SIM module are saved in the *Image Acquisition Archive* in form of file “*Acquisition_2019_08_23-17_30_27_18.jpg*” and then they are sent to the Image Display module. This module has the following methods:

- **IMG_LIST ()**: this method displays to the operator all the incoming image packets.
- **window ()**: this method allows the operator to display and share the incoming images.
- **Histogram ()**: this method provides the operator with the first post-processing operations. This function generates the RGB diagram for each incoming image and display the diagrams to the operator.

In conclusion, this branch is developed starting from the TC branch where, from the TC interface, the operator can require the sending of images from the S/c and display them.

4.6 SAT SIM OVERVIEW: INTERFACE OVERVIEW

The SAT SIM architecture is connected to an interface to allow the operator a simple use of the code to manage telemetry and commands. Python provide different tools to create interface for several applications like wxPython, PyQt and kivy. To achieve the functions of the control centre, the interface library used for the SAT SIM Interface is **Tkinter (Tk Interface)** [13].

Tkinter is Python’s standard cross-platform package for creating graphical user interfaces (GUIs). It provides access to an underlying Tcl interpreter with the Tk toolkit, with itself is a cross-platform, multilanguage graphical user interface

library [13]. Tkinter gives the ability to create windows with widgets (graphical component on the screen) in them.

The SAT SIM Interface architecture is divided into different modules, each of which is linked to a branch of the software shown in the previous sections of chapter 4.

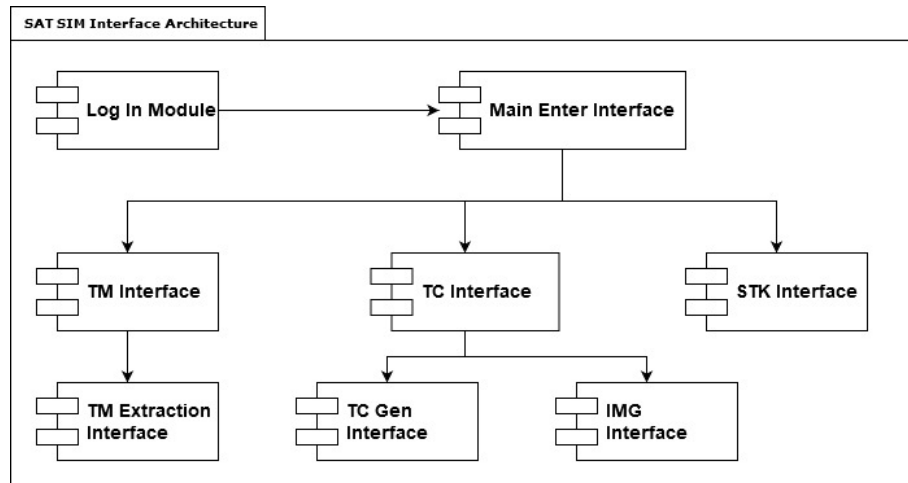


Figure 53: SAT SIM Interface architecture

As shown in Figure 53, the first module of the interface is the **Log in Module**. In this module the operator can register himself as operator or log in into the software with his credentials (name and password).

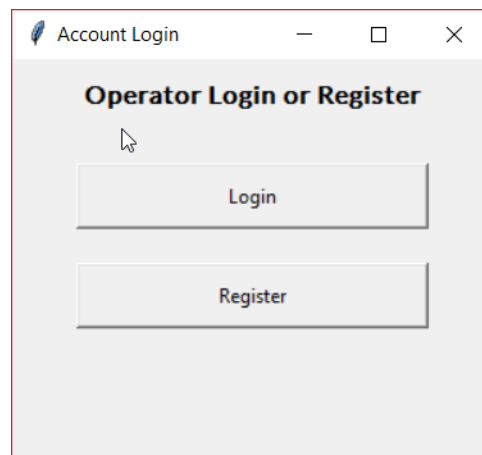


Figure 54: SAT SIM Login/Register Interface

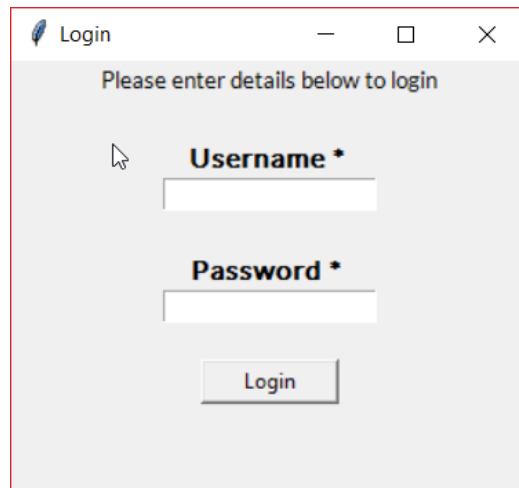


Figure 55: SAT SIM Login Interface

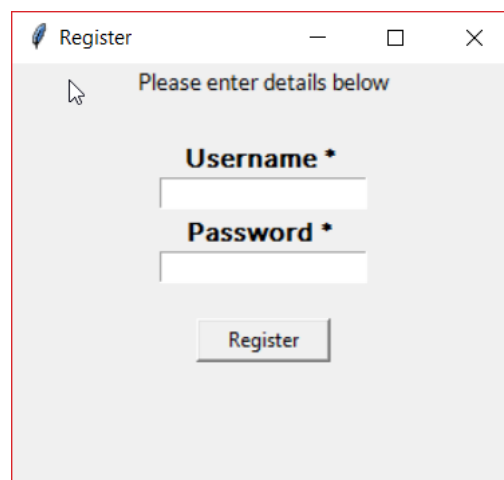


Figure 56: SAT SIM Registration Interface

After the Log-in phase, the operator can access the **Main Enter Interface** module. This is the high level interface of the SAT SIM software and from it the operator can access the TM branch or the TC branch or the STK simulations branch.

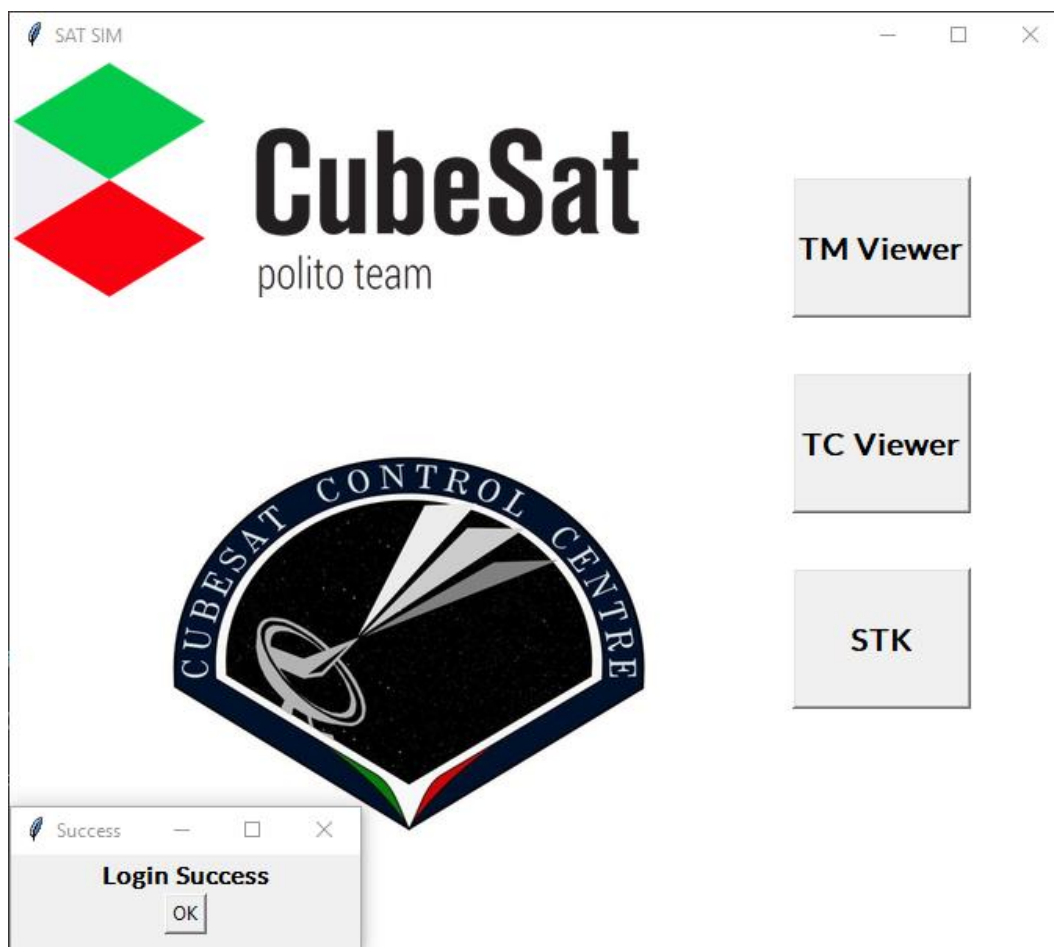


Figure 57: SAT SIM main interface

Through the *TM Viewer* button, the operator can access the TM branch. The interface shows the results of the telemetry extractions of the incoming packets as shown in Figure 58.

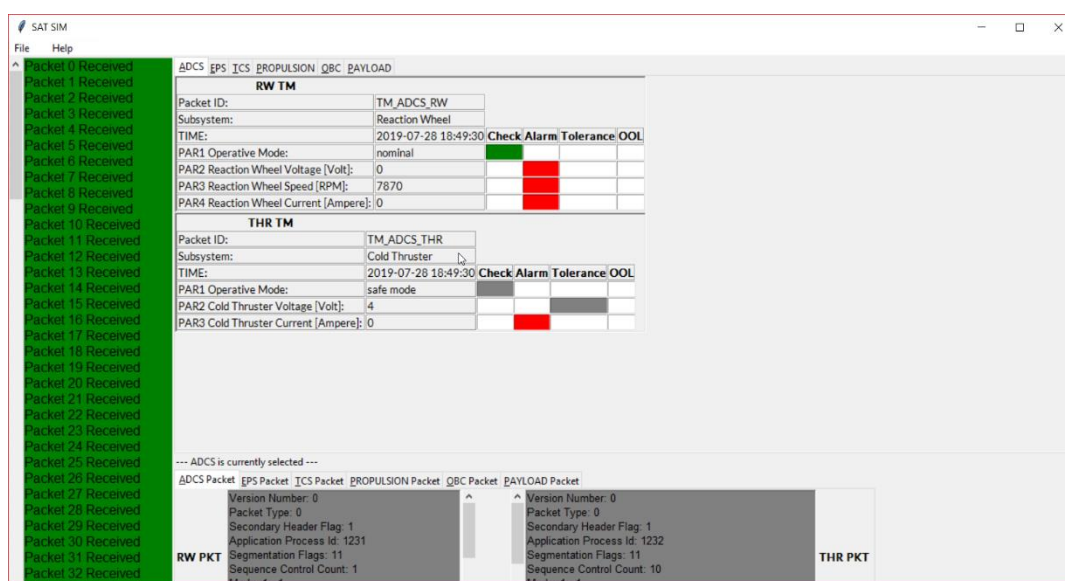


Figure 58: SAT SIM TM Interface

From the TM interface it is possible to see the following modules:

- **Incoming Packets:** the queue of the incoming packets (in green correct acquisition of the packet, red incorrect acquisition)

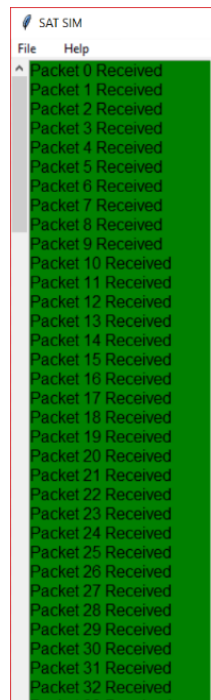


Figure 59: Queue of the incoming packets

- **Telemetry Display:** The TM information are displayed and divided by on-board system (ADCS, EPS, TCS, OBC, PROPULSION and PAYLOAD). In this section there is also the displaying of the consistency check where it is shown if the parameters are acceptable or not acceptable. In addition, in this section are shown the packet type, the time of packet sent and the packet parameters.

| ADCS EPS TCS PROPULSION OBC PAYLOAD | | | | | |
|---------------------------------------|---------------------|-------|-------|-----------|-----|
| RW TM | | | | | |
| Packet ID: | TM_ADCS_RW | | | | |
| Subsystem: | Reaction Wheel | | | | |
| TIME: | 2019-07-28 18:49:30 | Check | Alarm | Tolerance | OOL |
| PAR1 Operative Mode: | nominal | Green | | | |
| PAR2 Reaction Wheel Voltage [Volt]: | 0 | | Red | | |
| PAR3 Reaction Wheel Speed [RPM]: | 7870 | | Red | | |
| PAR4 Reaction Wheel Current [Ampere]: | 0 | | Red | | |
| THR TM | | | | | |
| Packet ID: | TM_ADCS_THR | | | | |
| Subsystem: | Cold Thruster | | | | |
| TIME: | 2019-07-28 18:49:30 | Check | Alarm | Tolerance | OOL |
| PAR1 Operative Mode: | safe mode | | | | |
| PAR2 Cold Thruster Voltage [Volt]: | 4 | | | | |
| PAR3 Cold Thruster Current [Ampere]: | 0 | | Red | | |

Figure 60: SAT SIM TM Display interface

- **Packet Visualization:** on the TM interface, the operator can also visualize the structure of the incoming packet divided by on-board system. In this way the operator can check if the structure of the CCSDS packet is respected.

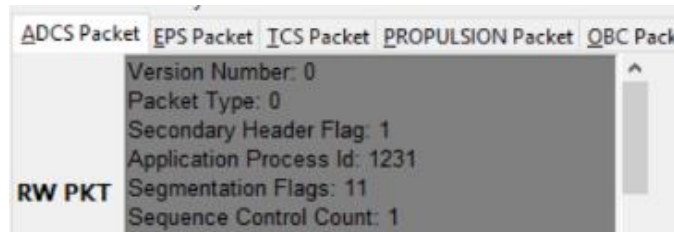


Figure 61: SAT SIM TM Packet Visualization

From the TM Interface is possible to access to the packet archive. Through this ability is possible to check a specific packet sent in a specific data and time.

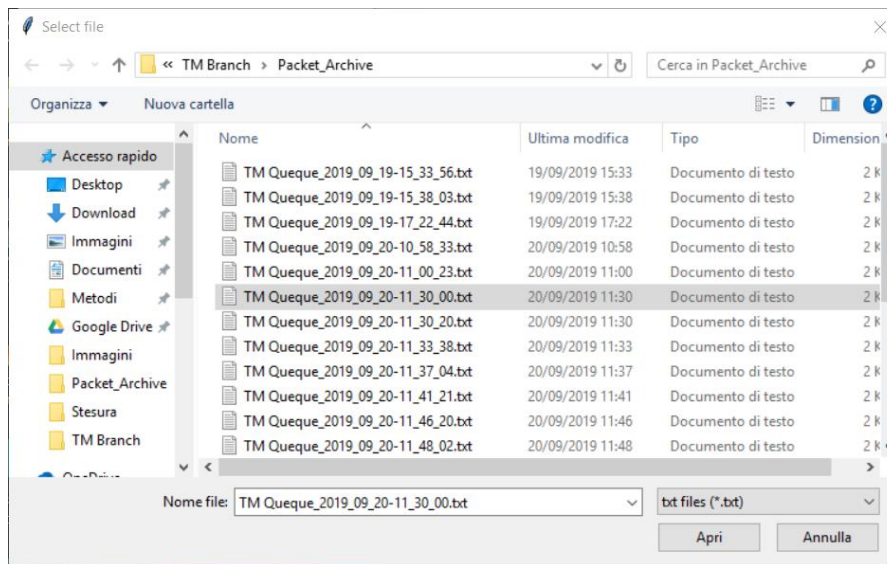


Figure 62: Packet Archive

After that the operator chooses the packet to display, he can open it packet and can display all the telemetry data sent in that specific data at that specific time.

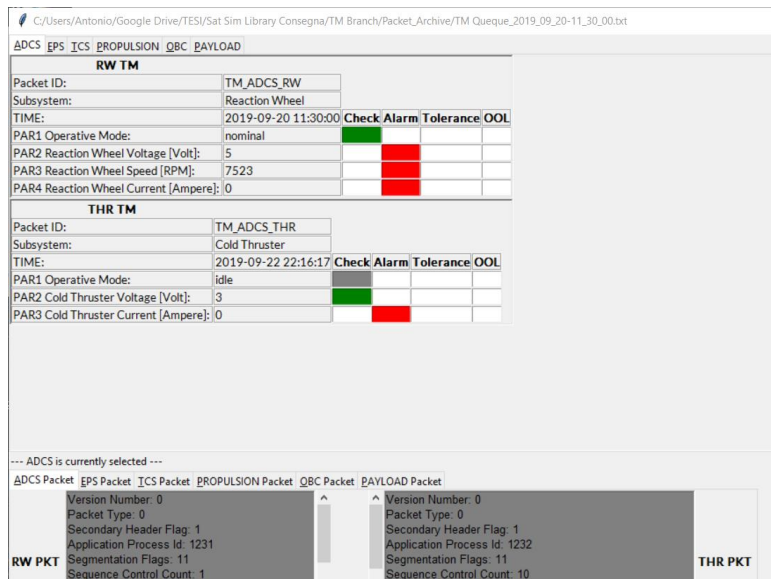


Figure 63: Display of a specific TM packet

The interface for the generation of the command packet is similar to the TM interface in which the operator can choose the parameters to change and generate TC packets to sent to the satellite.

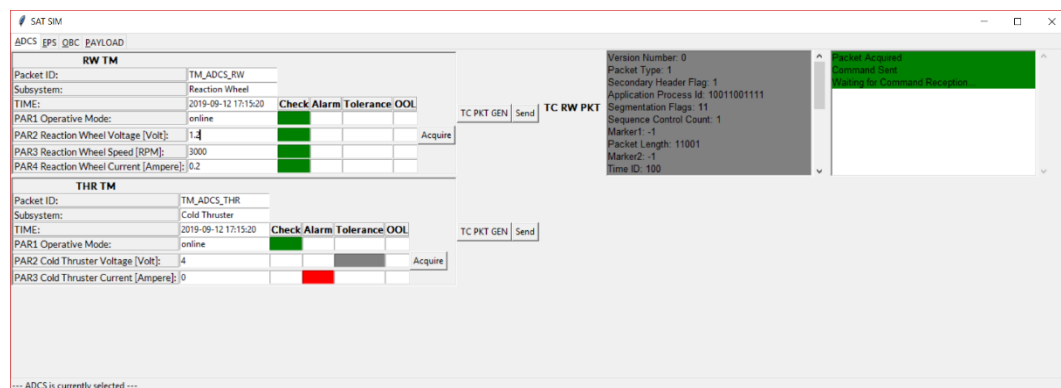


Figure 64: TC Interface

In detail, from Figure 64 it is possible to distinguish the following modules:

- **Telecommand Manage Interface:** from this module the operator can change the not acceptable parameters and can check if the new chosen parameter is acceptable or not through the *Acquire* button that call the consistency check methods to control the different data.

SAT SIM

ADCS EPS QBC PAYLOAD

RW TM

| | | | | | |
|---------------------------------------|---------------------|-------|-------|-----------|---------|
| Packet ID: | TM_ADCS_RW | | | | |
| Subsystem: | Reaction Wheel | | | | |
| TIME: | 2019-09-12 17:15:20 | Check | Alarm | Tolerance | OOL |
| PAR1 Operative Mode: | online | | | | |
| PAR2 Reaction Wheel Voltage [Volt]: | 1.4 | | | | Acquire |
| PAR3 Reaction Wheel Speed [RPM]: | 3000 | | | | |
| PAR4 Reaction Wheel Current [Ampere]: | 0.2 | | | | |

Figure 65: TC Manage Interface

- TC Packet Generation Module:** through the *TC PKT GEN* the operator generates the TC packets according to the CCSDS standards examined in the section 3.2.2. At this moment the packet is only a dictionary where the keys are the CCSDS packet fields and the values are the new parameters generated by the operator. Instead, with the *Send* button, the TC packet is converted into a binary string and sent to the extraction module of the SAT SIM thread. As proof of the correct sending, messages of information are displayed to the operator on the interface.

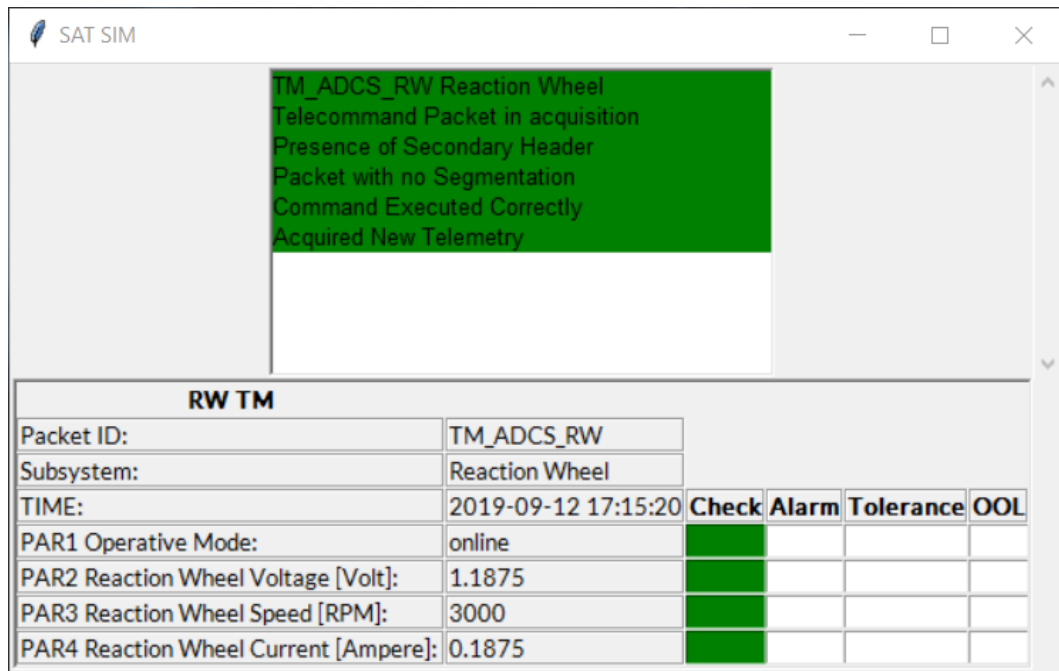
Version Number: 0
 Packet Type: 1
 Secondary Header Flag: 1
 Application Process Id: 10011001111
 Segmentation Flags: 11
 Sequence Control Count: 1
 Marker: 1

TC PKT GEN Send TC RW PKT

Packet Acquired
 Command Sent
 Waiting for Command Reception

Figure 66: TC Packet Generation Module Interface

When the packet is sent to the extraction module, the S\c executes the command sent, updates the telemetry and generates a new TM packet as proof of the correct command execution.



TM_ADCS_RW Reaction Wheel
Telecommand Packet in acquisition
Presence of Secondary Header
Packet with no Segmentation
Command Executed Correctly
Acquired New Telemetry

| RW TM | | | | | |
|---------------------------------------|---------------------|-------|-------|-----------|-----|
| Packet ID: | TM_ADCS_RW | | | | |
| Subsystem: | Reaction Wheel | | | | |
| TIME: | 2019-09-12 17:15:20 | Check | Alarm | Tolerance | OOL |
| PAR1 Operative Mode: | online | | | | |
| PAR2 Reaction Wheel Voltage [Volt]: | 1.1875 | | | | |
| PAR3 Reaction Wheel Speed [RPM]: | 3000 | | | | |
| PAR4 Reaction Wheel Current [Ampere]: | 0.1875 | | | | |

Figure 67: Correct execution of the command

The *STK* button allows the operator to call a new STK scenario to simulate access and different actions before effectively sending them to the satellite. With the STK software the operator can also simulate the track of satellite before update the TLE date in the real tracking system of the C3.

The last part of the software is the capability of schedule different type of pre-set commands ordered by the operator and executed by the SAT SIM thread.

The **schedule part** consists into receive in input a command with a specific structure shown in Figure 68.

```
command_name_dict = {
    "display_name":
    "description":
    "resources":
    "priority":
}
```

Figure 68: Command Structure

The command structure in input is a dictionary that has the follow keys:

- **Command_name_dict:** name of the dictionary that indicates the name of the command.

- **Display name:** is the name of the command displayed on the interface.
- **Description:** is a brief description of what the command does.
- **Resources:** are the sequence of resource involved in the execution of the command.
- **Priority:** is a number that establish the priority with which the command must be execute. The scheduler, in fact, will execute the command with the highest priority (priority = 1) and then in sequence all the others.

In this part, 8 type of commands are pre-set in the scheduler. The commands are the following:

- **Pings:** is the simple request to the satellite to ping a signal.
- **Error:** is the command to evidence an error.
- **Connect:** is the command to points antennas and starts broadcasting carrier signal to establish RF lock with the spacecraft.
- **Safe Mode:** is the command to switch the spacecraft in its safe mode state.
- **Detumbling:** is the command to star all the sequences for the execution of the spacecraft detumbling mode.
- **Offline:** is the command to turn off all the systems.
- **Nominal:** is the command to set all the systems in their operative mode (all the systems operate in their nominal mode).
- **Acquire Event:** is the command to acquire a specific event with the payload.

From the interface the operator can choose what kind of command want to select for the schedule. Each command is associate with a tab where there are all the details of the chosen command and the possibility to assign a priority number to it. After assigned the priority to the command, the operator can add the command to the schedule queue that are waiting to be executed.



Figure 69: Scheduler Interface Command Choose

When the operator chooses the command to schedule, he can see all the details relative to that specific command as shown in Figure 70.

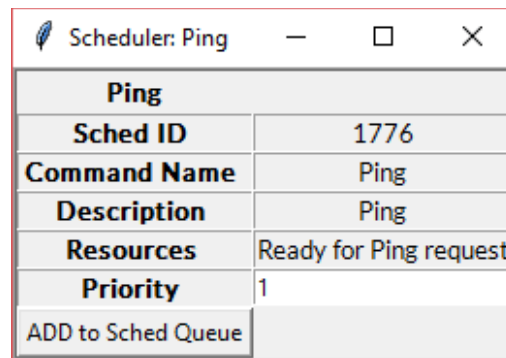


Figure 70: Command detail window

From the window shown in Figure 6, is possible to see the following fields:

- **Schedule ID:** is the ID generate for the command to add to the schedule queue.
- **Command Name**
- **Description:** Description of the command to choose.
- **Resources:** the resources involved in that specific command.
- **Priority:** the operator can assign a priority number to the command.

This number will be read by the scheduler that will execute all the schedule queue sorted by priority number.

Through the *ADD to Sched Queue* button the operator can add the command to the queue and choose another command to schedule in the same way.

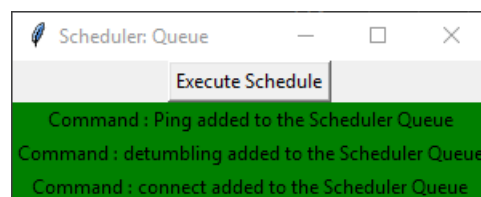


Figure 71: Scheduler Queue Interface

Through the *Execute Schedule* button, the entire schedule will be execute starting from the command with the highest priority (1) and then in sequence all the other commands.



Figure 72: Example of a schedule execution

In conclusion, all the SAT SIM software provides the possibility to manage TM and TC and Images packets following the CCSDS standards and then to simulate a schedule of commands for the correct execution and monitoring of the spacecraft operations.

In the next session it will be described all the test sessions used to validate and verify the entire software and its interface.

5. TEST & VALIDATION

The software testing life cycle typically includes different phases like: Planning, Analysis, Design, Construction, Testing Cycles, Final testing and implementation and Post implementation. Each phase is described with the respective activities as follows [14]:

- **Planning:** includes the high-level test plan, the quality goals plan, problem identification and classification, acceptance criteria, measurement criteria and the reporting procedures.
- **Analysis:** involves activities that develop test cycles, identify test case, plan the test cycles required for the project and review of documentation.
- **Design:** in this design phase the activities included are revision tests based on software changes, revision and addition of new test cases based on software changes, finalization of the test cycles (number of test case per cycle) and finalization of the test plan.
- **Coding:** complete all plans from test cycle to the automated testing and fix the bugs (bug reporting, verification, and revision/addition of the test cases).
- **Verification:** this phase includes the execution of all test cases (automated and manual), updating estimates for test cases and test plans, document test cycles, regression testing, and updating accordingly.
- **Validation:** activities in this phase are review of the test cases to evaluate other cases to be automated, clean up the automated test cases and variables and review process of integrating results from automated testing in with results from manual testing.

In the next sections it will be reported, for the SAT SIM software, all the test cases classification and tracking according to the test objectives and requirements.

5.1 OBJECTIVES & REQUIREMENTS

The objectives of this last part of the master thesis work are to evaluate the software performances, to verify the software and to validate it with test cases. In particular, in this section all the procedures included in the verification of the software requirements and all the test cases used to validate the software will be explained.

The **Verification** process checks if the software is conformed to its specification and requirements (in general this phase answer to the question “Are we building the product in the right way?”). Some of the requirements for the software are shown in Table 9.

| REQ ID | Text |
|--------|---|
| REQ-1 | <i>C3 shall recognize the TM packets</i> |
| REQ-2 | <i>C3 shall acquire the TM packets</i> |
| REQ-3 | <i>C3 shall save the TM packets</i> |
| REQ-4 | <i>C3 shall recognize the TC packets</i> |
| REQ-5 | <i>C3 shall built the TC packets</i> |
| REQ-6 | <i>C3 shall save the TC packets</i> |
| REQ-7 | <i>C3 shall recognize the Image packets</i> |
| REQ-8 | <i>C3 shall acquire the Image packets</i> |
| REQ-9 | <i>C3 shall save the Image packets</i> |
| REQ-10 | <i>C3 shall share information with the mission stakeholders</i> |
| REQ-11 | <i>C3 shall share images with public</i> |
| REQ-12 | <i>C3 shall display the acquired information</i> |
| REQ-13 | <i>C3 shall guarantee the existence of an interface between operator and PC</i> |
| REQ-14 | <i>C3 shall be developed according to ECSS standards</i> |
| REQ-15 | <i>C3 shall be compliant to CCSDS standard</i> |
| REQ-16 | <i>C3's control centre shall be implemented in Python</i> |
| REQ-17 | <i>C3 Scheduler shall recognize a specific structure for pre-set commands</i> |
| REQ-18 | <i>C3 Scheduler shall generate a unique ID for the pre-set commands each time a schedule is created</i> |
| REQ-19 | <i>C3 Scheduler shall provide an interface to display the commands information</i> |
| REQ-20 | <i>C3 Scheduler shall generate a queue sort by the command's priority</i> |
| REQ-21 | <i>C3 Scheduler shall provide an interface to display the correct execution of the schedule</i> |

Table 9: Software Requirements

The **Validation** process, instead, checks if the software does what the user really requires; this goes beyond checking that the software meets its specification, but this phase requires that the software is able to achieve the mission objectives and the needs of the mission stakeholders [14]. The whole life-cycle process of the Verification and Validation (V&V process) must be applied at each stage of the software process and has two principal objectives:

- The discovery of defects in a system.
- The assessment of whether or not the system is usable in an operational situation.

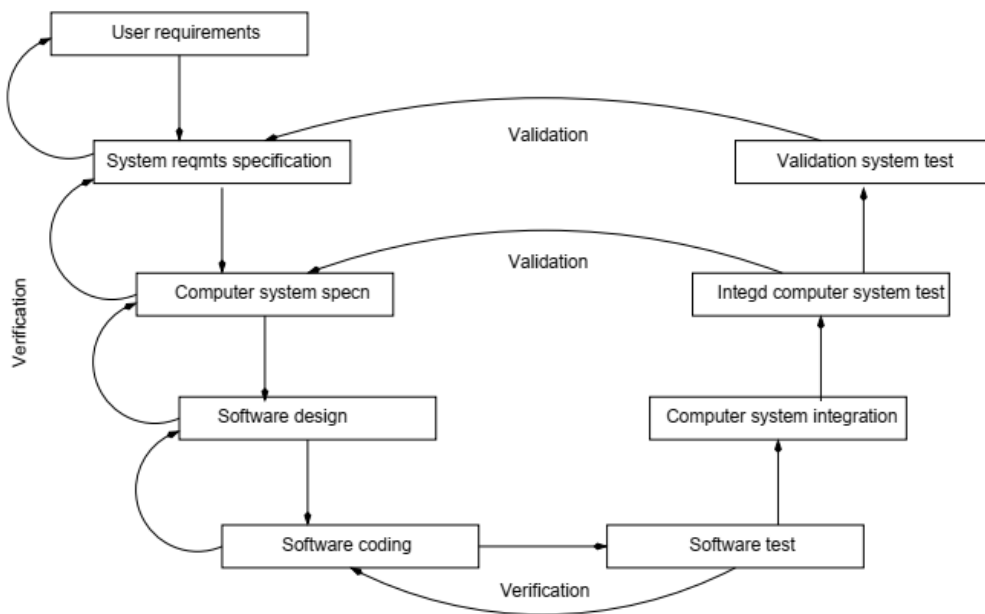


Figure 73: The V-model of the V&V process

The V&V process establishes a degree of confidence that the software is fit for purpose; this does not mean that it is completely free of defects and the degree of confidence depends upon several different factors as the test typology used to validate the software and the experience of who tests the code.

To achieve the objectives of the V&V process, in the next sections all the tests conducted, and the respective results will be described.

5.2 TEST SESSIONS

The test sessions have the main purpose of locating the defects of the software. Each test should be repeatable, but there are some exceptions in case the software changes the test environment without the possibility to restore it or in case there are some indeterministic elements (non-controllable inputs) in the code.

To take a test it is useful to consider the following points:

- It is important to know the expected behaviour to compare with the observed behaviour from the code.
- During all the test it is important to have an *Oracle* that knows the expected results for each test case. It is possible to have a *human Oracle*, the operator follows the software specification and compare the expected results with the real results, or an *Automatic Oracle* that is generated by the software specification. It is possible that this oracle could be the same software but developed by other operators or a previous version of the same software. In this test session the oracle is the operator that compares the expected results with the real results.

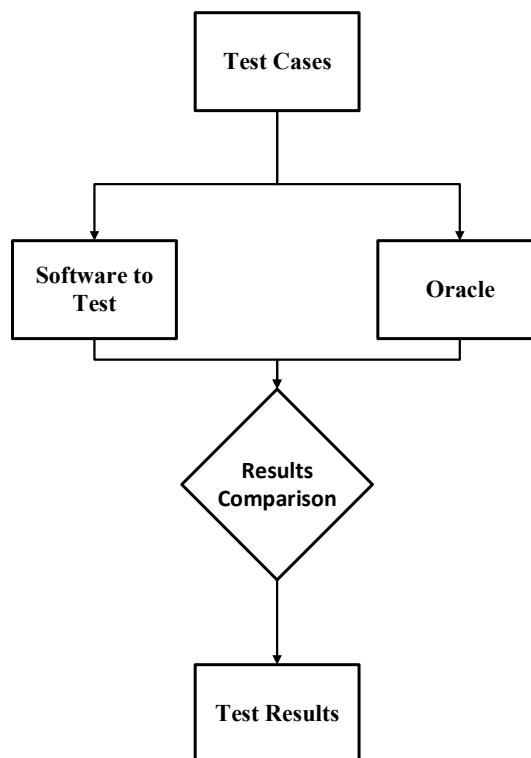


Figure 74: Test evaluation flowchart

To establish if a test is ended and if it is successful, it is important to create the *pass criteria* for each test, which establish if the test is passed by the software or if the test is failed by the software. The number of the tests that can occur in validating the software depends from time available for each test, coverage (test all the macro areas of the code) and from statistics criteria (if the last test cases are passed it is possible to end the validation process).

It is not possible to evaluate the ideal number of the test cases, but each of them is described by the following parameters:

- **Effectiveness:** it is the rate between the number of bugs found and the number of bugs to find.
- **Efficiency:** it is the rate between the number of tests able to find bugs and number of total tests.

To keep track of the results, each test is characterized by different identification field as shown in Table 10.

| Test Case ID | Test Case Description | Input Data/ Requirements | Expected Result | Pass/Fail |
|--------------|-----------------------|--------------------------|-----------------|-----------|
| | | | | |

Table 10: Test Classification

From Table 10 it is possible to see:

- **Test Case ID:** this is the identification code of each test conducted. The ID is simple and structured as in the example: *TC-001* (the ID code of the first test case).
- **Test Case Description:** this is a description of the objectives of the test case and how the test case is conducted.
- **Input Data/ Requirements:** this field describes the type of inputs for each test case (if there are input) and which requirements the test case would need to verify and test.
- **Expected Result:** this field describes the expected result (if there is any) for each test case.
- **Pass/Fail:** these are the results of the test. (P = pass, F = fail).

5.2.1 TEST SESSION: DEBUGGING PROCESS

The Debugging process is concerned with locating and repairing the errors discovered in the code. Debugging involves:

- To formulate a hypothesis about program behaviour.
- To test these hypotheses to find the system error.

There is no simple process for debugging and it often involves looking for patterns in test outputs with defect and using a programmer's skill to locate the error. The Debugging process includes the location and repairing of errors like syntax errors (usually caught by the compiler which locates the error occurred in and the type of error), and semantic errors (logical error) which occurred when the software produces incorrect output on some input. These errors are harder to detect since the compiler may not be able to indicate where and what the problem is.

Once errors are located and fixed, it is necessary to re-test the program to make sure that the fix operation has not introduced new problems. Experience could help the programmer to reduce the introduction of new errors in the debugging process.

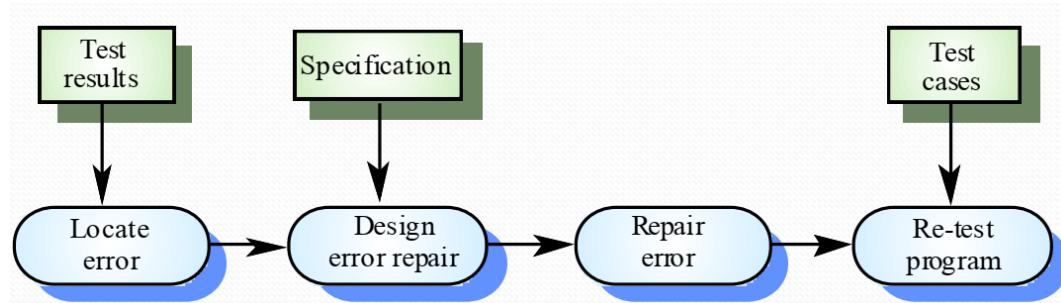


Figure 75: The Debugging Process

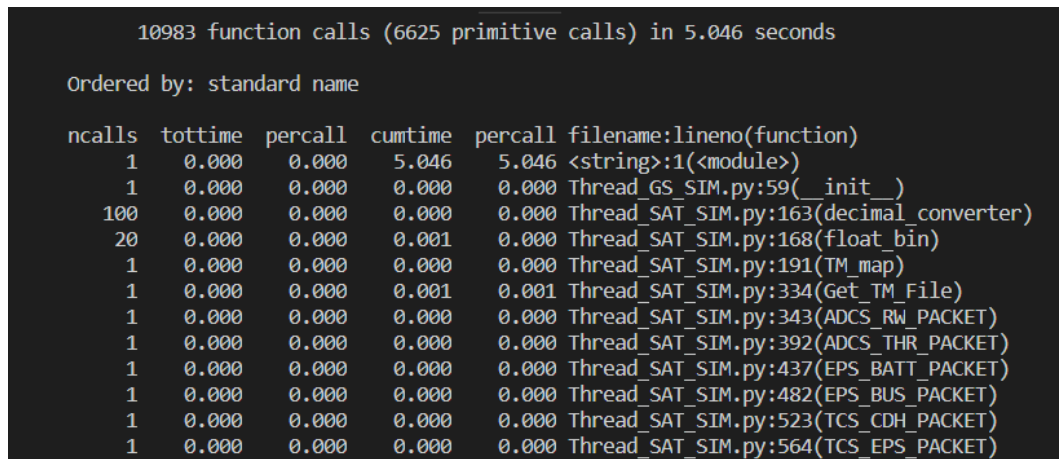
5.2.2 TEST SESSION: SOFTWARE PROFILING

One of the first tests performed after the physiological debugging phase is the tracking of software performances. Testing the software's performance means monitoring the execution times of the various classes and the entire software and track the entire software path to control the presence of errors.

In Python it is possible to monitor all these characteristics using its profiler. A profiler is a program that describes the run time performance of a code, providing a variety of statistics and graphs. The profiler provides also a series of report generation tools to allow users to rapidly examine the results of a profile operation.

The Python profiler library used to profile the SAT SIM software is **cProfile**. It is a C extension with reasonable overhead that makes it suitable for profiling long-running programs.

The module `cProfile.run()` receives in input the function to profile and returns as output a series of statistics that describe the function in all its performances. As first profile in this test session, the SAT SIM main is profiled as shown in Figure 76.

The image shows a terminal window with the output of the cProfile module. The first line indicates that 10983 function calls (6625 primitive calls) were monitored in 5.046 seconds. The second line shows the output is ordered by standard name. The third line is a header for the table of results. The table has five columns: ncalls, tottime, percall, cumtime, and filename:lineno(function). The data rows show the performance of various functions, including the main module, Thread_GS_SIM.py, and several Thread_SAT_SIM.py functions.

| 10983 function calls (6625 primitive calls) in 5.046 seconds | | | | |
|--|---------|---------|---------|--|
| Ordered by: standard name | | | | |
| ncalls | tottime | percall | cumtime | filename:lineno(function) |
| 1 | 0.000 | 0.000 | 5.046 | 5.046 <string>:1(<module>) |
| 1 | 0.000 | 0.000 | 0.000 | 0.000 Thread_GS_SIM.py:59(__init__) |
| 100 | 0.000 | 0.000 | 0.000 | 0.000 Thread_SAT_SIM.py:163(decimal_converter) |
| 20 | 0.000 | 0.000 | 0.001 | 0.000 Thread_SAT_SIM.py:168(float_bin) |
| 1 | 0.000 | 0.000 | 0.000 | 0.000 Thread_SAT_SIM.py:191(TM_map) |
| 1 | 0.000 | 0.000 | 0.001 | 0.001 Thread_SAT_SIM.py:334(Get_TM_File) |
| 1 | 0.000 | 0.000 | 0.000 | 0.000 Thread_SAT_SIM.py:343(ADCS_RW_PACKET) |
| 1 | 0.000 | 0.000 | 0.000 | 0.000 Thread_SAT_SIM.py:392(ADCS_THR_PACKET) |
| 1 | 0.000 | 0.000 | 0.000 | 0.000 Thread_SAT_SIM.py:437(EPS_BATT_PACKET) |
| 1 | 0.000 | 0.000 | 0.000 | 0.000 Thread_SAT_SIM.py:482(EPS_BUS_PACKET) |
| 1 | 0.000 | 0.000 | 0.000 | 0.000 Thread_SAT_SIM.py:523(TCS_CDH_PACKET) |
| 1 | 0.000 | 0.000 | 0.000 | 0.000 Thread_SAT_SIM.py:564(TCS_EPS_PACKET) |

Figure 76: cProfiler output of SAT SIM main

The first line indicates that 10983 calls were monitored and, of those calls, 6625 were *primitive*. The term *primitive* indicate that these calls were not induced via recursion. The next line *Ordered by: standard name*, indicates that the text string in the far right column was used to sort the output. The other columns include:

- **ncalls**: number of calls.
- **tottime**: total time spent in the given function (and excluding time made in call to sub-functions).
- **percall**: the tottime divided by ncalls.
- **cumtime**: total time spent in this and all subfunctions (from invocation till exit).
- **percall**: the cumtime divided by primitive calls.
- **filename:lineno(function)**: provide the respective data of each function.

It is possible to find two numbers in the first column like 43/3; that means that the second number is the number of primitive calls and the first is the actual number of calls. When the function does not recurse, these two values are the same, and only the single number is printed.

In the same way the interface code is profiled, and the outputs are shown in Figure 77.

```
87256 function calls (81246 primitive calls) in 55.677 seconds
```

Ordered by: standard name

| ncalls | tottime | percall | cumtime | percall | filename:lineno(function) |
|--------|---------|---------|---------|---------|--|
| 49/7 | 0.000 | 0.000 | 0.078 | 0.011 | <frozen importlib._bootstrap>:1009(_handle_fromlist) |
| 16 | 0.000 | 0.000 | 0.000 | 0.000 | <frozen importlib._bootstrap>:103(release) |
| 16 | 0.000 | 0.000 | 0.000 | 0.000 | <frozen importlib._bootstrap>:143(__init__) |
| 16 | 0.000 | 0.000 | 0.000 | 0.000 | <frozen importlib._bootstrap>:147(__enter__) |
| 16 | 0.000 | 0.000 | 0.000 | 0.000 | <frozen importlib._bootstrap>:151(__exit__) |
| 16 | 0.000 | 0.000 | 0.000 | 0.000 | <frozen importlib._bootstrap>:157(_get_module_lock) |
| 16 | 0.000 | 0.000 | 0.000 | 0.000 | <frozen importlib._bootstrap>:176(cb) |
| 31/6 | 0.000 | 0.000 | 0.078 | 0.013 | <frozen importlib._bootstrap>:211(_call_with_frames_removed) |
| 109 | 0.000 | 0.000 | 0.000 | 0.000 | <frozen importlib._bootstrap>:222(verbose_message) |
| 16 | 0.000 | 0.000 | 0.000 | 0.000 | <frozen importlib._bootstrap>:307(__init__) |
| 16 | 0.000 | 0.000 | 0.000 | 0.000 | <frozen importlib._bootstrap>:311(__enter__) |

Figure 77: cProfiler output of the Interface main

To visualize the actual calls and the connection between the classes the profiler provides some library to automatically generate a graph of all connections. The Python library used to generate graphs is *pycallgraph*. It is a library created to visual profiling tool for Python application. Its major function is to track the name of every function called, the time take within each function, number of calls and other statistics.

In the Figure 78-79 the profiling graph of the SAT SIM main and the Interface main are shown (for reason of clarity and space only a part of the graph is reported, for the detail of the graph see the Appendix).

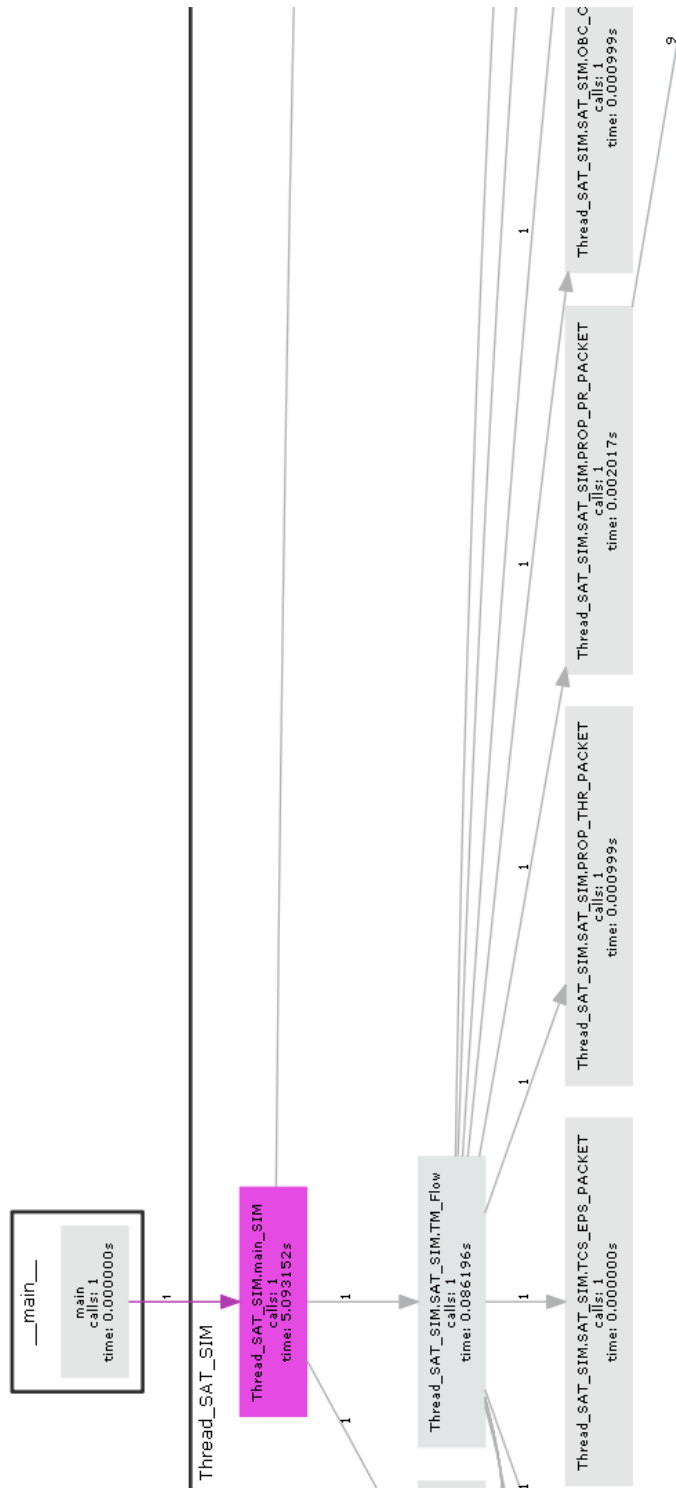


Figure 78: Call graph of the SAT SIM main

From the profiling table and from the graph is possible to see that the execution time of the code is about 5.040 seconds and it is possible to monitor all the connection in the code.

As last profile, the call graph of the Interface main is shown below (for reason of clarity and space only a part, for example purpose, of the graph is reported, for the detail of the graph see the Appendix):

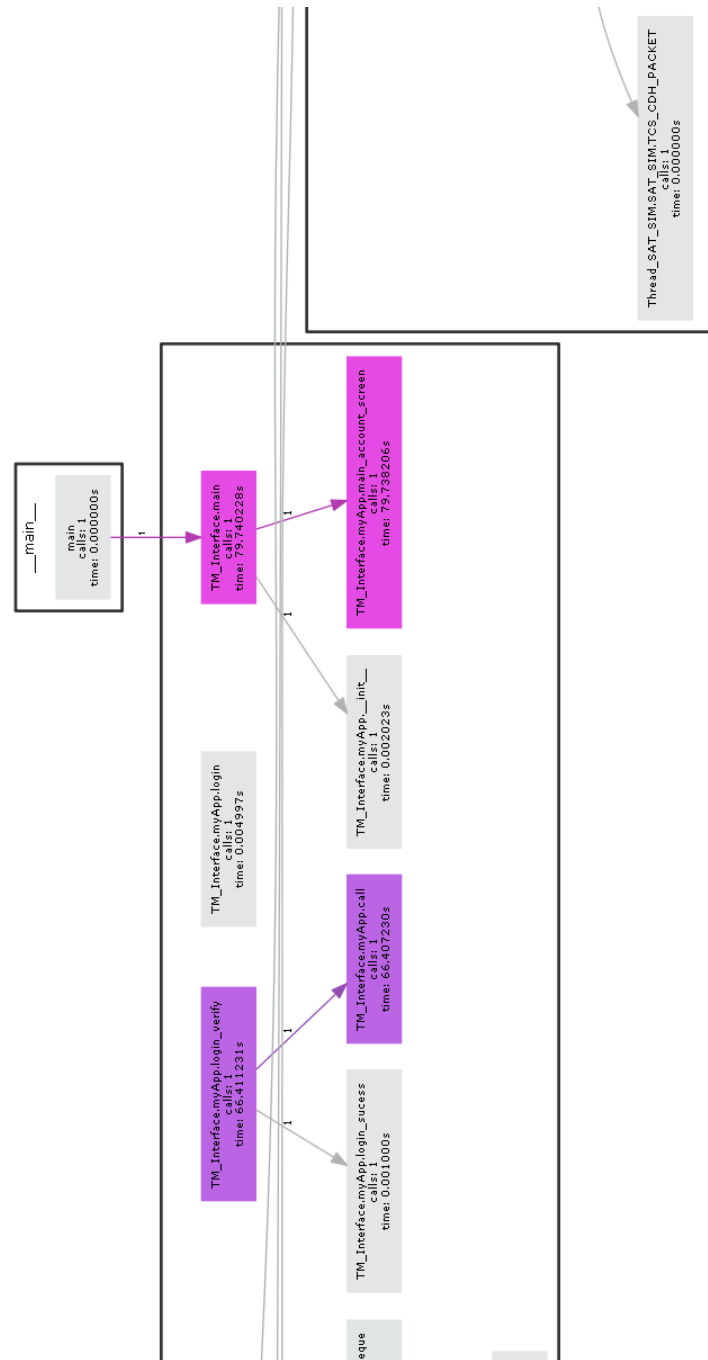


Figure 79: Call graph of Interface main

The graph of the Interface main is only indicative of what and which are the functions called in the code, the execution time depends by the operator that use the interface.

5.2.3 TEST SESSION: TM/TC NOMINAL PROFILE (Tc-001)

The test sessions of the SAT SIM software consist into establishing different test cases (Tc) in which the operator can compare the expected result with the actual outputs generate by the code. It is important to say that each Tc is aimed at verification of the requirements expressed in the section 5.1.

The first test case execute is the generation of a telemetry nominal profile. This test consists into verifying if the SAT SIM thread is able to generate telemetry in the acceptable range (nominal range) and if it is able to packetize them and sent them to the GS SIM thread. In addition, the test has the purpose to establish if the GS SIM can recognize the packet, extract the useful data and convert them into an engineering language. In conclusion, if the interface is able to display the correct TM generated and if it is able to recognize the nominal profile, the test is considered as passed.

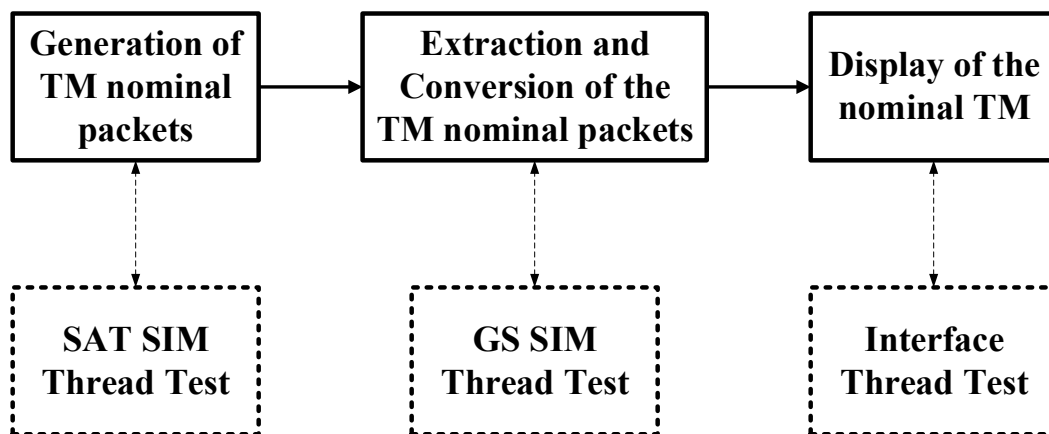


Figure 80: Tc Nominal TM generation flowchart

The first step of the test case is to generate the nominal TM profile and packets; in this phase the SAT SIM Thread is under test.

The second step of the test it to receive correctly the packets, extract the useful parameters from them and check the correct execution after the extraction, in this phase the GS SIM Thread is under test.

The last step is to test the interface. From the interface it is possible to check if the previous two phases are ended correctly and if the test is passed.

SAT SIM

File

Help

Packet 4 Received

Packet 5 Received

Packet 6 Received

Packet 7 Received

Packet 8 Received

Packet 9 Received

Packet 10 Received

Packet 11 Received

Packet 12 Received

Packet 13 Received

Packet 14 Received

Packet 15 Received

Packet 16 Received

Packet 17 Received

Packet 18 Received

Packet 19 Received

Packet 20 Received

Packet 21 Received

Packet 22 Received

Packet 23 Received

Packet 24 Received

Packet 25 Received

Packet 26 Received

Packet 27 Received

Packet 28 Received

Packet 29 Received

Packet 30 Received

Packet 31 Received

Packet 32 Received

Packet 33 Received

Packet 34 Received

Packet 35 Received

ADCS

EPS

TCS

PROPULSION

QBC

PAYLOAD

BATT TM

Packet ID:TM_EPS_BATT

Subsystem:Batteries

TIME:2019-09-23 14:23:46

PAR1 Operative Mode:online

PAR2 Batteries Voltage [Volt]:2

PAR3 Status Charge [%]:96

Check

Alarm

Tolerance

OOL

BUS TM

Packet ID:TM_EPS_BUS

Subsystem:BUS

TIME:2019-09-23 14:23:46

PAR1 Bus 5v Voltage [Volt]:0.9375

PAR2 Bus 3.3v Voltage [Volt]:0.125

Check

Alarm

Tolerance

OOL

EPS is currently selected ---

ADCS Packet

EPS Packet

TCS Packet

PROPULSION Packet

QBC Packet

PAYLOAD Packet

Version Number: 0

Packet Type: 0

Secondary Header Flag: 1

Application Process Id: 1231

Segmentation Flags: 11

Sequence Control Count: 1

RW PKT

Version Number: 0

Packet Type: 0

Secondary Header Flag: 1

Application Process Id: 1232

Segmentation Flags: 11

Sequence Control Count: 10

THR PKT

Figure 81: Tc-001 Nominal TM packets

In this test session, as shown in Figure 81, all the packets are correctly generated, sent and received (the green list indicates the correct acquisition of the packets). The extraction of the packets parameters has happened correctly, and all the consistency checks return a positive result indicated by the green cells in all systems pages. The test is conducted on about 150 nominal packets generated to have a substantial number of data on which make statistical considerations.

As proof of validation it is possible to check that from the telecommand (TC) interface it is not useful to generate TC packets to change the parameters that are already correct (in the TC interface there are all green cells, so the operator does not change the parameters).

| ADCS | | EPS | | OBC | | PAYLOAD | |
|--|---------------------|-------|-------|-----------|-----|---------|--|
| CPU TM | | | | | | | |
| Packet ID: | TM_OBC_CPU | | | | | | |
| Subsystem: | CPU | | | | | | |
| TIME: | 2019-09-23 14:23:46 | | | | | | |
| PAR1 Operative Mode: | online | Check | Alarm | Tolerance | OOL | | |
| PAR2 Memory Storage [%]: | 47.875 | | | | | Acquire | |
| PAR3 Payload Bit Rate [bit per sec]: | 1720000.1875 | | | | | | |
| PAR4 AOCS Bit Rate [bit per sec]: | 7800448.59375 | | | | | | |
| PAR5 Housekeeping TM Bit Rate [bit per sec]: | 5523279.5625 | | | | | | |
| WD TM | | | | | | | |
| Packet ID: | TM_OBC_WATCHDOG | | | | | | |
| Subsystem: | Watchdog | | | | | | |
| TIME: | 2019-09-23 14:23:46 | | | | | | |
| PAR1 Operative Mode: | Online | Check | Alarm | Tolerance | OOL | | |
| PAR2 Memory Storage [%]: | 28.375 | | | | | Acquire | |

TC PKT GEN | Send

Acquire

TC PKT GEN | Send

Figure 82: Tc-001 Nominal TC interface

In conclusion, based on the tests carried out, and based on the data collected, it is possible to say that the Tc-001 reflects the expected results and verifies the relative requirements. Ultimately, the test has passed.

5.2.4 TEST SESSION: TM/TC ERROR PROFILE (Tc-002)

The second test case execute is the generation of a telemetry error profile. This test consists into verifying that the SAT SIM thread is able to generate telemetry in the not acceptable range (error range) and if it is able to packetize them and sent them to the GS SIM thread. In addition, the test has the purpose to establish if the GS SIM can recognize the packet, extract the useful data and convert them into an engineering language. In conclusion, if the interface is able to display the error TM generated, send error to the operator's mobile and if it is able to recognize the error profile, the test is considered as passed.

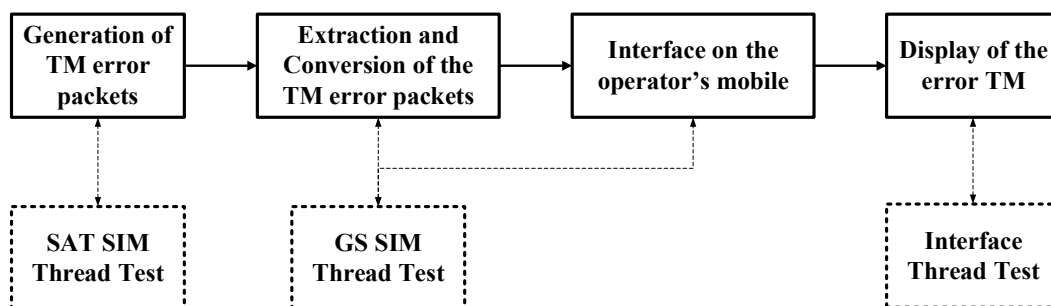


Figure 83: Tc-002 Error TM generation flowchart

The last step is to test the interface. From the interface it is possible to check if the previous two phases are ended correctly and if the test is passed.

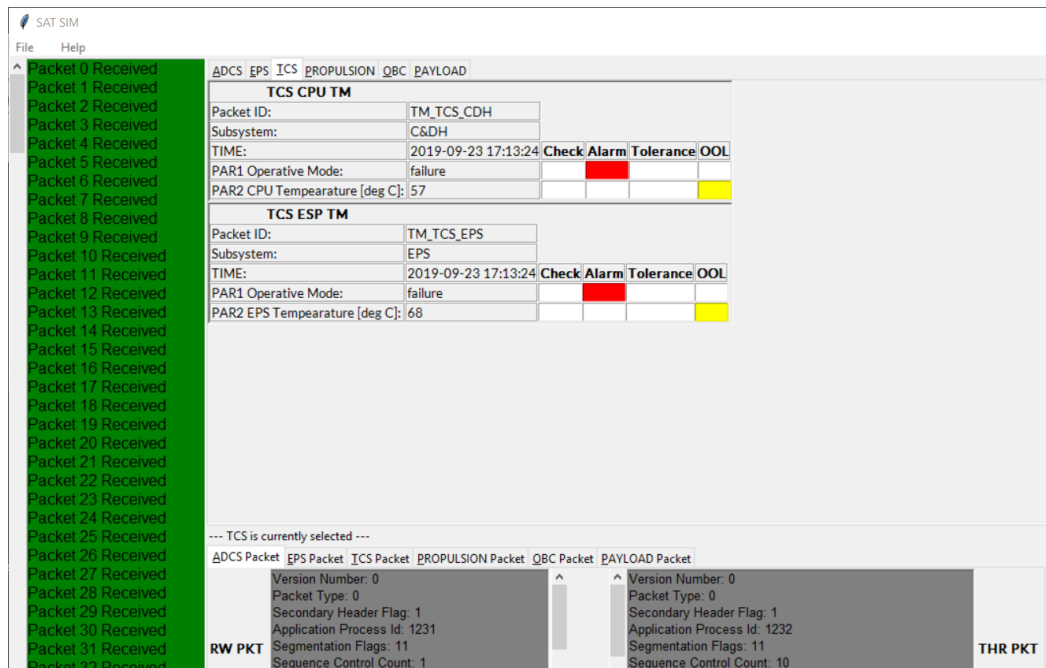


Figure 84: Tc-002 Error TM packets

In this test session, as shown in Figure 84, all the packets are correctly generated, sent and received (the green list indicates the correct acquisition of the packets). The extraction of the packets parameters has happened correctly, and all the consistency checks return a result indicated by the colors of cells in all systems pages. The test is conducted on about 150 nominal packets generated to have a substantial number of data on which make statistical considerations.

All the alert messages and the warning messages are correctly sent to the operator's mobile. From the smartphone the operator receives an alert notification as shown in Figure 83. This notification provides the general information about the type of packet and the type of message incoming to the GS SIM.

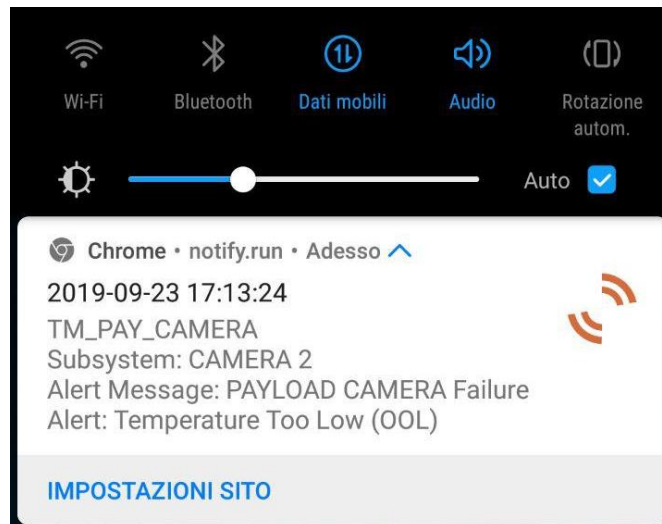


Figure 85: Alert Notification

The operator can click on the notifications and see the relative details of the incoming message, as shown in Figure 86.

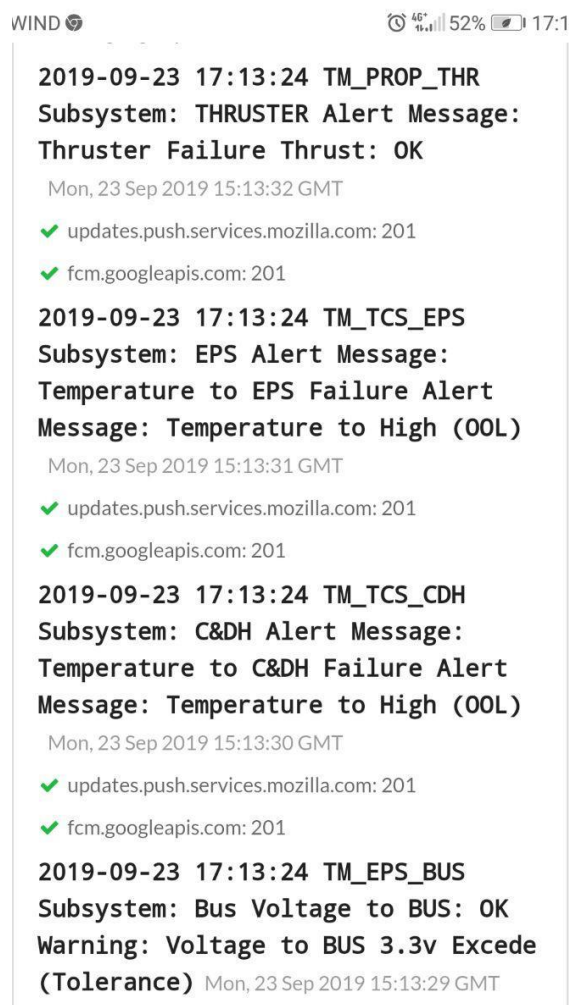


Figure 86: Detail of the incoming message

The Python library used to connect the operator's smartphone to the code is *notifyRun*. This library consent to connect a smartphone to a server and then through a line command send string as message directly to the registered smartphone.

This skill provides the operator the possibility to monitor the TM packets incoming into the ground station even being away from the control center. It is important to emphasize that the operator can only monitor the situation in revenue, for any action the presence within the control center is necessary.

To resolve the alert the operator must generate TC packets to correct the parameters through different protocols and acquire new telemetry.

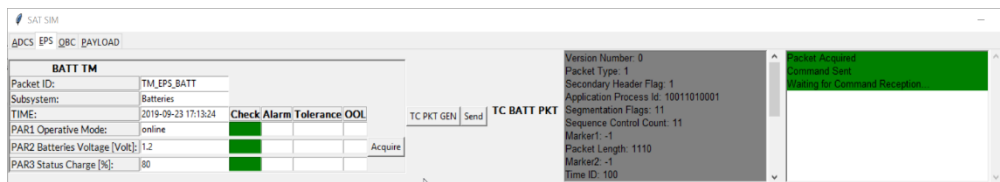


Figure 87: TC packets generation

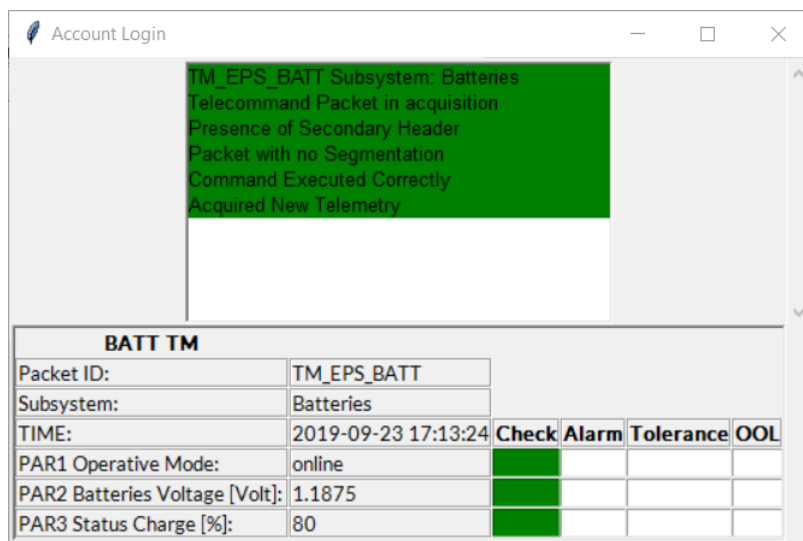


Figure 88: New Tm correction acquired

During the test, the software's ability to control the correct acquisition of packets was also tested. After several tests, it was possible to see that if the packet length exceeds the length indicated by the CCSDS standards (65536 octets relative to the data field), the packet is automatically discarded, and an error is shown on the interface indicating the number of the discarded packet.

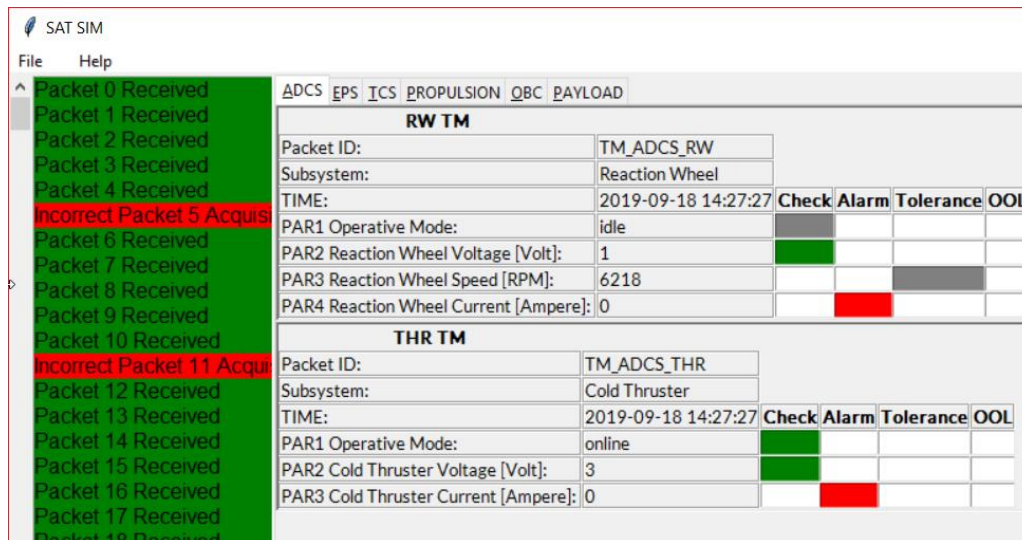


Figure 89: Incorrect Packet Acquisition

In conclusion, based on the tests carried out, and based on the data collected it is possible to say that the Tc-002 reflects the expected results and verifies the relative requirements. Ultimately, the test is passed.

5.2.5 TEST SESSION: IMAGE PROFILE (Tc-003)

The third test case execute is the management of the image profiles. This test consists into verify that the software is able to manage the images. In particular, the test controls that the software can take images from a dataset, compress them into images packets and send them to the GS SIM that extracts and converts the images and displays them to the operator. The test was conducted on about 24 images of Turin in different spectral bands [12]. In conclusion, if the interface is able to display the images, and their relative RGB diagram, and the software is able to save these images, the test is considered as passed.

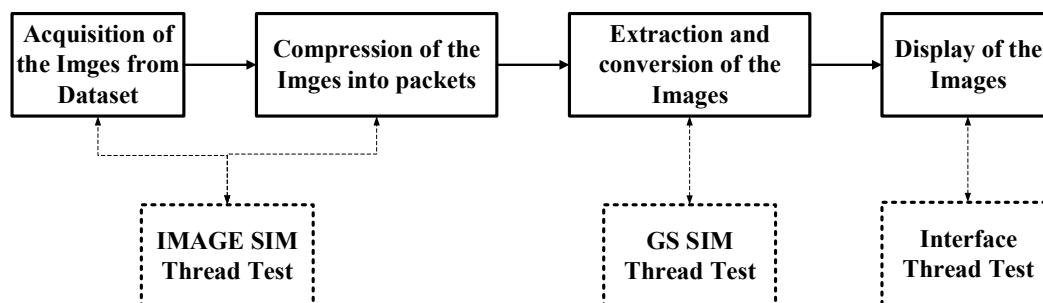


Figure 90: Tc-003 Image management flowchart

From the interface is possible to control all the phases of the test. The first phase is the acquisition of the image and the compression of the images. From the TC Viewer through the button *Acquire Image* is possible to require the compression of the images from the dataset into packets and sent them to the GS SIM for the extraction phase.

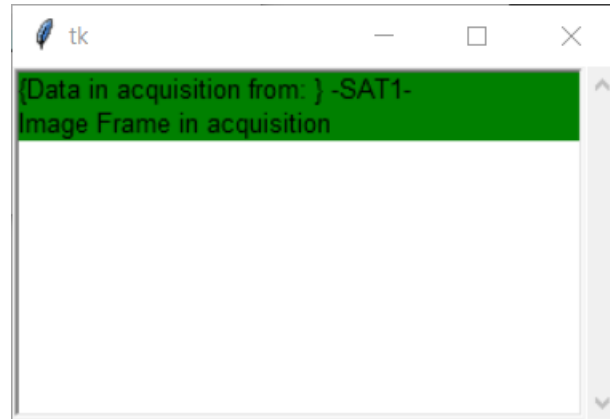


Figure 91: Tc-003 Image compression Interface

After the compression phase, the packets are sent to the GS SIM to be extracted and to display the images.

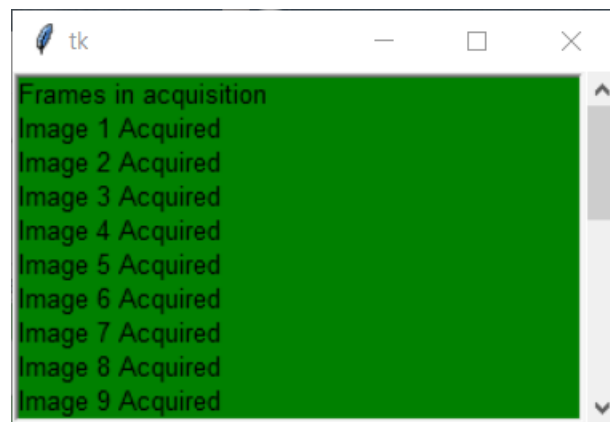


Figure 92: Tc-003 Frame Extraction

If the acquisition phase ends correctly, it is possible to display the acquired images and monitor the relative RGB graph.

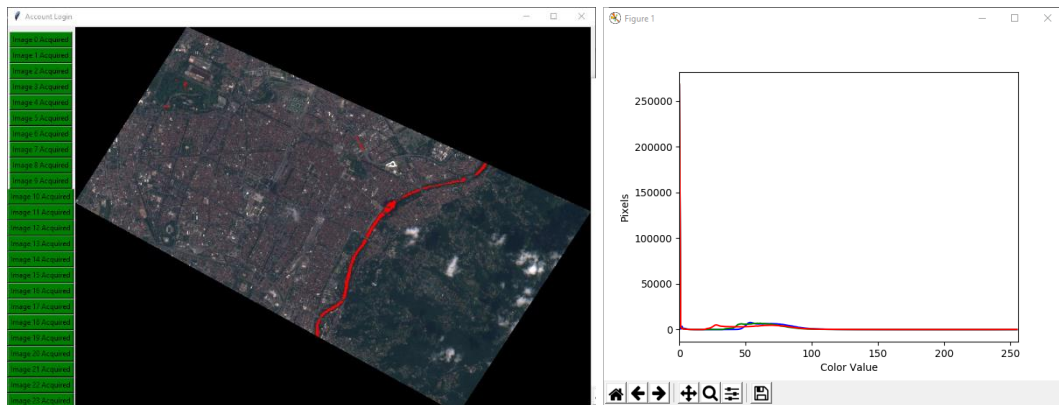


Figure 93: Tc-003 Images Display

From the *Image list* on the left of Figure 84, is possible to select which image and relative RGB graph the operator wants to display. As said in the section 5.2.4, if the length of the packets does not respect the recommended CCSDS length [10], the packet is automatically discarded, and an error is shown on the interface indicating the number of the discarded packet. During the test session, based on 24 images takes from the dataset, no error has occurred so the CCSDS compression expressed in the recommendations [10] is respected.

In conclusion, based on the tests carried out, and based on the data collected it is possible to say that the Tc-003 reflects the expected results and verifies the relative requirements. Ultimately, the test is passed.

5.2.6 TEST SESSION: SCHEDULER TEST (Tc-004)

The fourth test case execute is the test of the SAT SIM Scheduler. This test consists into verifying that the scheduler thread is able to recognize the pre-set command, add them to a scheduler queue and execute them according to the priority number associated by the operator. In particular, the test control the correct acquisition of the command structures, generates the schedule correctly and sends them to the SAT SIM thread that executes all the scheduled commands follow the priority number order.

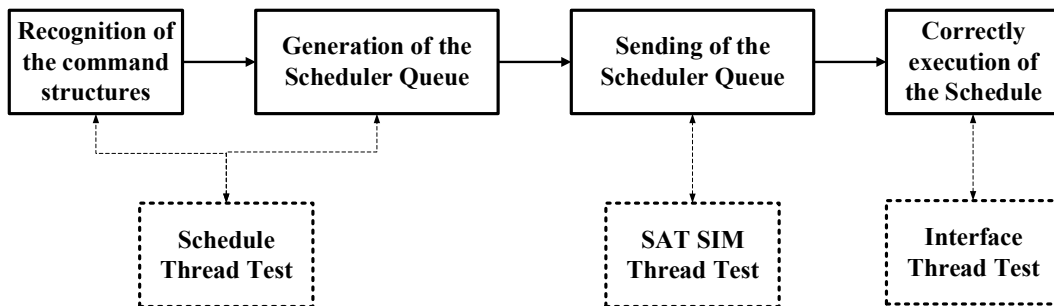


Figure 94: Tc-004 Schedule test flowchart

From the interface it is possible to control all the phases of the test. The first phase is the recognition of the command structure. It means that the interface could be able to display all the command and all the relative information about a specific command chosen by the operator.

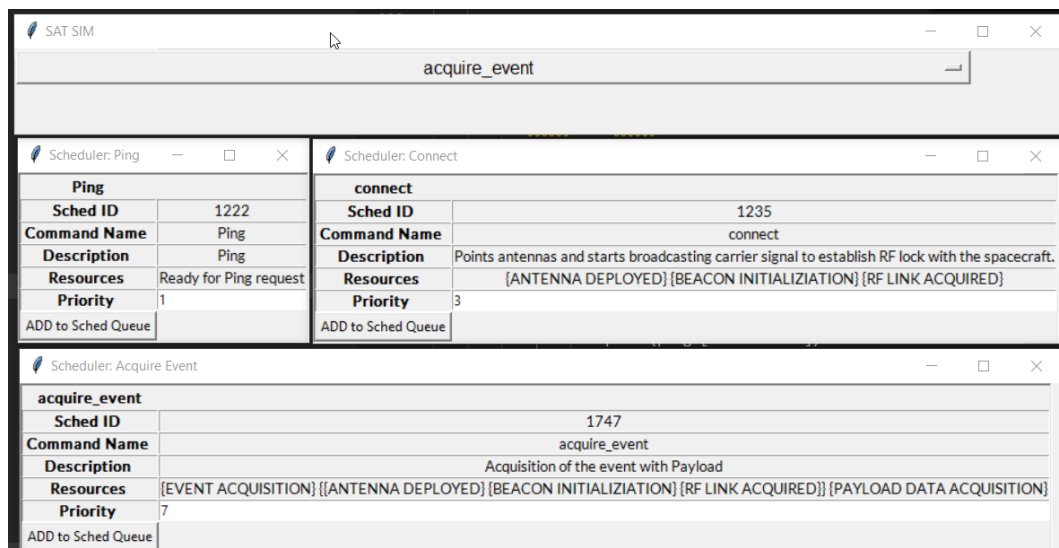


Figure 95: Tc-004 Recognition and display of the pre-set commands

As shown in Figure 95, each command window presents an unique ID, generated every time the operator wants to create a new schedule, for the addition into the scheduler queue, the name of the command, a brief description, the resources involved into command execution and the priority where the operator can set the priority number.

Through the command windows the operator can set the priority number to all the commands that he wants to schedule. To establish a correct priority number the theory of space operations and the operator experience could help to schedule correctly the commands.

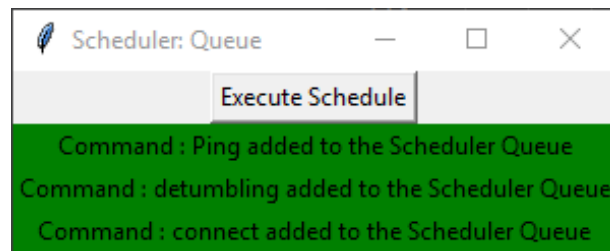


Figure 96: Tc-004 Generation of the Schedule queue

Through the *Execute Schedule* button the Schedule thread generates a queue sorted by priority number where the number 1 indicates the maximum priority and then, in sequence, the other number indicates a lower level o priority. This queue is sent to the SAT SIM thread that executes the command according to the priority number established by the operator.

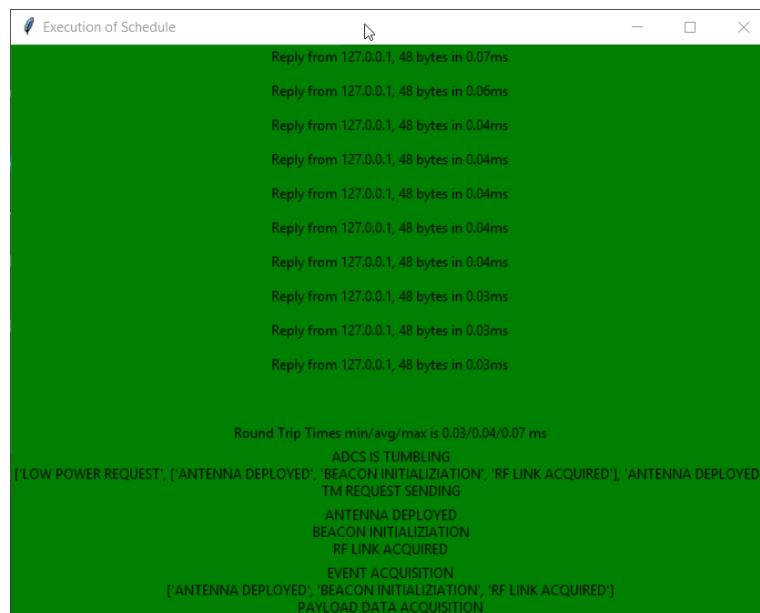


Figure 97: Tc-004 Schedule queue executed correctly

It is important to say that the operator cannot assign the same priority to different commands because it is not possible for the spacecraft to execute two different operations at the same time. For this reason, if the operator wants to assign the same priority to different commands, the scheduler interface provides him a warning message to alert the operator that this operation is incorrect, and the schedule is destroyed.

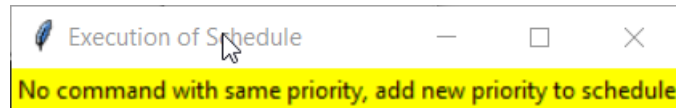


Figure 98: Tc-004 warning message for command with same priority number

In conclusion, based on the tests carried out, and based on the data collected it is possible to say that the Tc-004 reflects the expected results and verifies the relative requirements. Ultimately, the test is passed.

5.3 RESULTS

All the tests performed are tracked in a table where is possible to monitor the type of test case (Tc) performed, the requirements the test wants to verify, the expected results and the result of the test (passed or failed). The Table 11 shows the actual result of the test sessions.

| Test Case ID | Test Case Description | Input Data/Requirements | Expected Result | Pass/Fail |
|--------------|--|--|---|-----------|
| Tc-001 | Generation of the TM packet describing the nominal condition | REQ-1; REQ-2; REQ-3; REQ-10; REQ-12; REQ-13 | Positive end of the generation of nominal packets | P |
| Tc-002 | Generation of the TM packet describing the error condition and correct the error with TC packets | REQ-1; REQ-2; REQ-3; REQ-4; REQ-5; REQ-6; REQ-10; REQ-12; REQ-13 | Positive end of the generation of error packets | P |
| Tc-003 | Manage of the Images packets | REQ-7; REQ-8; REQ-9; REQ-10; REQ-11; REQ-13 | Positive end in managing images packets | P |
| Tc-004 | Test of the correct functions of the SAT SIM Scheduler | REQ-17; REQ-18; REQ-19; REQ-20; REQ-21 | Positive end in test the scheduler operations | P |
| Tc-005 | Inspection of the code to verify the design requirements | REQ-14; REQ-15; REQ-16 | \ | P |
| Tc-006 | Test of the Login interface | REQ-13 | Login interface that works only with the correct credentials registered | P |

Table 11: Test Cases Results

From Table 11 it is possible to see the test cases performed in the test sessions. The major tests are the test from Tc-001 to Tc-004 described in detail in the previous sections.

The Tc-005 is an inspection test to verify the design requirements like the implementation of the CCSDS standards. The test consists in inspecting the lines

of code and verifying that the software meets the design requirements expressed in Table 10. At the end of the inspection the requirements are verified and the Tc-005 is considered as passed.

The last test case, Tc-006 is a test to verify the initial interface of login. The idea of the SAT SIM software is to have a Python library that the operator can use on any computer that can handle the Python language. The operator can then use this library through his credentials and access the software and use the incoming telemetry data.

The test consists in verifying the correct registration of the operator and the correct access to the software with the registered credentials.

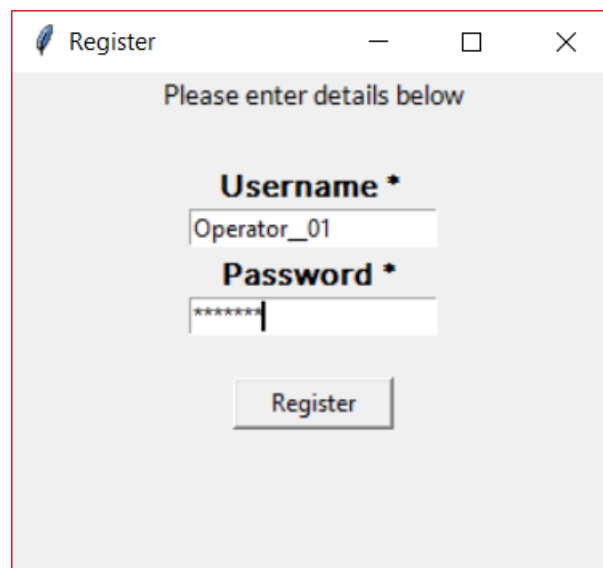
A screenshot of a web application window titled "Register". The window has a light gray background and a white border. At the top, there is a header bar with the title "Register" and standard window controls (minimize, maximize, close). Below the header, the text "Please enter details below" is displayed in a dark gray font. The main content area contains two input fields. The first field is labeled "Username *" and contains the text "Operator_01". The second field is labeled "Password *" and contains a series of asterisks "*****". Below the password field, there is a "Register" button with a gray background and white text. The entire form is centered on the page.

Figure 99: Tc-006 Register Interface

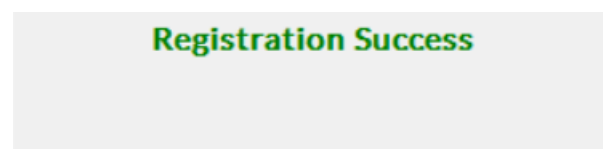


Figure 100: Tc-006 Registration Success

After the registration the operator can access whit his credentials to the SAT SIM software.

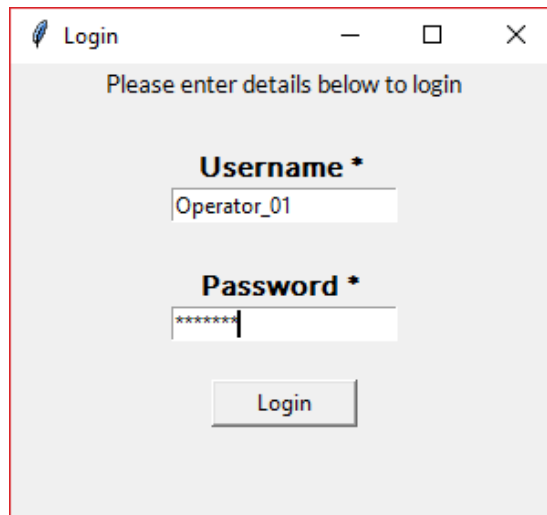


Figure 101: Tc-006 Login Interface

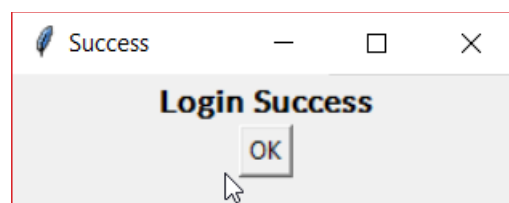


Figure 102: Tc-006 Login Success

In conclusion, based on the tests carried out, and based on the data collected it is possible to say that the Tc-006 reflects the expected results and verifies the relative requirements. Ultimately, the test has passed.

6. CONCLUSIONS

The completion of the project led to some reflections about the work that has been done.

Standards and recommendations are fundamental guidelines when a project is in its design phase. However, these references may require a very long study due to their complexity and the numerous volumes dedicated to a specific area. Therefore, the present work required, at beginning, a phase of organization research and study of the necessary references that led to the selection of the different macro-areas necessary to achieve the objectives of this thesis.

The aim of the present work is to provide to student and non-professional operators a software to manage, control and study space packets and protocols following the CCSDS standards. This thesis also has the purpose to provide a control software for the C3 project in which the students can support CubeSats operations and manage the entire ground station.

The first chapters of the thesis describe the context in which the software is collocated. From the space operations world, in which the SAT SIM software propose itself to train future spacecraft operators via CubeSat operations to achieve an important method to increase the effectiveness of future operations with already trained experts, to the C3 project, in which the SAT SIM software will be the core of the control centre.

The third chapter describes the standards used to support the generation of the code. Mainly the CCSDS recommendation are used to the construction of the TM, TC, IMG PKTs and for the operations scheduler philosophy.

The fourth chapter provides a complete overview of the SAT SIM library architecture and describes all the main functions of the software with particular focus on the data flow from a thread to another.

Last chapter is focused on all the test session performed to validate the software and verify all its specifications and requirements. As said in the results section 5.3, all the tests are passed, and the software is verified and validated.

It is important to say that even if the software is validated and verified, it requires some future works to be completely integrate in the C3 control centre.

Some future steps identified for the next upgrade of the SAT SIM library are:

- Integration of the software on different hardware. The first step identified for the future is the test of communication between two hardware. This test will need the implementation of the SAT SIM thread and GS thread on different boards so to test the generation of packets from the SAT SIM board, sending and extracting of the packets from GS board.
- Automatization of specific procedures. This point will require the study of automatic algorithms in order to automatize the command and schedule procedures.
- Implementation of different missions and CubeSat architecture. In this step will be upgrade the software to support multiple CubeSat missions and operations in order to create a substantial database with mission information and CubeSat architecture structures able to support the software and the entire ground station.
- Integration of the SAT SIM software in the full control centre of C3 in order to integrate also the RF software and the Tracking software.

In conclusion, this work of thesis hopes to provide a useful starting point to support the future implementations of the control centre software to support the C3 project and the future CubeSat operations.

REFERENCES

- [1] ECSS, “ECSS-E-ST-70-41C – Telemetry and telecommand packet utilization,” *Ecscs-E-St-70-41C*, no. April 2016.
- [2] J. J. James, R. R. Wertz, and W. J. Larson, Wertz Mission Geometry; Orbit and Constellation Design and Management, James R. Wertz Influence of Psychological Factors on Product Development. 1999.
- [3] ECSS, “ECSS-E-ST-10C – Space engineering requirements,” *Interface*, no. March 2009.
- [4] ECSS-M-ST-80C, “Space project management Risk management,” no. July, p. 43, 2008.
- [5] CCSDS 102.0-B5, “Packet Telemetry,” no. November, pp. 2000–2000, 2000.
- [6] CCSDS 301.0-B-4, “TIME CODE FORMATS Recommendation for Space Data System Standards,” no. November 2010.
- [7] CCSDS 201.0-B-3, “Telecommand Part 1—Channel Service.” no. June 2000
- [8] CCSDS 202.0-B-2, “Telecommand Part 2—Data Routing Service.” no. November 1992.
- [9] CCSDS 203.0-B-2, “Telecommand Part 3—Data Management Service.” no. June 2001.
- [10] CCSDS 122.0-B-2, “IMAGE DATA COMPRESSION Recommendation for Space Data System Standards 122.0-B-2,” no. September 2017.
- [11] L. Wall “About the Tutorial Copyright & Disclaimer,” p. 2, 2015.
- [12] Sentinel Hub refer link: <https://apps.sentinel-hub.com/eo-browser>
- [13] “Learning Tkinter,” *Stack Overflow contributors*, 1966.
- [14] J. Jacobson and B. Johansson, “MID Software Work Package 2 Methods for Validation and Testing of Software,” no. September 2004.
- [15] CCSDS 529.0-G-1, “Mission Planning and Scheduling,” no. June 2018.

APPENDIX

The profiling graph of the code is generated by a Python library called pycallgraph. Pycallgraph is a Python module that creates call graph visualizations for Python. It uses a debugging Python function called `sys.set_trace()` which makes a callback every time the code enters or leaves a function. This consents to Python to track the name of every function called, as well as which function called which, the time taken within each function, number of calls, etc.

In the figures below the profiling of the SAT SIM code and the Interface is shown. The description of the profiling sessions is described in the section 5.2.2.

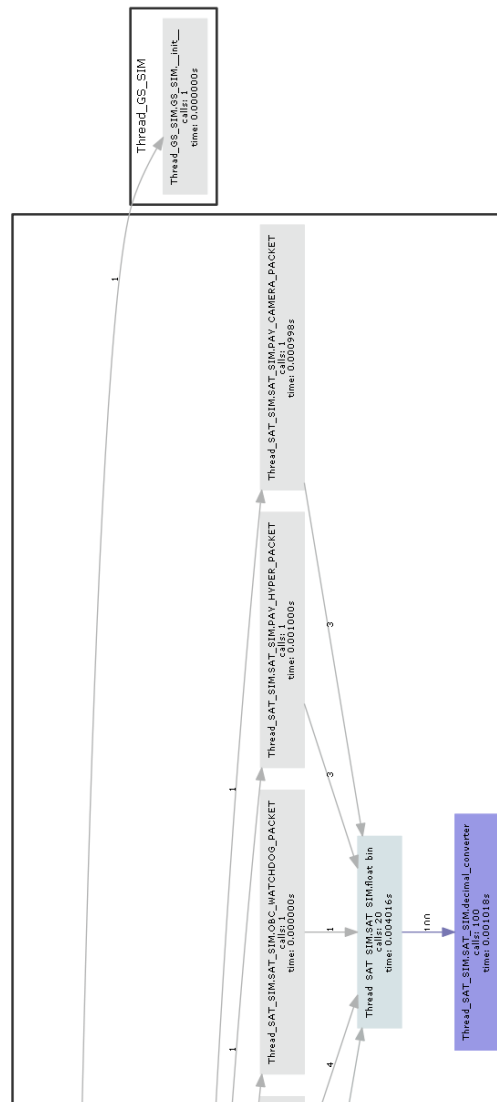


Figure 103: SAT SIM Call Graph (1)

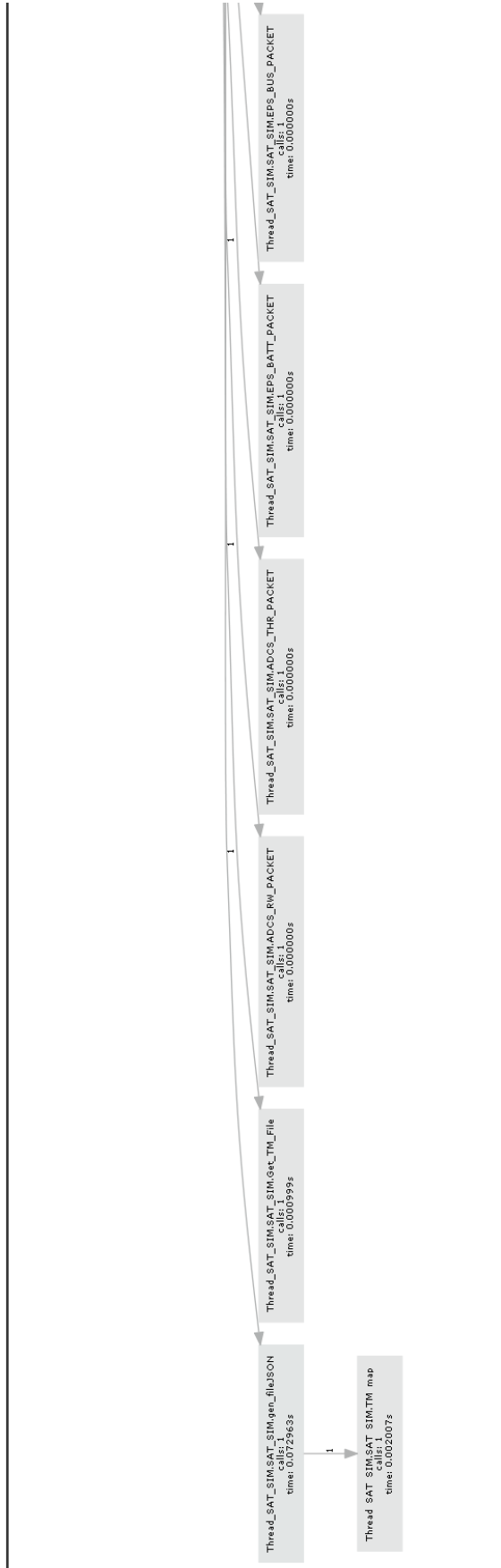
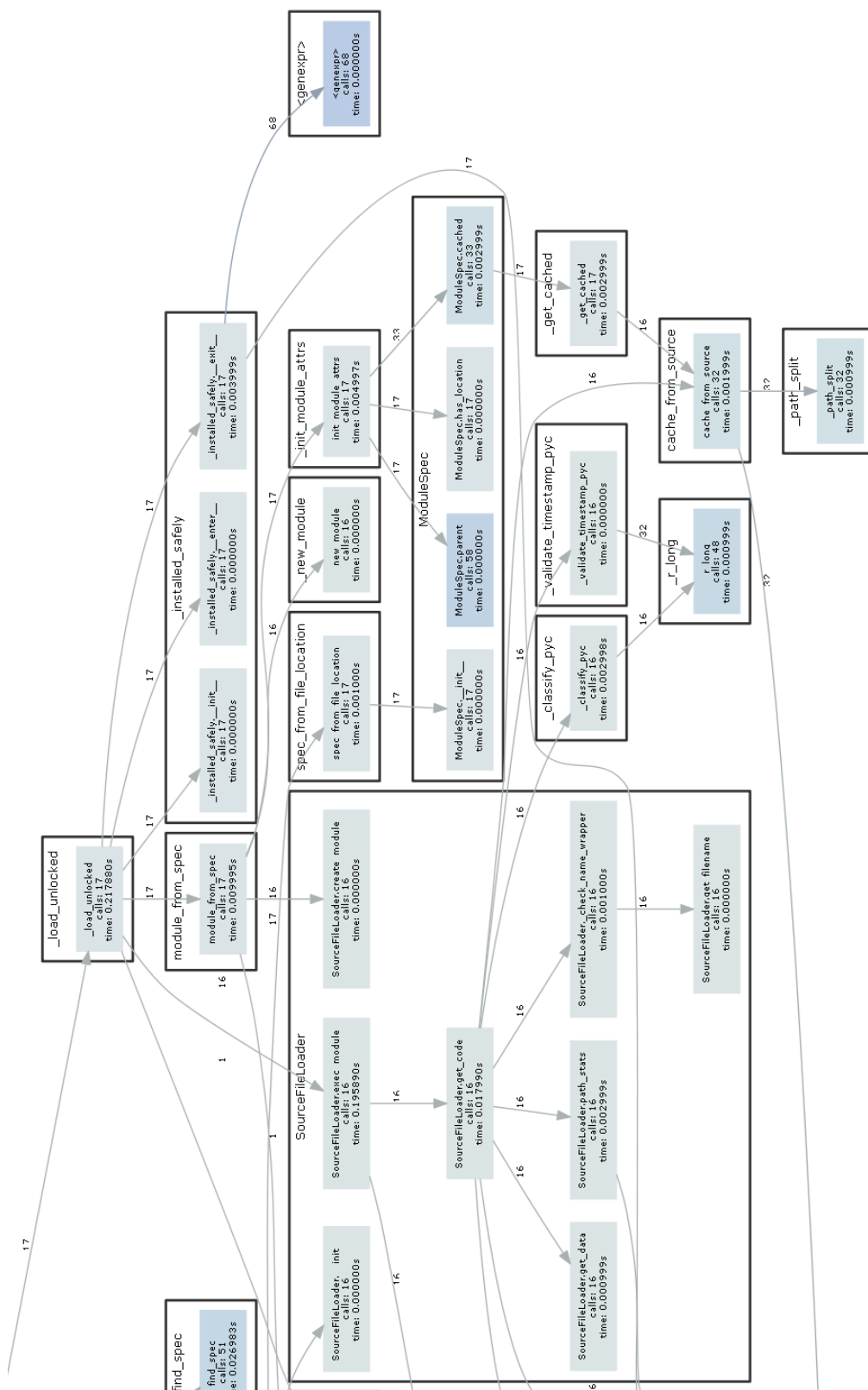
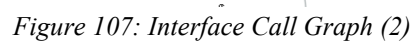
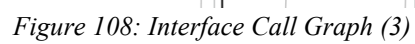


Figure 105. SAT SIM Call Graph (3)









113

Figure 110: Interface Call Graph (5)

