Politecnico di Torino

Politecnico di torino

Department of Control and Computer Engineering
Master's Degree Thesis
A.Y 2020/21

Design and Development of a Ground Segment Software for Microsatellite Operations

Supervisors
Luca Sterpone
Biagio Cotugno

Candidate
Simone Gallo

A nonna Nena,
che ha sempre creduto in me

Abstract

In recent years outer space is increasingly becoming more accessible, as the costs necessary to launch objects in low orbit or even beyond is significantly lower than ever before. One of the factors that contributed to this trend is the development of microsatellites and particularly CubeSats.

CubeSats are becoming more and more prevalent in the space industry as they can be used for primary missions thanks to the improvement of technology which allows for the implementation of many functionalities in smaller volumes, but most significantly they can act as support satellites for bigger missions resulting in relatively easy to carry and deploy. One of the main examples is the Double Asteroid Redirection Test (DART) mission that will demonstrate the kinetic effects of crashing an impactor spacecraft on the surface of an asteroid for planetary defence purposes. The DART spacecraft will also carry a secondary spacecraft called LICIACube, a 6U (unit) satellite that will separate from the main spacecraft 10 days before the impact to acquire images of the collision and ejecta plume as it fly-bys past the asteroid. Despite being relatively small concerning regular satellites the ground segment required to send commands and receive telemetry for the CubeSats is the same.

The scope of this master's thesis is to develop a ground segment software for Argotec's *Mission Control Center* (MCC). Although Argotec already uses a main mission control software, it has been decided to implement a tailored software devoted to telemetry retrieval and display to include new functionalities not present in the main MCS. A preliminary study of the telemetry structure and network architecture of the company have been conducted by the author of this thesis together with the *Flight Control Team* (FCT) of Argotec. Then the software requirements have been identified taking into account possible use cases of the MCS and the possibility of working seamlessly with other company's software like the *Mission Planning Tool* (MPT). Subsequently, a security study has been carried out to guarantee user authentication and data access policies depending on the user clearance. In the end, an extensive campaign of system validation tests has been performed. The project definition, the analysis of requirements, the system development and testing are here presented, together with a final assessment for future improvements. Furthermore, in this document, the Ground Segment Software will also be referred to as Margot (**M**ultianalysis **A**nd **R**eal-time **G**round **O**perations **T**ool).

List of Figures

List of Tables

Listings

Summary

1. Introduction

At the current state, Argotec uses a *Mission Control Software* (MCS) to connect to the NASA Deep Space Network.
The amount of operations performed during real-time communications could strain the system robustness and usability. For this reason, the necessity to relieve the system from some of its duties was an important manner to resolve. In order to do that, functional areas have been identified inside the MCS. In fact, during a communication window, the MCS needs to:

- Receive data (such as telemetries);
- Send commands;
- Visualize display monitors;
- Send data to the MCS clients used by other Operations Team engineers.

The first two can't be substituted as they're validated and certified to work with the NASA network and the last point does not add much impact on performance. For these reasons, the necessity to delegate the visualization of telemetry and plots to another application has been identified. Besides, the necessity to have a tool that can be used by other company's clients inhibiting telecommands management, controlling at the same time what data is visualized and for how much time, is compliant with the first necessity and can be both implemented together in a single project.
To accomplish these objectives the need to retrieve data from MCS and visualize it employing a graphic framework arises. For either of the two macro operations a product analysis has been performed to identify the possible solutions:

Retrieving Data must be custom implemented as there is no market-ready solution and MCS doesn't have a native function of this kind;

Visualize Data : for this task, there are many solutions, either with pros and cons:

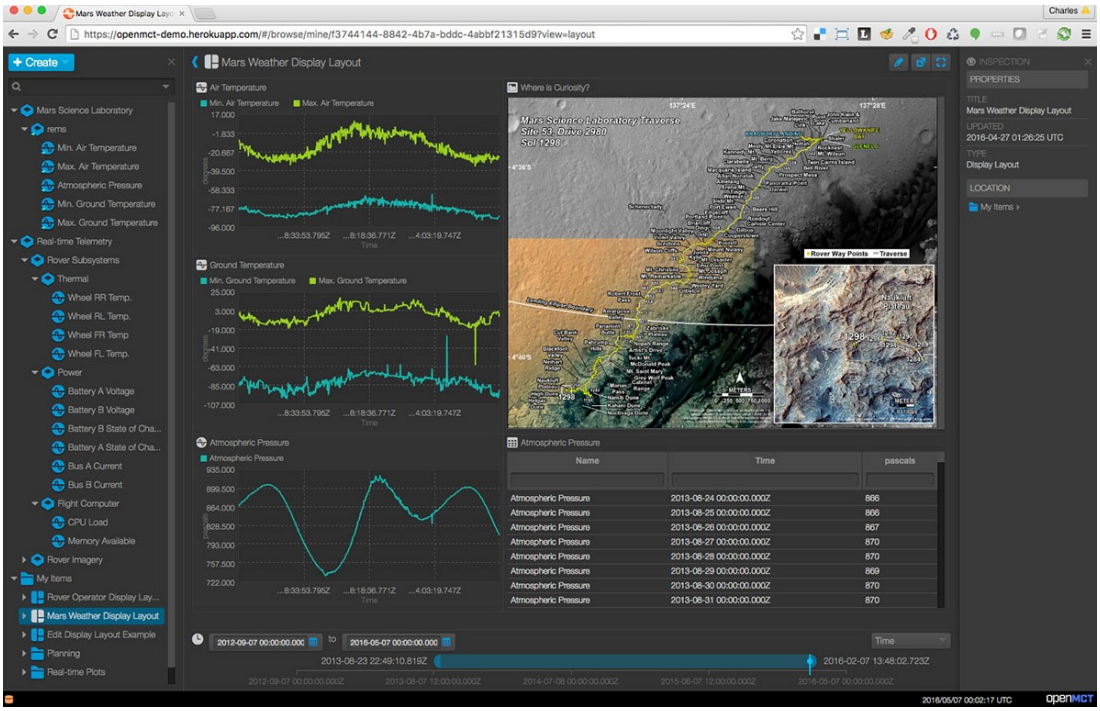
- *Custom-made solution* using graphical libraries like QT, Tkinter or OpenGL for desktop implementations or CSS for web implementations. This solution is highly customizable and certainly has the potential to accomplish all requirements, but it is more complex to implement and optimize in the short terms.
- *Matlab* is a proprietary multi-paradigm programming language and numeric computing environment developed by MathWorks. It supports developing *Graphical User Interface* (GUI) applications. GUIs can be generated either programmatically or using visual design environments such as *GUIDE* and *App Designer*. Moreover, MATLAB has tightly integrated graph plotting features [11].

The main disadvantage of this solution is its proprietary commercial license which could heavily impact the project budget and limit the on number of licenses that can be used.

- *OpenMCT* is a next-generation mission control framework for the visualization of data on desktop and mobile devices. It is developed at NASA's Ames Research Center and is being used by NASA for data analysis of spacecraft missions, as well as planning and operations [13].

  OpenMCT allows to have a good starting point in terms of functionalities with less code written than previous solutions, also the graphical interface is modern and reflects the company's views as can be seen in figure 1.1. One of the most important advantages of this solution is that the code is completely open-source and so fully customizable. The framework also permits the implementation of new features and in-house functionalities by the development of new plug-ins. The main disadvantage is represented by the necessity of following NASA's original development workflow which results are really tricky.

After a trade-off study, OpenMCT has been selected as the right solution for this project (*Table 1*).



Source: https://github.com/nasa/openmct/blob/master/README.md

Figure 1.1: OpenMCT example display

	Custom-made	Matlab	OpenMCT
Customization	High customization capabilities, the choice of the programming language can influence how the features are implemented	High customization capabilities	Limited customization capabilities out of the box, new functions are implemented through plugins
Plotting Features	Plotting features depend on the kind of implementation and/or framework used. In general, there is high plotting capability	High Plotting capability following Matlab language potential but it is not easy to set up and the possibility to navigate inside a graph is limited	High plotting capability with easy implementation. Functionality of navigating through time inside the graph is built-in.
Implementation Time	Really long, every aspect of the program must be developed, also the concurrency policies and resource management should be defined.	Fast implementation of plots, for the user interface the implementation time is longer	Really fast implementation, the graphical interface and plots are ready to be connected to a data provider.
Platform Independence	Depending on the platform, either way need a platform dedicated compiler/interpreter.	Needs a valid MatLab license on each machine	Runs on a browser, potentially can run on every machine (even mobile devices).

Table 1: Client implementation trade-off

2. State Of the Art

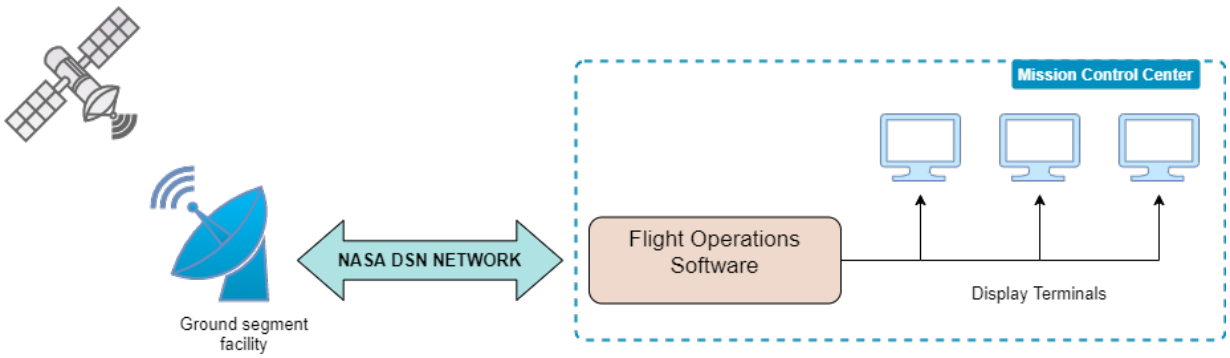


Figure 2.1: Current Ground system architecture

Before the completion of this master's thesis, the ground segment architecture was composed mainly by three components:

The Ground Segment facility represented by JPL/NASA's *Deep Space Network* (DSN). It is a worldwide network of American spacecraft communication ground segment facilities, located in Goldstone (United States, California), Madrid (Spain), and Canberra (Australia). The antennas at all three DSN complexes communicate directly with the *Deep Space Operations Center* (DSOC) located at the JPL facilities in Pasadena, California. The DSN supports NASA's and other selected partner's interplanetary spacecraft missions as well as some selected Earth-orbiting missions. It can also performs radio and radar astronomy observations for the exploration of the Solar System and the universe.
All DSN antennas are steerable, high gain, parabolic reflector antennas and provide a two-way communications link [3].

Flight Operation Software (FOS) is the main interface that the operations team has when communicating with the satellite. FOS is based on *Satellite Control and Operation System 2000* (SCOS-2000) and is interfaced with the DSN by means of the *Internet Space Link Extension* (SLE). The SLE identifies a set of Transfer Services that enables missions to send forward space link data units to a spacecraft and to receive return space link data units from a spacecraft [16]. The SLE standard specifies:

- the operations necessary to provide the Transfer Service;
- the parameter data associated with each operation;
- the behaviors that result from the invocation of each operation;
- the relationship between, and the valid sequence of, the operations and resulting behaviors.

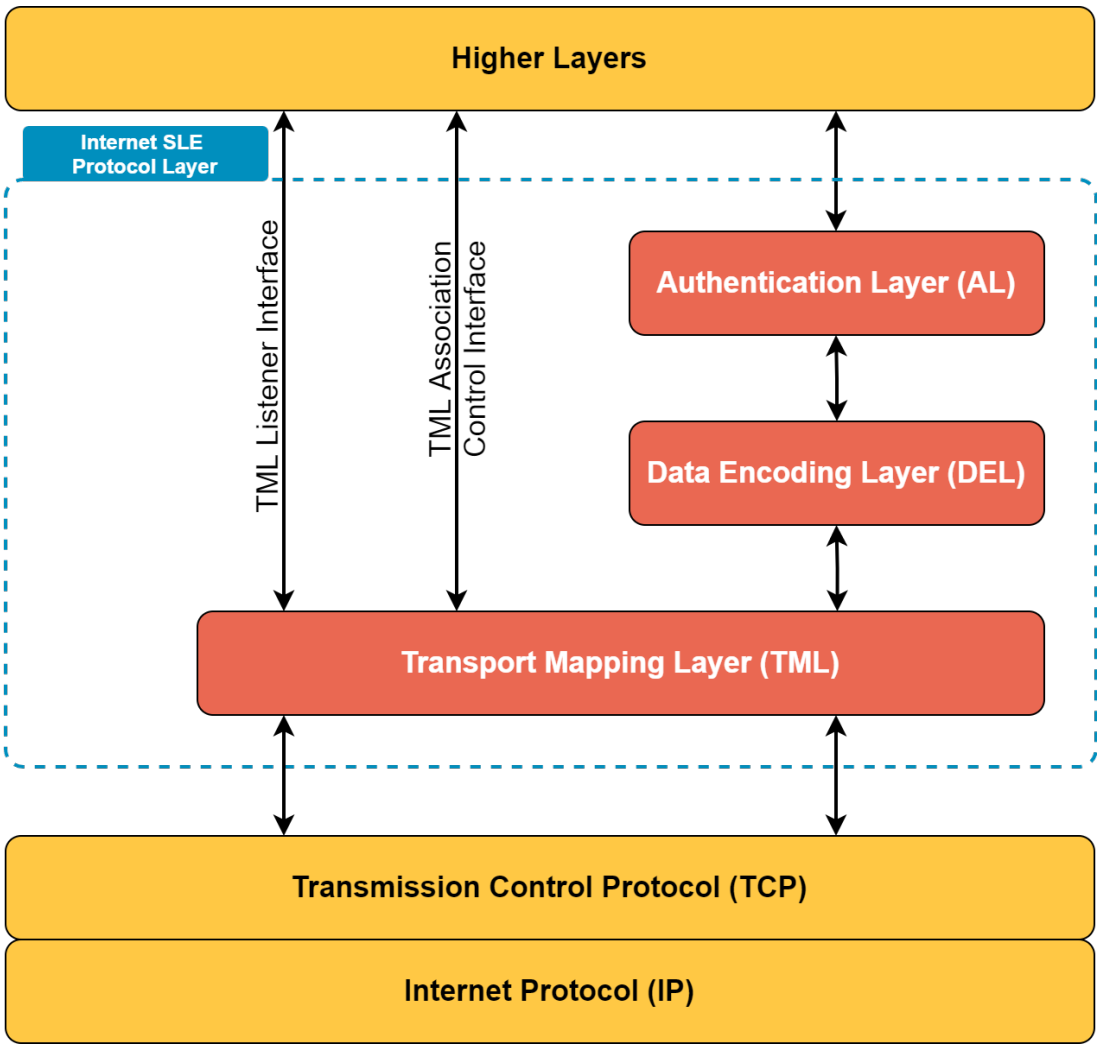


Figure 2.2: SLE Architecture Model

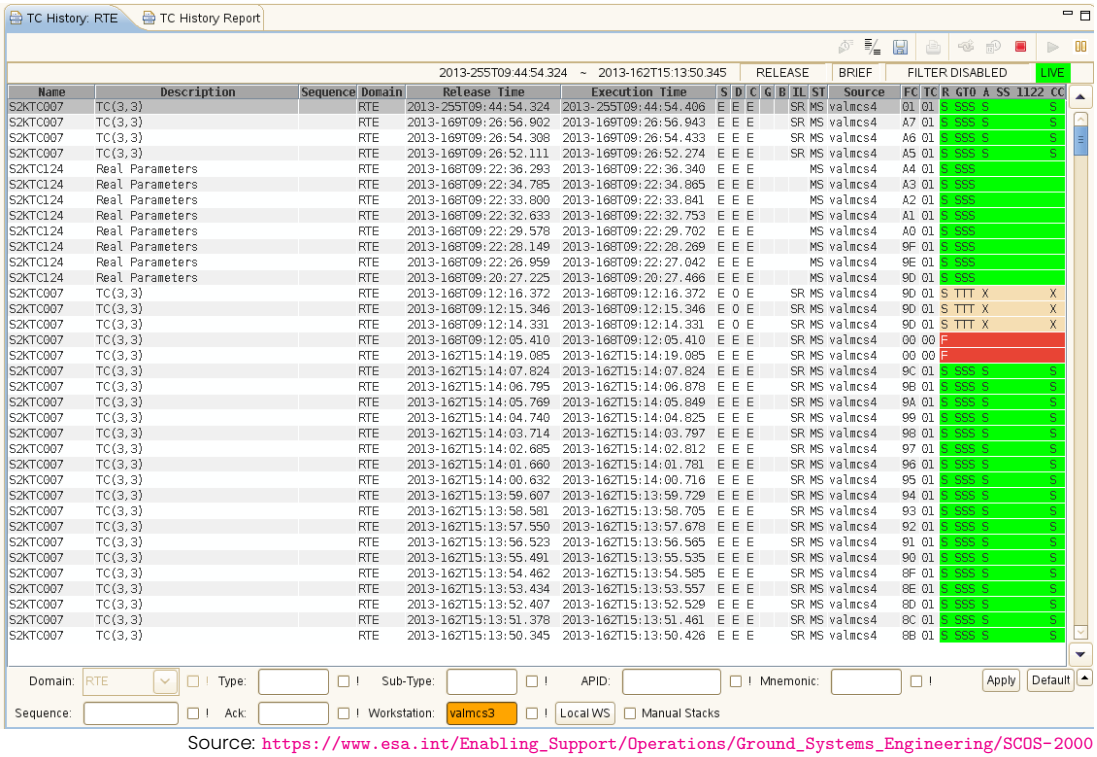
The Internet SLE Protocol is specified by a layered architecture model in which the interfaces between the layers are defined using abstract service primitives, roughly following the concepts in the OSI Basic Reference Model [9]. As can be seen in figure 2.2, the SLE protocol can be further decomposed into the following sub-layers:

- The *Authentication Layer* (AL) is responsible for generating and analyzing the credentials specified in the Recommended Standards for SLE transfer services. For this purpose, this Recommended Standard specifies use of the simple authentication scheme using public-key and attribute certificate frameworks, and the Secure Hash Function (SHA-256);
- The *Data Encoding Layer* (DEL) is responsible for the encoding of the SLE protocol data units received from higher layers and the decoding of the protocol data units received from the peer application;
- The *Transport Mapping Layer* (TML) handles the interface to the Transmission Control Protocol (TCP).

Using the SLE protocol, the FOS can receive Telemetry (TM) from DSN and send Telecommands (TC) using the transport layer. In order to decode receiving packets and encode outgoing commands, the FOS relies on the *Management Information Base* (MIB) database which, based on the CCSDS standards, specifies:

- type and sub-type of each packet;
- name of each telemetry and command items;
- the packet each item belongs to;
- position inside the packet (bit/byte offset);
- format of each item;
- polynomial coefficients of the function used to derive the engineering value from a raw value;
- other information depending on the mission scenario.

The MIB is also used by *Display Terminals* (DP) to visualize data to the user.
Furthermore, during real-time operations, the FOS stores the communication session data in a database which can be accessed later in order to generate replays.



Source: https://www.esa.int/Enabling_Support/Operations/Ground_Systems_Engineering/SCOS-2000

Figure 2.3: TC Display

Display Terminals are devices (usually PCs or laptops) where the user can monitor everything he/she needs to check the status of the spacecraft. They can include graphics, plots, tables, indicators, warning labels or numeric dump data and can emit sounds in case values are

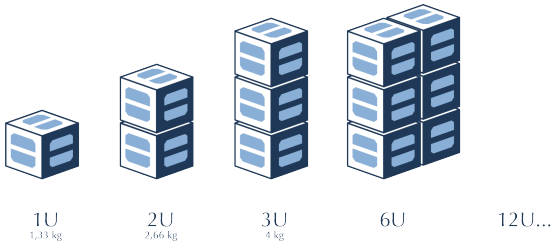
exceeding limits indicated in the MIB.

Display Terminals are usually used by all components of the Flight Operations Team in the Front Room as well as in the Back Room and they are really useful as they allow the operator to perceive really fast if the behavior of the spacecraft is nominal or there is something odd which has to be verified. These displays can also show non real-time data allowing the possibility to replay previous communication sessions and test sessions for troubleshooting.

2.1. Some Context: What is a Cubesat

CubeSat is a type of nanosatellites composed by multiple cubic modules of 10 cm x 10 cm x 10 cm in size and can have a mass of 1.33 kilograms per unit. Typical configurations are 1U, 3U, 6U, and 12U (*figure 2.4*). It's miniaturization allows reducing the costs of development and deployment as they are often suitable for launch in multiples, using the excess capacity of larger launch vehicles. These characteristics allowed many scientific organizations, universities and countries to send to space their first ever satellite.

Source: https://tinyurl.com/2p8je9c2



Source: https://tinyurl.com/5m4fc44e

Figure 2.4: Example of CubeSat systems

Despite being smaller with respect to typical satellites, a CubeSat shares the same subsystem of typical larger satellites:

Structure which consists of the frame. It is usually made of aluminum alloys.

CubeSats structures do not have all the same strength concerns as larger satellites do, as they can provide on the benefit of the deployer supporting them structurally during launch. Still, some CubeSats will undergo vibration and structural analyses.

Computing CubeSats often feature multiple computers handling different tasks in parallel including the attitude control (orientation), power management, payload operation and primary control tasks. For very Low Earth Orbit (LEO) in which atmospheric reentry would occur in just days or weeks, radiation can largely be ignored, and standard grade electronics may be used. Consumer electronic devices can survive LEO radiation for that time as the chance of a

single event upset (SEU) is very low. Spacecraft orbiting in LEOs, lasting months or years, are subjected to radiation effects and risks and only certified or flight hardware designed and tested for radiation environment are selected. Missions beyond LEO or which would remain in LEO for many years must use radiation-hardened devices.

Attitude Control is necessary in order to maintain or change the orientation to a desired value. Systems that perform attitude determination and control include reaction wheels, magnetorquers, thrusters, star trackers, Sun sensors etc. Combination of these systems are typically seen in order to take each method's advantages and mitigate their shortcomings.

Propulsion The biggest challenge with CubeSat propulsion is preventing risk to the launch vehicle and its primary payload while still providing significant capability. Components and methods that are commonly used in lager satellites are disallowed or limited. These restriction pose great challenges for CubeSat propulsion systems. Most used technologies for propulsion are cold gas, chemical propulsion, electric propulsion and solar sails.

Power CubeSats use solar cells to convert solar power to electricity that is stored in rechargeable lithium-ion batteries (mainly) that provide the CubeSat with the required power during eclipses as well as during peak load times. CubeSats have a limited surface area on their external walls for solar cells assembly and has to be effectively shared with other parts, such as antennas, optical sensors, camera lens, propulsion systems and access ports.

Telecommunications are a challenge on CubeSat because of tumbling and low power range. Many of them use an omnidirectional monopole or dipole antennas. For more demanding needs, high gain antennas are available but their deployment and pointing systems are significantly more complex. For LEO Ultra-High Frequency (UHF) ad S-band frequencies are used instead, to venture farther in the Solar System, larger antenna compatible with the Deep Space Network (X-band and Ka-band) are required.

Thermal management is really important as different components possess different acceptable temperature ranges as they're heated by radiative heat emitted by the Sun directly and reflected off Earth, as well as heat generated by the spacecraft's components. Components used to ensure the temperature requirements are met in CubeSats include multi-layer insulation and heater for the battery.

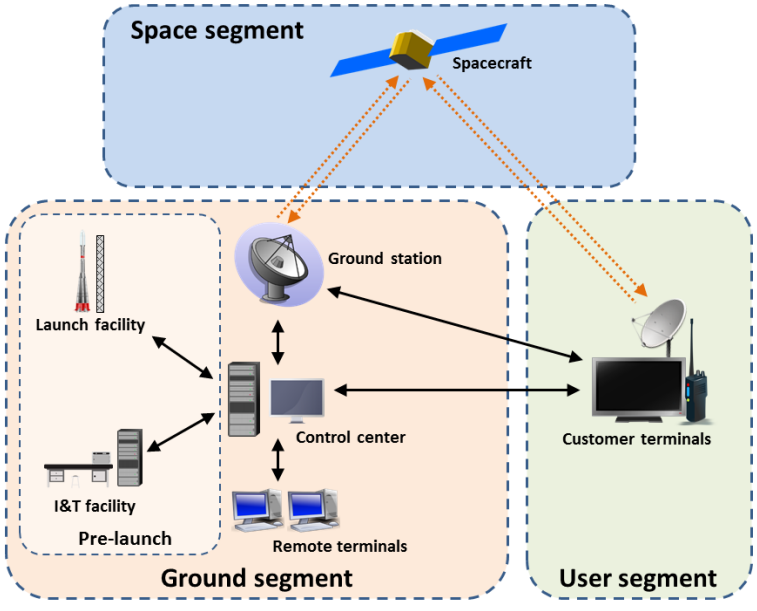
Payload is the object or the entity which is being carried by the CubeSat. Depending on the nature of the flight or mission, the payload may include cameras and several types of sensors. Most CubeSats carry one or two scientific instruments as their primary payload. There could be secondary payloads used in order to achieve secondary objectives of a mission.

CubeSat form a cost-effective independent means of getting a payload to orbit. The cost varies depending of the complexity of the project from few thousands to millions dollars. For these reasons, CubeSats are a viable option for some schools and universities as well as small businesses to develop a microsatellite for commercial purposes.

2.2. Some Context: Mission Control Center

The Mission Control Center (MCC) is part of the ground segment that consists of all ground-based elements of a spacecraft system used by operators and support personnel. The primary elements of a ground segment are:

- Ground stations which provide radio interfaces with the spacecraft.
- Mission Control Center from which spacecrafts are managed.
- Ground Networks which connect ground elements to one another.
- Remote Terminals used by support personnel.
- Spacecraft integration and test facilities which involve the overall testing of a complete system composed of many subsystem components or elements.
- Launch facilities which are used for launching (or retrieving) spacecrafts.



Source: https://tinyurl.com/yahjaef3

Figure 2.5: Simplified spacecraft system

In particular, the program described here will be used within the Mission Control Center (MCC). Sometimes called Flight Control Center or Operations Center, the MCC is a facility that manages space flights, usually from the point of launch until the end of the mission. A staff of flight controllers

and other personnel monitor all aspects of the mission using telemetry, and send commands to the spacecraft using ground stations (the operations team).
Personnel supporting the mission from an MCC can include representatives of the attitude control system, power, propulsion, thermal, attitude dynamics, orbital operations and other subsystem disciplines. The main activities performed by the MCC are:

- Monitoring and control of the spacecraft subsystems and payloads.
- Spacecraft performance analysis and reporting.
- Planning, scheduling, and executions of the procedure.
- Delivery of mission data product.

Therefore, mission operations contain different activities, linked by each other and often executed by different actors during the mission. It is possible to divide different phases of the operation procedures of a spacecraft:

Mission Operation Preparation *comprises all measures related to management, development, test integration, validation, organization, training, certification, and documentation of the ground segment of a space project. The result of successful mission operations preparation is a ready-for-launch ground segment.* [19]
Space missions are characterized by being different from each other, often what is designed and developed for one mission can be completely useless for another. As a result, the definition of a general rule for the duration of the operations preparation is nearly impossible.

Mission Operation Execution After the preparation, the test and launch of the spacecraft, the mission operations will be mainly focused on the flight operation involving the space segment. Satellites are usually designed to be as much autonomous as possible. One of the major drivers to involve human Flight Control Teams in the operations is the handling of unexpected situations (called anomalies or contingencies) in the ground or in the space segment. It is very important to have a reliable tools in order to analyze the telemetry and recognize dangerous scenarios for the good success of the mission. Here, humans need to be involved to handle the sometimes very complex situations and to put together either troubleshooting plans to identify the root cause of the problem, or to resolve the issue via corrective actions.

2.3. Some Context: The Flight Control Team

The operations team, also called Flight Control Team (FCT), is the staff of flight controllers working in the Mission Control Center. Each controller is an expert in a specific area and constantly communicates with additional experts in the Back Room. The preparation for the operations of a space

project requires the organization and assignation of roles and responsibilities as can be seen in *figure 2.6*. The Argotec's mission project structure is here presented:

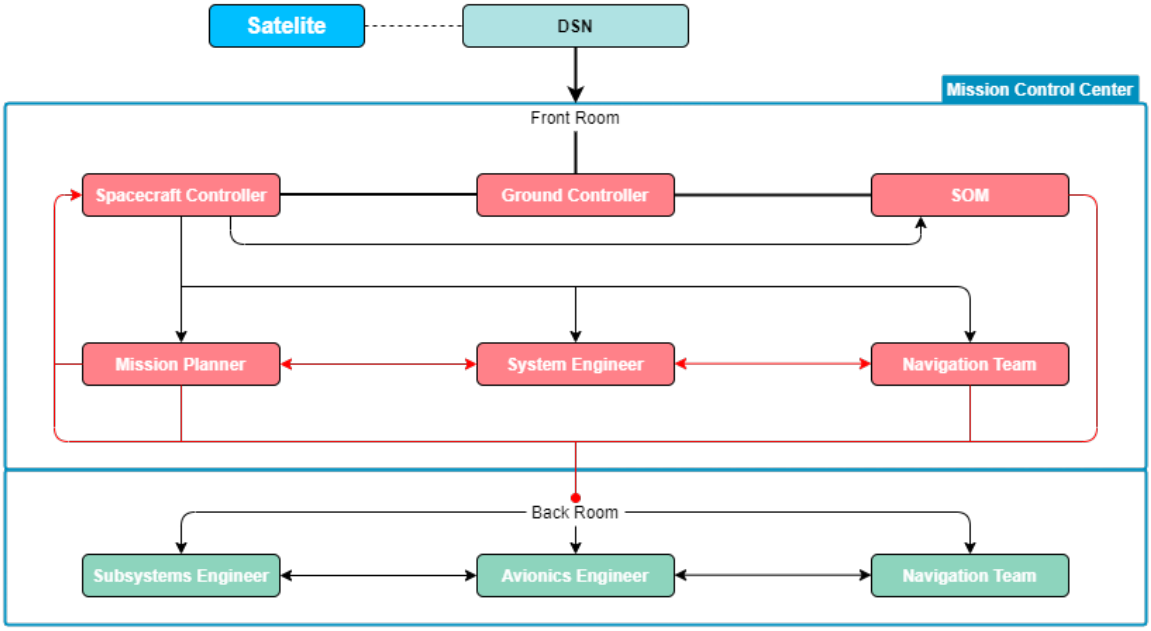


Figure 2.6: Mission Control Center Scheme and Data-Flow between the front-room and the back-room

Program Manager (PM) is responsible for the organization and overall management of the project. PM is the contact point to the customer and appoints the Flight Director (FD) and System Engineer (SE).

Spacecraft Operation Manager (SOM) is responsible for the preparation and the execution of the mission operations. The SOM develops the operations concepts, formulates technical low-level requirements resulting from that concept, and supervises the FCT during operations. In case of contingencies, the SOM will gather all recommendations of FCT and then decide appropriate actions to be taken. It has also the authority on issuing critical commands to the spacecraft. The SOM interfaces with NASA JPL/DSN and also with the backroom support.

System Engineer (SYS) is responsible for ground system engineering and defines the technical support concept of a mission's ground segment. It supervises the development of new components and adaption of existing ones also is responsible for the technical implementation of the ground segment.

Mission Planner (PLAN) is responsible for defining the mission plan, collecting inputs from other actors, and coordinating the scheduling of activities. An essential part of the PLAN is to ensure there is a communication pass available when activities are planned that require either data downloading, commanding uplink, and/or ranging information.

Spacecraft Controller (SPACON) is in charge of executing the relevant timeline (e.g. sending commands to the S/C following specific Operational Procedure, and verifying the overall health status of the spacecraft; reports to the SOM in case of anomalies etc.). Under the SOM authorization the SPACON can issue critical commands.

In case of contingencies, the SPACON need to re-schedule, in coordination with the Navigation Team and the dedicated Subsystem Engineer, the activities and is the first responder for anomaly handling.

Navigation (NAV) is in charge of the spacecraft trajectory and orbit determination for all mission phases. This information is required for both planning purposes and for real-time mission contingencies. The NAV monitors the on-board navigation system. The NAV is responsible for determining the position of the spacecraft. The NAV also analyzes the flight parameters e.g. ranging, attitude and orbital data coming from the ADCS sensors. The NAV keeps an overview of station visibility and sun pointing for power purposes.

Ground Controller (GC) is responsible for the support to ground operations and for the configuration of the control center hardware and software. The GC is the primary point of contact for any hardware or software related troubleshooting.

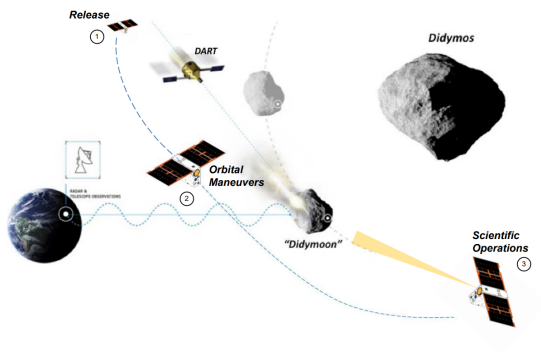
The GC position is responsible for the coordination of communication sessions between the MCC and the Deep Space Network via the Jet Propulsion Laboratory (JPL). The GC monitors the reliability of the communication link and is responsible for network configuration in the Argotec Mission Control Center.

2.4. Product Scenario

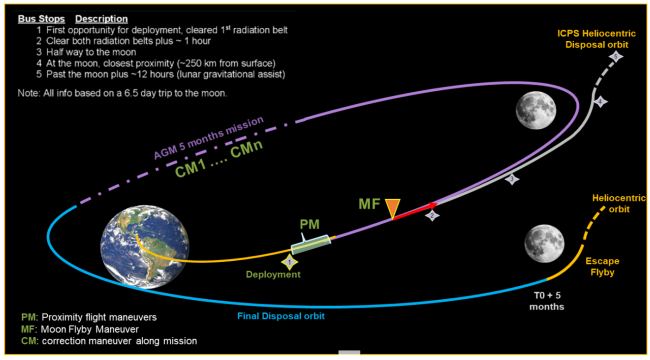
The space business environment is a challenging place for software developers, here the errors must be reduced at its minimum and the ability of properly work in real time scenarios with satellites, especially when the mission involves the Low Earth Orbit, is essential.

The Ground Segment Software must be able to help monitor the satellite telemetry during a real time communication window. It is important that the information available is optimized and easy to access by the user which must also be able to inspect historical data, of the same communication window, in case of needed analysis or contingency.

MARGOT is being developed with the possibility of changing missions profile very easily. For this master's thesis the scenarios considered are the ArgoMoon and the LICIACube missions which are Argotec's upcoming operation campaigns.



(a) Infographic of the LICIACube Mission



(b) Infographic of ArgoMoon Mission

2.4.1. LICIACube

Following the scenario presented in the Abstract, after a successful deployment from DART (planned to occur on 16 September 2022), LICIA will turn on all its subsystems, to perform in-orbit testing and check the status pf the subsystems, calibrate the equipment necessary to fulfill the mission objectives and reconstruct the satellite attitude. DART impact on Dimorphos will occur approximately 10 days after the LICIACube deployment, following a series of orbit and attitude maneuvers. Along these days, and during the Close Apporach phase to the asteroid, the satellite(LICIACube) will shoot photo of the impact area and the plume ejection. After this phase, LICIA will continue its path for a total of about 6 months in which it will be able to communicate with Earth and exchange scientific data and telemetries.

2.4.2. ArgoMoon

In an effort of increasing the scientific and explorative capability of the Space Launch System (SLS), the National Aeronautics and Space Administration (NASA) Headquarter Exploration Systems Directorate has directed the SLS Program to accommodate Secondary Payloads.

The SLS is a heavy-lift launch vehicle designed to place Exploration elements into Low Earth Orbit, to be transferred to higher orbits, and to evolve in capability to accommodate more complex and demanding missions.

The ArgoMoon mission has been selected as the first European Mission promoting the development of a CubeSat able to operate in deep space. Its photography will be used to support the NASA and payload communities in providing information regarding the status of the secondary payload deployment. The photographs of the SLS also provide for the NASA community the opportunity to visually inspect the condition of the second stage as it completes the final phase of its mission. In addition, the ArgoMoon project will be an important demonstrator for new nano/small technologies to be used in deep space for at least 6 months of lifetime.

Therefore, after the mission phase, the satellite will perform a series of autonomous maneuvers with thechemical thruster before being injected in a geocentric orbit to test the AI algorithms and

to acquire photos of the Moon and the Earth. The satellite disposal will be performed into e helio-centric orbit, as illustrated in *figure 2.7b*.

3. Requirements

There are many ways to describe what a requirement is, below are listed the two most used definitions:

> *"The software requirements specification is the basis for software development, it describes the functional and non-functional requirements and includes a set of use cases that describe user interactions that the software must provided."[4]*

> *"A functional requirement defines a function of system or its component where a function is described as a specification of behavior between inputs and outputs."[6]*

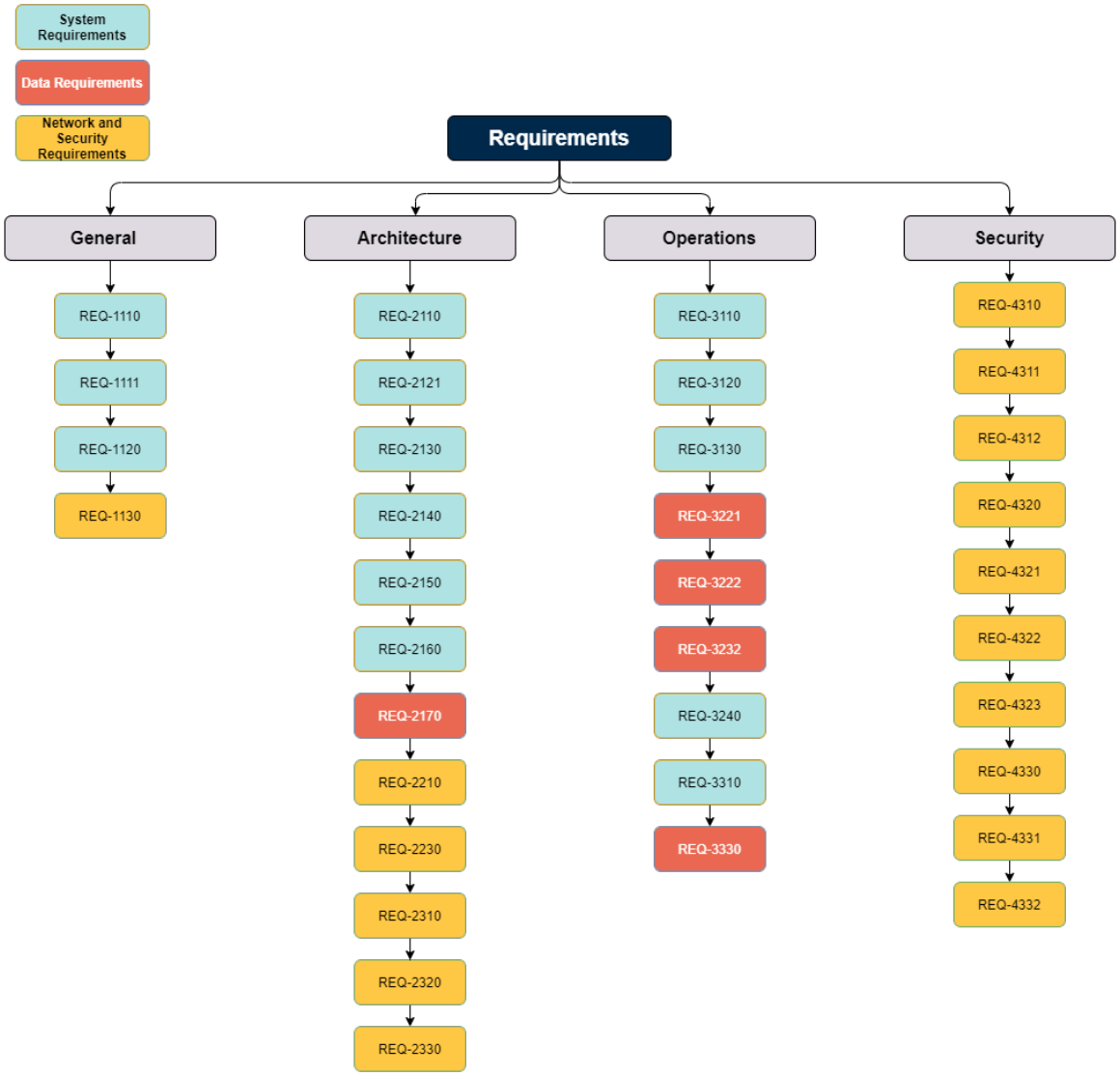


Figure 3.1: Requirements Table

A level of obligation a requirement has to met, to ensure a functional software is classified, can be set by several keywords

Shall - The requirement is essential and must be implemented

Should - The requirement is a recommendation. Without its implementation the system can be less efficient.

Can - The requirement is a possibility.

The search for these requirements involved Argotec's FCT, considering the purposes of the missions in which Argomoon and LICIACube will be operated. However, the main requirements have been developed in order to meet the possible requirements of other future missions.

3.1. System Requirements

Requirement	Description
REQ-1110	Purpose: Margot shall display to several users telemetry and other information in a personalized and secured way through a server connected to the main Argotec's Mission Control Software
REQ-1111	Margot shall later permit to review the data.
REQ-1120	Audience: Margot should be used by Argotec and other possible companies depending on stipulated contracts.
REQ-2110	There shall be three main components: • A Data Provider • A Server • A Margot Client
REQ-2121	The *data provider* (DP) shall send the telemetry through a communication channel based on TCP
REQ-2130	The *Server* shall be responsible for receiving data from DP, store it and send it to the clients
REQ-2140	The *Client* shall be devoted to data visualization in tabular and graphical representations
REQ-2150	The DP shall be able to retrieve all telemetry available from the spacecraft
REQ-2160	The Client shall be connected to the Server in order to retrieve real-time and non real-time telemetry

REQ-3110	The DP shall connect to the server which has to be already running. When the connection has been established the provider starts to send the telemetry until the session is finished or the operator stops the program
REQ-3120	At startup the Server shall read its configuration file, open a connection with the database and listening for clients connection requests
REQ-3130	During a telemetry flow, the Server will relay immediately the data coming from the DP to the clients
REQ-3240	The Client shall retrieve data from the server following user requests
REQ-3310	The Client shall be able to visualize and retrieve data after the communication has ended

Table 2: High-Level System Requirements

3.2. Data Requirements

Data requirements describe how data is managed by the system, indicating sources and destinations as well as how data is stored on memory.

Requirement	Description
REQ-2170	The Server shall be connected to the database in order to store data
REQ-3221	The Server, during a telemetry flow from the DP, shall relay the data towards the clients and at the same time store them in the database
REQ-3222	The data shall be stored indicating the telemetry, the value, the corresponding time and the communication session associated
REQ-3232	The Client can request for historical data. After the request the Server shall retrieve them from the database and send them to the corresponding client
REQ-3330	When the connection with a client is closed, the Server shall not send anymore real-time and historical data

Table 3: High-Level Data Requirements

3.3. Network Security Requirements

This section describes how the system network shall be based on and what are the security requirements needed in order to avoid unauthorized access to data or to the Arotec's network from outside.

Requirement	Description
REQ-1130	The Data Provider shall isolate the ground segment from incoming unauthorized incoming commands from the Server
REQ-2210	The DP shall connect to the Server by means of a TCP connection, only one connection shall be allowed
REQ-2230	The Client shall connect to Server by means of a TCP connection, each Client shall open its own tcp session and there is no limit to the number of connection allowed
REQ-2310	The DP shall be located in the company's Ground Segment Network
REQ-2320	The Server shall be located in a *demilitarized zone* (DMZ) in order to linsten for new client connections
REQ-2330	The clients shall be located inside the company network physically or by means of a *Virtual Private Network* (VPN) algorithm
REQ-4310	The DP shall be allowed to establish a secure connection with the Server
REQ-4311	The Ground Segment Network shall not be accessible from outside (no incoming connections)
REQ-4312	The DP shall be authenticated by the server by means of a secure certificate
REQ-4320	The Client shall authenticate in order to receive data
REQ-4321	Each Client shall connect to the server with unique credentials
REQ-4322	The Client shall receive only data authorized by means of policies
REQ-4323	The Server shall respond to clients non-real time requests only if they're compliant with data policies
REQ-4330	The Server shall accept only authorized users
REQ-4331	The Server operator shall be able to manage data policies
REQ-4332	The Server shall send each telemetry only to authorized Clients

Table 4: High-Level Network and Security Requirements

4. Design

"Software Design is the process by which an agent creates a specification of a software artifact intended to accomplish goals, using a set of primitive components and subject to constraints" [15]

Together with Argotec's FCT team a big work has been carried out in analyzing the current ground software state of the art and developing a new architecture that better fits our needs. In this section, the current software architecture is described, and a new design is proposed by defining the data structure and management, security strategies and implementation details.

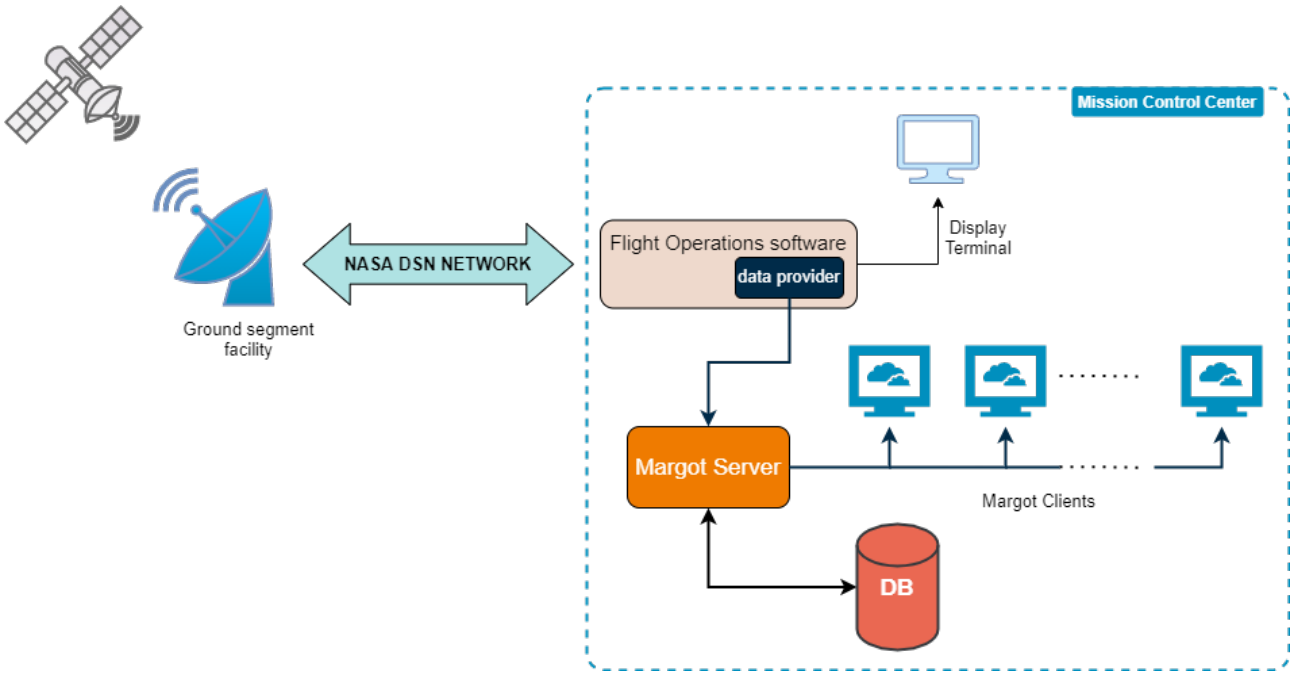


Figure 4.1: Proposed Ground system architecture

The state of the art of the system presented earlier in *chapter 2* allows to correctly visualize telemetry through Display Terminals and send telecommands using FOS.

In this section the project design is explained.

4.1. How the system communicates

As can be seen in *figure 4.1*, the system is composed of a distributed architecture, this means that the communication medium plays a fundamental role in the system composition, robustness and security, as described in later sections.

The choice for implementing communications between system components laid on the *Message Queue Telemetry Transport* (MQTT) protocol.

MQTT is a Client Server publish/subscribe messaging transport protocol. It is lightweight, open,

simple, and designed to be easy to implement. These characteristics make it ideal for use in many situations, including constrained environments such as for communication in *Machine to Machine* (M2M) and *Internet of Things* (IoT) contexts where a small code footprint is required and/or network bandwidth is at a premium [12].
The protocol runs over TCP/IP. Its features include:

- use of the publish/subscribe message pattern which provides one-to-many message distribution and decoupling of applications.
- A messaging transport that is agnostic to the content of the payload.
- Three qualities of service for message delivery:
  - *"At most once"*, where messages are delivered according to the best efforts of the operating environment. Message loss can occur.
  - *"At least once"*, where messages are assured to arrive but duplicates can occur.
  - *"Exactly once"*, where messages are assured to arrive exactly once.
- A small transport overhead and protocol exchanges minimized to reduce network traffic.
- A mechanism to notify interested parties when an abnormal disconnection occurs.

MQTT defines two types of network entities:

- A message broker, constituting of a server that receives all messages from the clients and then routes the messages to the appropriate destination clients.
- An MQTT client, representing any device that runs an MQTT library and connects to an MQTT broker over the network.

Information is organized in a hierarchy of topics. Topics are treated as a hierarchy, using a slash (/) as a separator. This allows sensible arrangement of common themes to be created, much in the same way as a file system.
For example, the mission ArgoMoon (here indicated as AGM) multiple telemetries can be published depending on their subsystem:

AGM/OBC/VTM

Which represents the topic where the on-board computer voltage telemetry will be published. Two wildcards are available:

- + that can be used as a wildcard for a single level of hierarchy. It can be placed multiple times for different levels.
  For example

AGM/+/VTM

could be used to get voltages from all subsystems.

- # that can be used as a wildcard for all remaining levels of hierarchy.

  AGM/OBC/#

  could be used to get all telemetry related to the on-board computer.

When a publisher has a new item of data to distribute, it sends a control message with the data to the connected broker. The broker then distributes the information to any clients that have subscribed to that topic. The publisher does not need to have any information about the number or locations of subscribers, and subscribers, in turn, do not have to be configured with any data about the publishers.
As a standard functionality, the broker doesn't store messages, in fact, if there is a publication on a topic on which there are no current clients subscribed the message is discarded. A possible solution to this is suing retained messages where the broker can store for a short period of time messages of a specific topic but it is not a functionality exploited in this project as data are stored in a dedicated memory location.
The open source nature of the MQTT protocol has permitted the creation of many libraries for almost all languages and development of ready-to-use brokers which only need to properly configured. For this project the Eclipse Mosquitto broker have been chosen, an open source (EPL/EDL licensed) message broker developed by the Eclipse Foundation and sponsored by Cedalo.

4.2. Subsystem Decomposition

As can be seen in figure 4.1 the new proposed architecture is composed by three main components:

- Margot Server
- Data Provider
- Margot Clients

4.2.1. Margot Server

The Margot Server represents the core of the project and is composed by two functional elements:

- A MQTT broker, placed on a server machine, which is responsible for authenticating the Data Provider and Margot Clients, receiving data from DP and send it to MCs subscribed to the corresponding topic. As stated before, the server doesn't normally manage retained messages

and delegate data storage to the DHM.

Topics are organized in a hierarchical way, as can be seen in *figure 4.2*. Spacecraft telemetry is divided in topics representing its subsystems. Report data such as events, anomalies, and other packets received on request are under the *report* head topic. Also, *history* head topic is used to handle historical data requests from Margot Clients. When a client requests historical data it publishes the message indicating the telemetry needed and the timespan in unix time inside the sub section called by its client ID repeated twice. The response, if present, is published under the corresponding subsection, indicated by the client ID, using the same topic structure used for the spacecraft telemetry.

To give an example, if a client having id equal to *SystemEngineer* requests the on-board computer voltage historical data for the ArgoMoon mission it will publish the request on

AGM/history/SystemEngineer/SystemEngineer

and the response will be published on

AGM/history/SystemEngineer/OBC/VTM

This kind of organization permits handling different requests at the same time without overlapping but it is less efficient because of possible repeated data over different topics.

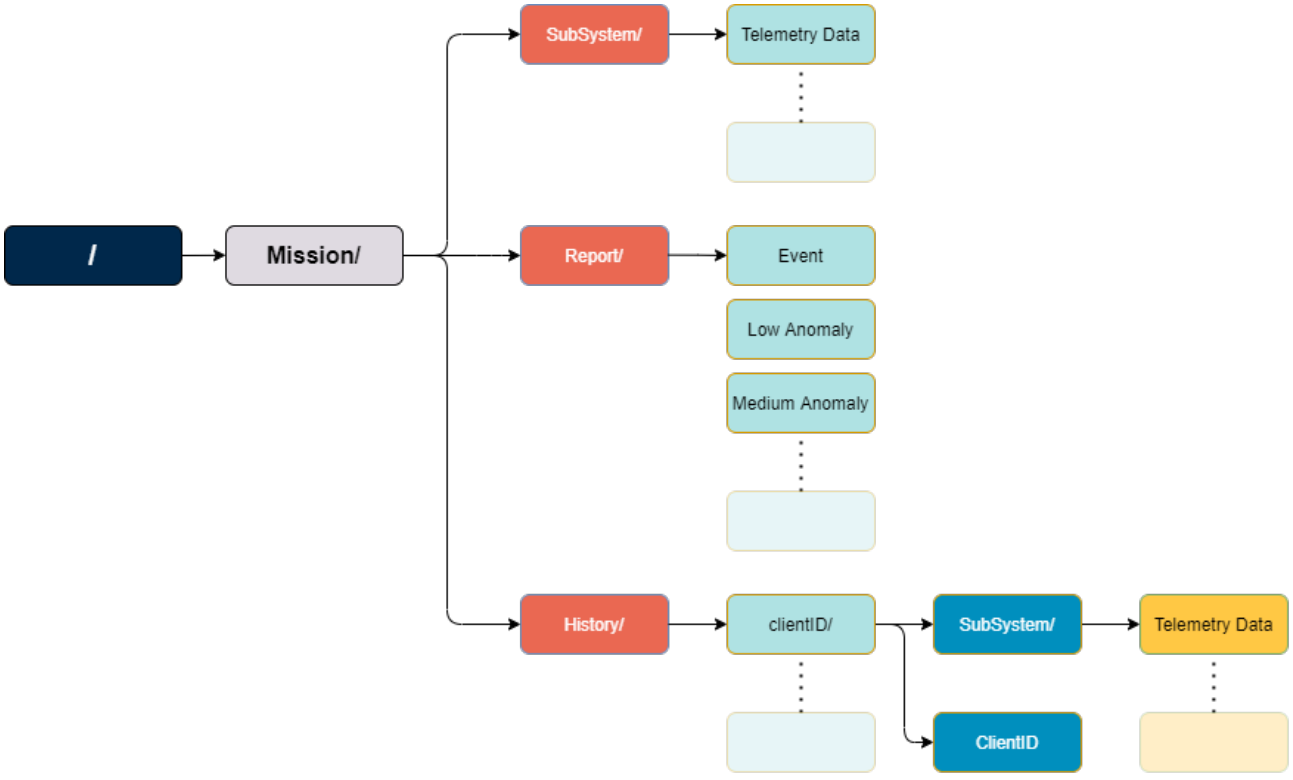


Figure 4.2: Topic Structure Scheme

- A *Data and History requests Manager* (DHM) runs alongside the MQTT server. It is responsible for data saving and historical data requests. The DHM manage a database containing all

telemetry as well as report data. When the application startups it loads the mission profile (e.g. ArgoMoon or LICIACube) for that communication session and will keep it until the end of the comm window. DHM is composed by two agents:

- The first one is the *Incoming Messages Manager* (IMM). It checks if packets are formally correct and if they contain all information required depending on the type of request. If all checks are passed the task is inserted in a queue waiting to be processed by the second agent.

  Messages are in the form of a JSON file. Telemetry and event data use the following format:

```
{
    "id": <topic.subtopic>,
    "timestamp": <unix time in milliseconds>,
    "value": <data (number or string)>,
    "TYPE": 0
}
```

Listing 1: Telementry JSON format

  where *id* indicates what telemetry is contained in the message, and it corresponds to the topic with slashed (/) substituted by dots (.). *TYPE* field is used to distinguish between incoming data (0) and historical data requests (1). Meanwhile, request messages use a slightly different structure:

```
{
    "id": <history.cliendID.CliendID>,
    "start": <start time requested in unix time in milliseconds>,
    "end": <end time requested in unix time in milliseconds>,
    "TYPE": 1
}
```

Listing 2: Request messages structure

  Where *start* and *end* fields delimit the timespan required for the historical data.

- The second agent is the *Incoming Tasks Consumer* (ITC). It manages all tasks inserted in the queue filled by IMM. The ITC uses a server-like implementation in order to respond to task execution requests. It is composed by a main thread used to retrieve new tasks and worker threads that will actually compute the assignment.

  All tasks need to access the database, either to data storing or for data retrieving, when requested from a client and send data over an IP network. Database access is made efficient thanks to the *Database Management System* (DBMS) but still data are stored on

a secondary memory which is intrinsically slower. Therefore, it is important to optimize tasks execution, memory access, and parallelize network usage in order to have faster response time and avoid too many tasks waiting time.

The main thread works on a loop system which extracts data from the queue and insert it to a dedicated thread pool as in *figure 4.3*.

The necessity of having an ordered and optimized tasks execution has led to the implementation of a thread pool. The thread pool permits achieving concurrency in a computer program and also maintaining multiple threads waiting for tasks to be allocated for concurrent execution by the supervising program. By maintaining a pool of threads, the program increases performance and avoids latency in execution due to frequent creation and destruction of threads for short-lived tasks [18]. As can be seen in *figure*

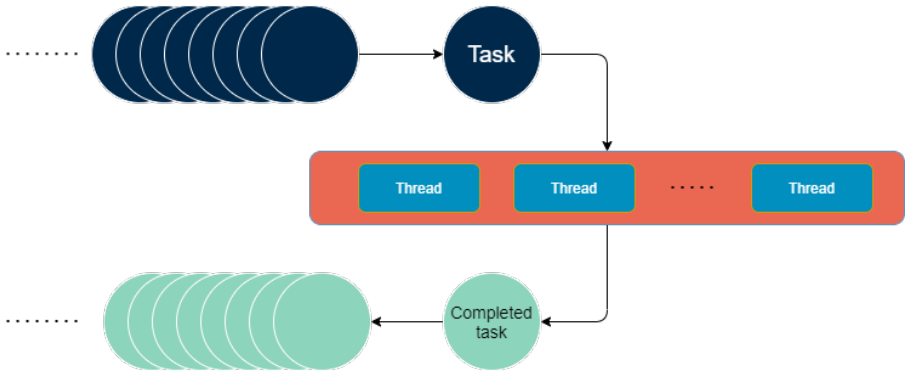


Figure 4.3: Simple thread pool implementation

4.3, the main thread will try to assign a task to the first available thread and, if none is available, it has to wait. With this kind of implementation the main thread must wait a long time before the task can be assigned to an available worker making the system less efficient.

To overcome this type of problematic a more efficient structure have been implemented [14]. The new architecture is composed by a series of queues, one for each thread in the thread pool. When a new task arrives it is assigned to the first available queue. The task will travel down the queue until it is assigned to a thread. The task assignment is performed by a *Task Stealer* (TS) which work is demanded to evenly distribute tasks over different queues. What it does is take a task from a queue and assign it to a possibly different thread than the one connected to the original queue. This kind of operation is performed for example when a queue is empty and its thread is not working, and another thread has waiting tasks in its queue, in this case the TS will take a task from the other queue and assigns it to the unloaded thread. There are on the market official frameworks like Windows Thread Pool, PPL for Windows or Apple Grand Central Dispatch for Unix Systems but they are platform dependent. Instead, the solution presented here is based on

C++17 standard and can work everywhere. This implementation can be 100 times better than a single queue thread pool [14], and results only 16% slower than Apple Central Dispatch framework which exploits kernel libraries.

```
2021_11_11_16_19:        Delivery complete for token: 39884
2021_11_11_16_19:        Delivery complete for token: 39885
2021_11_11_16_19:        Delivery complete for token: 39886
2021_11_11_16_19:        Delivery complete for token: 39887
2021_11_11_16_19:        Delivery complete for token: 39888
2021_11_11_16_19:        Delivery complete for token: 39889
2021_11_11_16_19:Querying ...
 SELECT time,value from tm_session, schedule where  tm_session.Sid = schedule.
    ↪ Sid AND tm_session.Tid = 1358 AND  tm_session.Sid = 41 AND tm_session.
    ↪ time BETWEEN 1636647271399 AND 1636647601399 ORDER BY time;
2021_11_11_16_19:Querying ...
 SELECT time,value from tm_session, schedule where  tm_session.Sid = schedule.
    ↪ Sid AND tm_session.Tid = 1360 AND  tm_session.Sid = 41 AND tm_session.
    ↪ time BETWEEN 1636647271399 AND 1636647601399 ORDER BY time;
2021_11_11_16_19:        Delivery complete for token: 39890
2021_11_11_16_19:        Delivery complete for token: 39891
2021_11_11_16_19:Querying ...
 SELECT time,value from tm_session, schedule where  tm_session.Sid = schedule.
    ↪ Sid AND tm_session.Tid = 1358 AND  tm_session.Sid = 41 AND tm_session.
    ↪ time BETWEEN 1636643973906 AND 1636647603906 ORDER BY time;
2021_11_11_16_19:Querying ...
 SELECT time,value from tm_session, schedule where  tm_session.Sid = schedule.
    ↪ Sid AND tm_session.Tid = 1360 AND  tm_session.Sid = 41 AND tm_session.
    ↪ time BETWEEN 1636643973906 AND 1636647603906 ORDER BY time;
2021_11_11_16_19:        Delivery complete for token: 39892
2021_11_11_16_19:        Delivery complete for token: 39893
```

Listing 3: Example of DHM Output

The ITC, with this optimization strategies, is able to manage thousands requests for historical data during a communication session in few seconds while still storing new incoming data. In *listing 3* can be seen an example the number of requests, reaching almost forty thousands, served by the DHM after an hour of communication session with the front room and back room clients connected together.

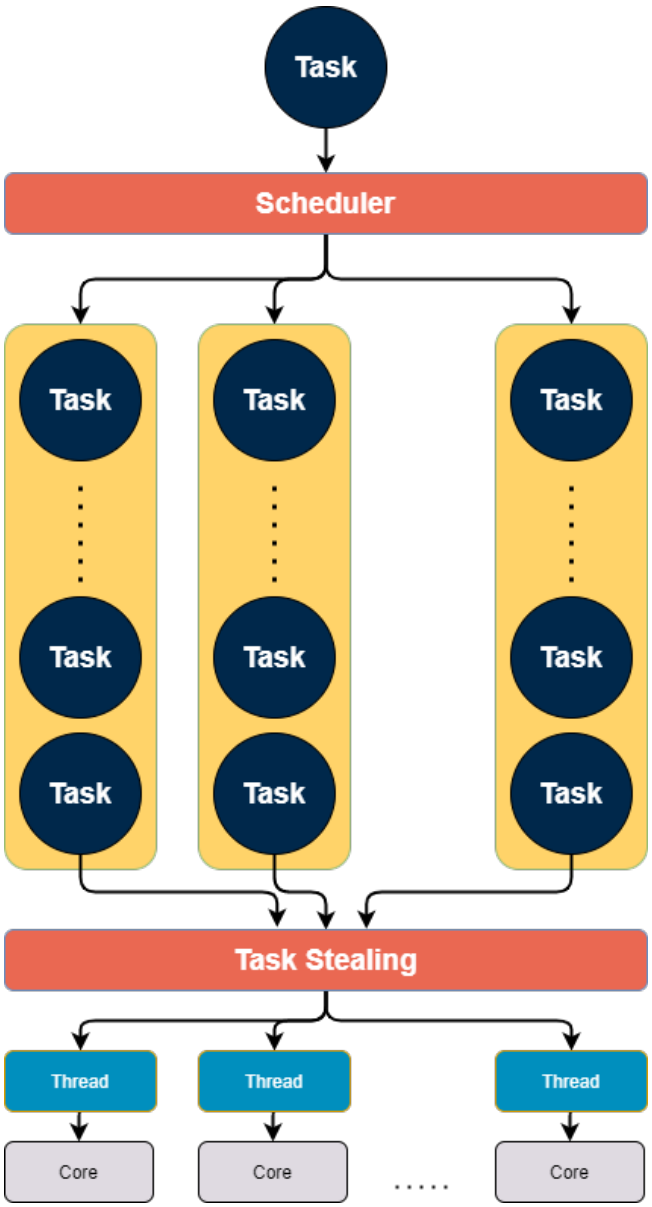


Figure 4.4: Complete thread pool architecture for the Incoming Task Consumer

4.2.2. Data Provider

The Data Provider is a TCL (pronounced "Tickle") framework used to retrieve data from FOS. *Tool Command Language* is a powerful, open source, cross-platform dynamic programming language, suitable for a very wide range of uses, including web and desktop applications, networking, administration, testing and many more [17].

DP is mainly composed by two working sequences (i.e. two scripts) each one running on a single thread environment:

- A *Housekeeping Provider* (HP) which listens for a *Housekeeping* message containing all the spacecraft telemetry. The frequency of this type of message combined with the need to have data as soon as possible, has raised the necessity to have a dedicated working sequence. When a new housekeeping message arrives the HP will read all telemetry and send it one by

one to the Margot Server.

The script is optimized to send data before a new packet arrives from the spacecraft considering a max data rate of 1 pck/s.

- An *Asynchronous Packet Provider* (APP) which listens for asynchronous messages like events, anomalies, and solicited messages (i.e. messages receives after an explicit request from the ground station). Because of the asynchronous nature of messages, the APP creates a series of interrupts serving a specific packet. In case one or more interrupts are fired in a short period of time an automatic schedule mechanism managed by the TCL interpreter inserts messages in an execution queue.

The two working sequences described earlier share some common areas as their functionalities are similar. The scripts at startup use a configuration file which indicates what type of data to acquire and how to use it using this notation:

Figure 4.5: Data Provider configuration file format

TM ID represents the FOS telemetry code and it is used to subscribe to the correct telemetry.

TOPIC represents the topic where the telemetry will be published. The dot is used as separator character instead of the slash (/) because, as described before, inside the MQTT package the topic field uses dots for a better compatibility with Margot Client software, but messages are still published on slashed separated topics.

SELECTION/TRANSFORMATION field is optional and is used in order to modify standard acquisition and sending flow of telemetry. This is due to the fact that each telemetry packet has two fields for data value:

RAW which is an integer value coming directly from a sensor or representing a state value.

ENG which represents a transformation of the raw value. In case of sensor or general numeric values it applies a polynomial transformation, for example a value of 3400 on the *Solar Panel Temperature* is then transformed to 35.6 degree Celsius.

Instead, if state values are considered, it is composed by a string taken from a lookup table, for example the value 3 on the *Operational Mode* of the spacecraft is translated in the *"Communication Mode"* string.

It is important to notice that not always the ENG value is present as the RAW value is the only available field in the packet.

Having described the transformation that can be applied directly by FOS before the acquisition, field states are described:

- row/eval indicates that the value to be taken from the package is the RAW value and if the ENG value is present it is ignored. Raw and eval keywords have the same effect but they are used for logical separation.
- 'function' indicates the path to an external function used to transform the raw value before sending it out. It used when a synthetic value is required and/or a polynomial transformation is not enough.
  A good example is represented by the "download image telemetry" where the script calculate the percentage of completion starting from the number of total packets composing the image and the number of the most recent packet arrived.

4.2.3. Margot Clients

A Margot Client (MC) is an application running on end terminals used to visualize telemetry by means of a browser. All subsystems and code implementation described before works together in order to provide a robust, reliable and fast way to retrieve data from the spacecraft and allow the user to efficiently analyze data. The standardization of incoming messages permits developing different clients which are used for various use cases. For example, a Margot Client is integrated inside the *Mission Planning Tool* (MPT) where few important telemetries are visualized to keep track of the correct execution of the planned communication windows timeline.
What has been developed in this thesis is a complete framework which permits to visualize all telemetries and construct displays similar to what can be seen in *figure 1.1*.
The MC is a web application, running on browser, based on the OpenMCT project developed by NASA [13]. OpenMCT permits to implement new functionalities via a plugin system where everyone can develop its own for the most various scenarios. For this project two plugins have been implemented:

- The first one configures OpenMCT by telling which are the telemetries. The plugin, at startup, loads a configuration file written in JSON format called *dictionary* where telemetry information is stored. An example of what a dictionary can contains can be seen here:

```
1 {
2     "name": "LICIACube",
3     "key": "LCC",
4     "measurements": [
5         {
```

```
            "name": "Battery State of Charge",
            "key": "LCC.POWER.soc",
            "values": [
                {
                    "key": "value",
                    "name": "Value",
                    "units": "%",
                    "format": "float",
                    "hints": {
                        "y": 1,
                        "range": 1
                    }
                },
                {
                    "key": "utc",
                    "source": "timestamp",
                    "name": "Timestamp",
                    "format": "utc",
                    "hints": {
                        "x": 1,
                        "domain": 1
                    }
                }
            ]
        }
    ]
}
```

Listing 4: Dictionary JSON structure

As can be seen the dictionary is formatted as a JSON object containing on the inside all information about the spacecraft, starting from the name and its identification key. The *measurement* field contains a vector where telemetries are listed. Each telemetry is identified by its name and key, which in this implementation also represents the relative topic with slashes substituted by dots. The *Values* field describes the telemetry format:

The first object is the value object and describe the telemetry data characteristics like the format and the unit. The other information are used to properly setup OpenMCT. In the case of an enumerating telemetry, this field will also contain the conversion lookup table.

```
{
```

```
    "key": "value",
    "name": "Operational Mode",
    "format": "enum",
    "enumerations": [
        {
            "value": 0,
            "string": "SUN POINT MODE"
        },
        {
            "value": 1,
            "string": "COMMUNICATION MODE"
        }
    ]
}
```

Listing 5: Values first object field structure

The second object defines the time format to use in order to position data in the correct timespan. The format field indicates what time of time is being used and in this implementation the standard configuration has been used by setting the *Coordinated Universal Time* (UTC).

The information contained in the dictionary file could be also used to hide telemetries to certain users following security measurements, but this aspect will be discussed in the *Access Control and Security* section.

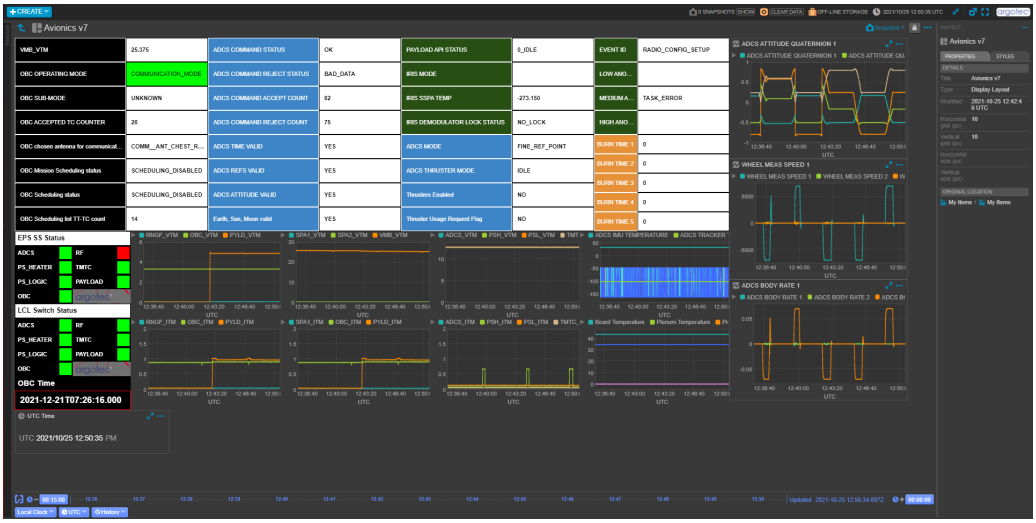


Figure 4.6: Margot display developed by Argotec

- The second plugin is demanded to data reception and requests. At startup the program connects to Margot Server and maintains the connection using ping messages. When the

user wants to visualize a telemetry it will subscribe to the corresponding topic, defined in the dictionary as described before, and at the same time sends a request to the DHM for historical data in order to retrieve old data depending on the time span visualized on graph. The plugin will request data also in case the user decides to manually move across the timeline.

If a display containing multiple telemetry like in *figure 4.6* is opened the plugin will perform the same operations describe before for each one of them.

4.3. Persistent Data management

As stated before, the Data and History requests Manager is responsible for data consistency by saving all incoming telemetry into a database managed by a DBMS. Formerly a DBMS is defined as a "software system that enables users to define, create, maintain and control access to the database" [2]. The DBMS chosen for this project is MySQL, an open-source Relational DBMS, called like that because it is based on a relational model.

The objective of having a database for this application is the necessity of retrieving data really fast and under some specific condition of time or location (in this case considered as the subsystems of the spacecraft).

4.3.1. Relational Model

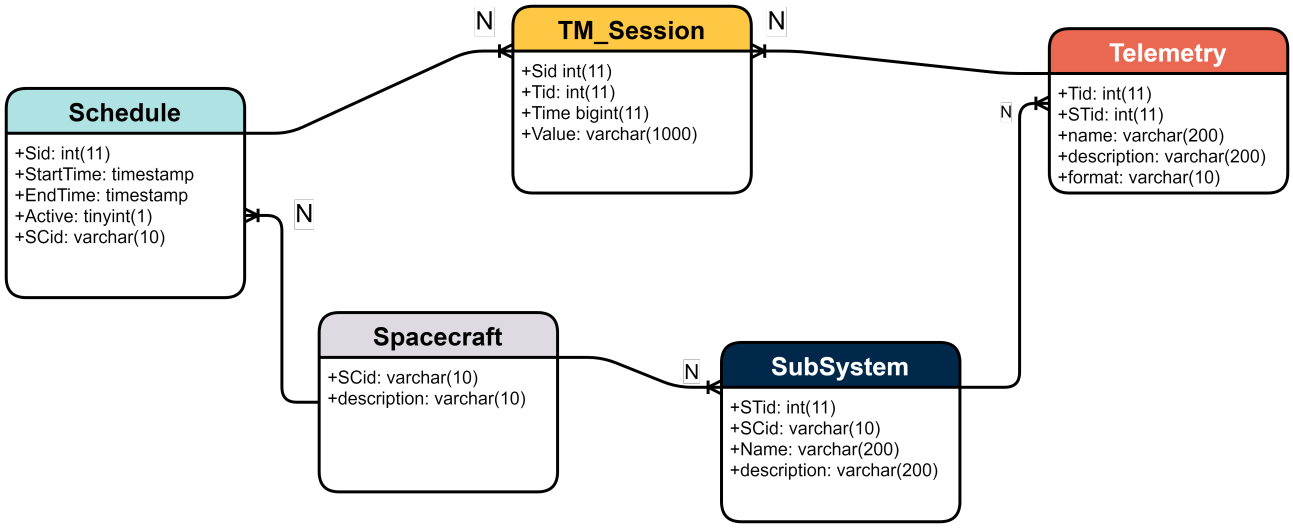


Figure 4.7: Database Relational Model

The relational model of the Spacecraft is very straightforward. The structure has been optimized in order to easily access telemetry based on the corresponding communication session or the belonging subsystem. The system is thought to be used as a single database for all missions leaving to proper querying the capability of filtering data using the spacecraft identification code (*table 5*). Since Margot is being developed as a spacecraft telemetry viewer, referring also to *figure 4.7*, each

mission is represented by an entry inside the *Spacecraft* table. Each system is then decomposed in subsystems stored in the *SubSystem* table and finally the telemetry is saved inside *Telemetry* table.

To keep track of each communication session the *Schedule* table is used containing for, for each entry, the associated mission code. The *Active* field indicates which session is in communication, and more generrally what session is used to store incoming data, or is scheduled to be next. This field is really important as it is used by the DHM to associate incoming telemetry to the correct communication session.

To store telemetry corresponding to a communication session, a *many-to-many* (N-to-N) relactionship between *Telemetry* and *Schedule* tables is necessary resulting in a new table indicated here as *TM_session* and used by DHM to memorize data coming from Margot Server.

Table	Item	Description
Spacecraft	SCid	Spacecraft (mission) identification code
	Description	Very brief description of the mission
SubSystem	STid	Subsystem identification code
	SCid	Spacecraft (mission) identification code
	Name	Name of the subsystem
	description	description of thr subsystem
Telemetry	Tid	Telemetry identification code
	STid	Subsystem identification code associated with the telemetry
	Name	Name of the telemetry
	description	description of the telemetry
	format	C-like telemetry format
TM_Session	Sid	Communication session identification code
	Tid	Telemetry identification code
	Time	timestamp associated with telemetry data
	Value	value associated with telemetry data
Schedule	Sid	Communication session identification code
	StartTime	Communication session start time

	EndTime	Communication session end time
	Active	Active session flag
	SCid	Spacecraft (mission) identification code associated with the communication session

Table 5: Database Description

4.4. Access Control and Security

As reported in the requirements, Margot will be used by a multitude of people coming from different areas, projects, and companies. This characteristic of the project led to the development of an access control system and the adoption of some security measures. Despite the *Access Control* (AC) is part of the information security field, its importance in this project demands a separate section.

4.4.1. Security

> *"Computer security is the protection of computer systems and networks from information disclosure, theft or damage to their hardware, software, or electronic data, as well as from the disruption or misdirection of the services they provide."[10]*

Margot is a web application where its components communicate using an IP network which is intrinsically not secure. The user authentication is performed using a specific plugin which is described later. The importance of having security measures is due to the fact that Margot's clients are allowed to connect from three different logical locations:

- From the *Mission Control Center* network where the Data Provider is located.
- From Argotec's network.
- From outside Argotec's network.

The first two are protected using specific network policies inside Argotec's firewalls that control the traffic looking for possible malicious irruptions. The third is connected using a secure connection tunneling based on the IPsec protocol [1] which provides packet encryption and authentication between two hosts over the internet protocol. It is also used by the DSN to send data to the Mission Control Center in the first place. In order to accomplish such functionalities the protocol is composed of these main protocols:

- *Authentication Header* (AH) which is used for authenticating data origin of IP datagrams, checking data integrity, and provide protection against replay attacks.
- *Encapsulating Security Payload* (ESP) provide authentication of data origin for IP datagrams, data confidentiality and a limited flow confidentiality.
- *Internet Security Association and Key Management Protocol* (ISAKMP) which provides a framework for key exchange services using pre-shared keys, *Kerberized Internet Negotiation of Keys* (KINK), IPSECKEY DSN records, or *Internet Key Exchange* (IKE), and authentication services. The purpose of this framework is to provide a *Security Association* (SA) for key signing.

With these security measures Margot is able to operate with secured authentications and data confidentiality inside Argotec's network but also outside.

4.4.2. Access Control

The AC is used for the user authentication service. Each agent that wants to authenticate to the Margot Server has to provide a backhoe of credential string formed by:

- ClientID
- Username
- Password

Having a combination of three credential fields permits to have a much more secure and strong authentication.
The authentication is performed by Margot Server using Mosquitto *Dynamic Security Plugin* (DSP) [5]. It permits user authentication using the triplet described before. Also, multiple users can be placed into a group in case they need to have the same access. DSP allows also to disable users without permanently removing the credential. Access control lists are configured by defining roles, which are containers of multiples ACLs and can be assigned to clients and/or groups.
ACLs are the feature which allows access to topics to be controlled. Checks are made on different events as they happen:

- *publishClientSend* occurs when a device sends a PUBLISH message to the broker (i.e. is the device allowed to publish to this topic). By default, it is set to deny.
- *publishClientReceive* occurs when a device is due to receive a PUBLISH message from the broker (i.e. it has a valid subscription and a matching message has been published to the broker). By default, it set to allow.
- *subscribe* occurs in response to a client sending a SUBSCRIBE message. By default, it set to deny.

- *unsubscribe* occurs in response to a device sending an UNSUBSCRIBE packet. By default, it set to allow.

Each ACL has a topic and a priority, and can be set to *allow* or *deny*. The *publishClientSend* and *publishClientReceive* types map directly to the events of the same name. The topic can contain wildcards, so allowing send access to *AGM/OBC/#* will allow devices to publish to all topics in the *AGM/OBC/#* hierarchy, including *AGM/OBC*. The *subscribe* and *unsubscribe* events have two ACL types each: *subscribeLiteral*, *subscribePattern*, *unsubscribeLiteral*, and *unsubscribePattern*.

- **Literal* ACL types make a literal comparison between the topic filter provided for the ACL and the topic filter provided during subscribing or unsubscribing. This means that setting a *subscribeLiteral* ACL with topic filter to # to deny would prevent matching devices from subscribing the # topic filter only, but still allow them to subscribe to *AGM/#*, for example.
- **Pattern* ACL types allow or deny access based on a wildcard comparison of the ACL topic filter and the topic provided during subscribing or unsubscribing. This means that setting a *subscibePattern* ACL with topic filter # to deny would prevent matching devices from subscribing to any topic at all.

There is some overlap between *publishClientRecevive* and *subscribe*. Usually, *subscribe* is sufficient, however they can be combined to allow subscriptions to a wildcard topic like *AGM/#*, but deny access for devices to receive messages on a specific topic within that hierarchy like *AGM/OBC*.

5. Software validation

> *"Verification and Validation processes are used to determine whether the development products of a given activity conform to the requirements of that activity and whether the product satisfies its intended use and user needs. The scope of software validation processes encompasses systems, software, and hardware, and it includes their interfaces. This standard applies to systems, software, and hardware being developed, maintained, or reused."[8]*

Performing a software validation is essential as it allows taking confidence with the new developed system, understand its limits, and more importantly verify that it can be used in an actual space mission. There are mainly two aspects of Margot that are critical. The first one is related to the system information reliability: Margot will be used during real-time communication sessions with a satellite, and each minute, in which the CubeSat is in *communication mode*, must be used in the more efficient way. For this reason, Margot shall provide real data without modifications of any kind that can cause false positive contingency situations. Therefore, it is necessary that data visualized by Margot is comparable to the one provided by the MCS.
The second aspect is also related to reliability: Margot must perform its work without stalling or stopping at all. For the same reason explained earlier, Margot is mostly used by users having a fundamental role in mission operations, so it is important for these figures to have constant access to the most recent telemetries to prevent missing crucial events or problematics that can compromise the good outcome of the communication session or the entire mission. With these objectives in mind, Margot has been tested and validated by Argotec's Flight Control Team exploiting the testing and training campaign for the two LICIACube and ArgoMoon missions.

5.1. Test Setup

For test setup is intended all operations needed to be performed on the environment, in this case Argotec's Mission Control Center, in order to simulate the working environment as much as possible. Two different setups have been taken in account, Ground Data System (GDS) tests and Operations Training tests, used depending on the mission preparation schedule, as can be seen in *figures 5.1* and *5.3* where Argotec's Mission Control Center is simplified for a better readability but for reference it is considered the same structure as in *figure 4.1*.
During each of the tests, Margot is used to display the arriving telemetry and, at the end, during debriefing, feedbacks are collected from the people involved in the testing, in order to resolve bugs and implement useful functions (i.e. addiction of telemetries or development of new displays).
For all tests Margot Server runs on a Windows 10 machine, with a 6-core Ryzen 5 3600X at 3.79Ghz

with 16 GB of RAM.

5.1.1. GDS Tests

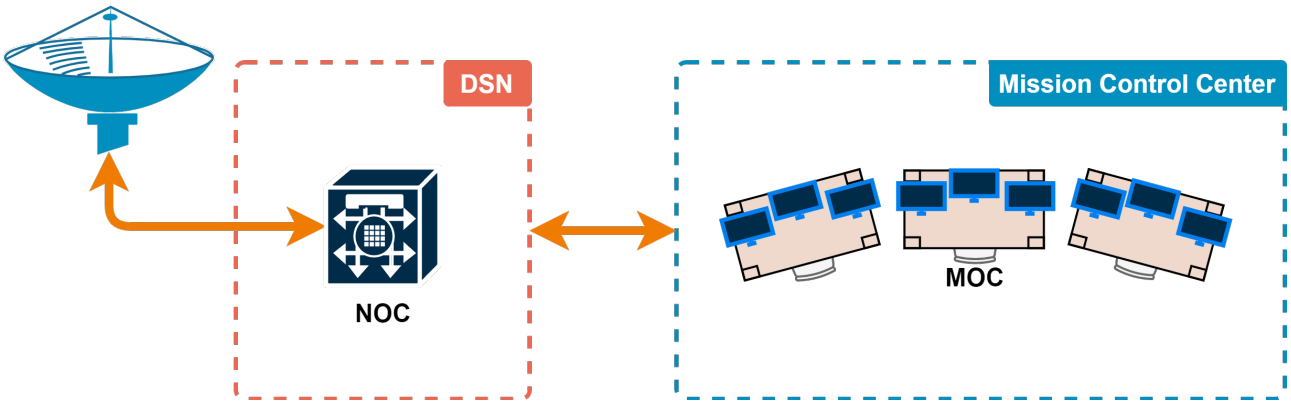


Figure 5.1: GDS setup

Ground Data System are tests required from the *Mission Interface Manager* assigned to the mission, where the MCC is requested to be directly connected to the DSN Operation Center and the dedicated Antenna, via the MCS. This test is required by NASA to ensure that the connection between the MCC and each of the antennas that will be used during real-time operations is working fine, telemetry successfully flows to the MCC, the MCC is able to send telecommands and the dedicated antenna for the test is able to irradiate the telecommands to the space. The test must be repeated for each of the antennas of the DSN and its good outcome is essential for the optimal approach at the mission startup. This kind of test is defined *End-To-End* (E2E) as the network infrastructure involved start from the antenna's uplink/downlink path and arrives to the Argotec's MCC.

The GDS is also useful to test the main MCS, including the FOS, and detect possible bugs that can compromise the communication sessions so that the FCT can work to a solution before actual real-time operations.

In particular, the test is performed between Argotec's *Mission Operations Center* (MOC) and NASA's *Network Operations Center* (NOC) by means of a real-time Voice over Intenet Protocol (VoIP) communication session called *LOOP*. The loop is established between the project's MCC (in this case, the Argotec's MCC) and the Station Conferencing and Monitoring Arrangement (SCAMA) and follows specific rules, mostly military-style, which help to reduce most communication ambiguity by defining calling procedures, voice check protocols, alphanumeric and numeric spelling rules (i.e. using the NATO phonetic alphabet like in *figure 5.2*).

During the GDS tests, the following tests have been performed on Margot using a single client equipped with a 4-core Intel i5-1135G7 at 2.42Ghz with 8 GB of RAM connected to Margot Server:

- System reliability over time.

- System resources usage, in particular primary memory, and CPU.
- System network activity and band occupancy.
- Real-time delay between MCS and Margot displays

A Alpha	B Bravo	C Charlie	D Delta	E Echo
F Foxtrot	G Golf	H Hotel	I India	J Juliet
K Kilo	L Lima	M Mike	N November	O Oscar
P Papa	Q Quebec	R Romeo	S Sierra	T Tango
U Uniform	V Victor	W Whiskey	X X-ray	Y Yankee
Z Zulu				

Source: https://tinyurl.com/dmsc633u

Figure 5.2: NATO Phonetic Alphabet

5.1.2. Operations Training

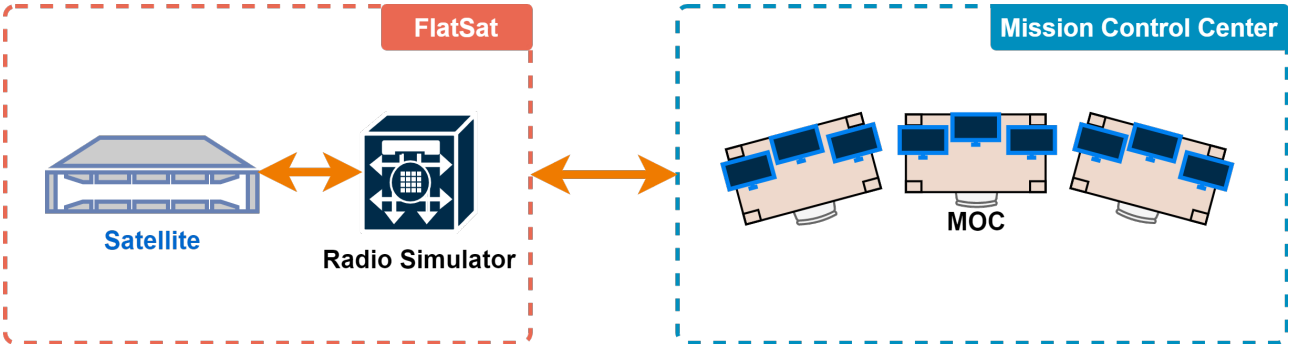


Figure 5.3: Operations trainig setup

Operations training consists of all those activities that aim at performing better, just like an athlete has to train to increase its performance, the FCT needs to train to improve its productivity and efficiency as a team. The training has the objective to prepare the FCT to real communication sessions in various aspects:

Voice Loop Protocol . Following the same protocol used during GDS tests, but this time it is used for internal communications. With the assistance of a dedicated software that permits to establish a main communication channel, called *master loop*, it is defined where all important information are communicated along with a series of secondary loops for private communications between components of the FCT. For example, if a private loop assigned to the Ground Controllers to the mission is present (i.e., the *GC loop*),everyone in the front or in the

back rooms who wants to speak with the Ground Controller privately can enter this loop.
The purpose of doing Operations Training using the voice loop protocol consists of becoming more confident with it and increase the efficiency using fast, concise and useful exchange of information.

Understanding the spacecraft behavior People working in the FCT usually differ to the ones who developed the spacecraft. The FCT needs to take confidence with the spacecraft, to understand how it behaves not only by reading the documentation but also by actually using it. The main difficulty resides in the impossibility of using the real spacecraft for security and logistic reasons (i.e. it could be already mounted on-board the launcher or the training campaign is performed after the launch event). To overcome this problematic Argotec's FCT is equipped with a *Hardware in the Loop* (HIL) system, called FlatSat, that is an exact replica of the flying spacecraft. They share the same hardware as well the same software version, unless the test is focused on a new software update. In that last case the new software release is tested to understand if the new version do what it is expected to (i.e. if a corrective patch does resolve the problems and doesn't add new ones) before being uploaded onto the real spacecraft.

Understanding ground software limitations and finding new bugs As for the GDS tests, operations training is an opportunity to gain experience with the MCS through different mission scenarios, to discover problems with the software functionalities, and trying new solutions that could prevent those to appear.

Verification and Validation of Procedures Each operation performed on the spacecraft, like shooting and downloading a photograph, usually can't be performed by just clicking a button but instead a series of telecommands and checks on the telemetry must be followed, as can be seen in *table 6*. A good comparison is the checklist that military pilots have to follow when starting up a fighter jet. Before it can be used in the actual mission, a procedure must be developed, tested and validated. Testing a procedure on the FlatSat permits to perform a debug campaign, prepare the team to predict what is considered a nominal behavior of the satellite, and being more efficient at recognizing incorrect behavior.
Moreover, trying a procedure is helpful in understanding the time needed for its completion and give the possibility to became faster maintaining at the same time high reliability of the operation performed.

Using this setup, the tests performed on Margot are the same as the ones performed during GDS tests, with the main difference represented here by the increasing number of Margot Clients connected. In fact, during this test, Margot client is used by front room and back room engineers counting in total up to 25 people connected at the same time.

Step	Event Description	Required Action	Expected Results
1.1	**PURPOSE** To change the On-Board Time (CUC Time)		
1.2	**REQUIRED CONFIGURATION** Radio in TX/RX Mode Uplink and Downlink comms estrablished		
1.3	**SPECIAL OPERATIONAL CONSTRAINTS** S/C not in Safe Mode		
1.4	**REQUIRED INPUT/INTERFACES** The following parameters are required to execture this procedure: • Coarse Time • Fine Time		
2	**Preliminary Checks**		
2.1	Call up display(s)		
2.2	RADIO Demodulator Lock	Verify telemetry: TM001 RADIO_DEMOD_MODE=LOCK	Telemetry confirms S/C is correctly locked
2.3	Radio in TX/RX mode	Verify telemetry: TM010 RADIO_MODE=TX+RX	Telemetry confirms S/C ready to transmit and receive
2.4	S/C not in Safe Mode	Verify telemetry: TMA90 OBC_MODE not in *SAFE_MODE*	Confirmation S/C not in Safe Mode

2.5	Verify Coomms are enabled	Send Telecommand **TC001** "Are you Alive test" Verify Telemetry: *"Are-you-alive connection test report"* TMFC2	Telemetry TMFC2 in the Telemetry Flow Panel confirms connection is working nominally
2.6	Verify current On-Board Time	Check the clock on the main screen of MCS, or, in alternative, the "Generated Time" field in the incoming Telemetry Packets	Indication of the current On-Board Time
3	**Changing the On-Board Time**		
3.1	Set a new On-Board Time	Send Telecommand: **TC067** "CUC_SET_TIME" With Input: **COARSE_TIME=timestamp**	A change in the clock on top of the main screen of MCS, and in the "Generated Time" field of the incoming Telemetry Packets
3.2	Send TT-command to verify execution wrt new OBT	Send Telecommand **TC001** "Are you Alive test" With Input: **timestamp**	Execution of TT-TC at the correct time
4	**Final Status:** **Change of the On-Boad Time** **Ground Link ON**		
	END OF PROCEDURE		

Table 6: Example procedure for the change of the On-Board Time

5.2. Test Results

During GDS and Operations Training information about the system have been collected in order to measure the system performance.
The first analysis is referred to the network usage of Margot Server while the second one is focused on analysing the performance on the machine in terms of resource utilization.

5.2.1. Network

Network bandwidth is a precious resource and it is fundamental that the its usage remains low so that the communication medium is not saturated and excessive response lag from the DHM is avoided. The downlink, representing incoming data on the machine where the server is hosted,

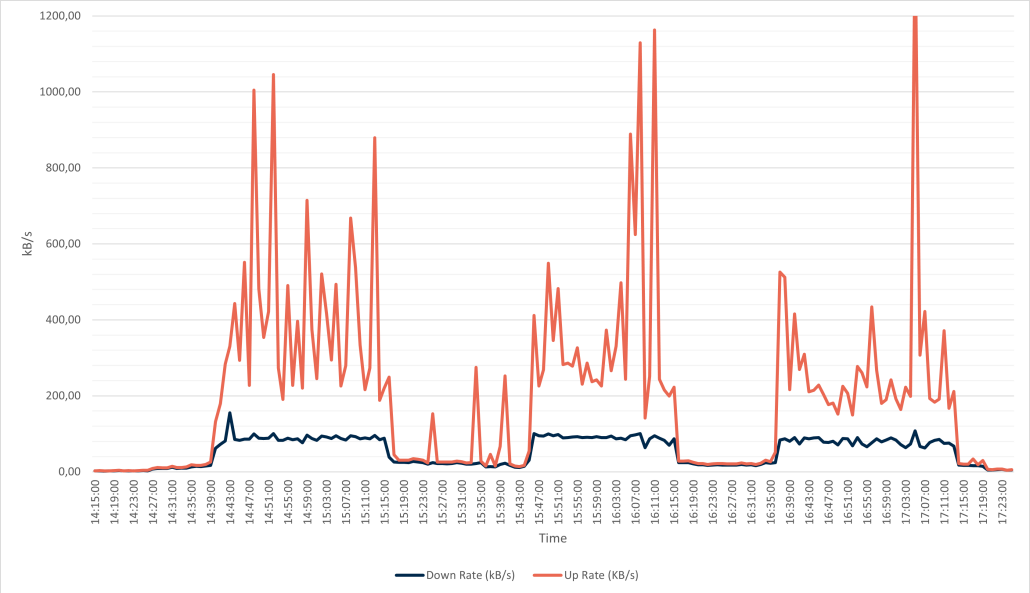


Figure 5.4: Network usage during Operations training

is mainly used by the connection with the Data Provider and requests by Margot Clients. Meanwhile, uplink, representing server outgoing data, is used to send real-time data and historical data responses to Margot Clients. In *figures 5.4* and *5.5* the uplink and downlink data utilization is measured during a training for space operations where it is evident the division between three distinct communication windows. The test has been carried out during a training session with 15 clients connected.
For the downlink the mean communication bandwidth is 88 KB/s instead for the uplink is 300 KB/s that, for modern network, are really low values. These results demonstrate that the system is really scalable in terms of network utilization and it can support the connection of far more clients than the ones needed for a space operation's session. It can be also noted that, between two communication windows, there is still some bandwidth being occupied which is due to ping messages and ping responses between the server and the clients used to maintain the connection

opened when there is no data to be sent or received. Regarding the network utilization through time, for the downlink the bandwidth remain more or less constant with some variations depending on the spacecraft message frequency. Meanwhile, the uplink has some spikes in the data rate which are particularly evident and they are also present between a communication window and another. Those spikes represent historical data requests which require more bandwidth utilization but the tests demonstrate that they never exceeded 1.17 MB/s confirming the network efficiency of the system. Looking at the *figure 5.5* is particularly evident the proportional relationship between the amount of data received by the spacecraft and the one sent out to the clients. For this training session the total amount of data received by the server is 435.7 MB, while for the uplink it is 1.5 GB.

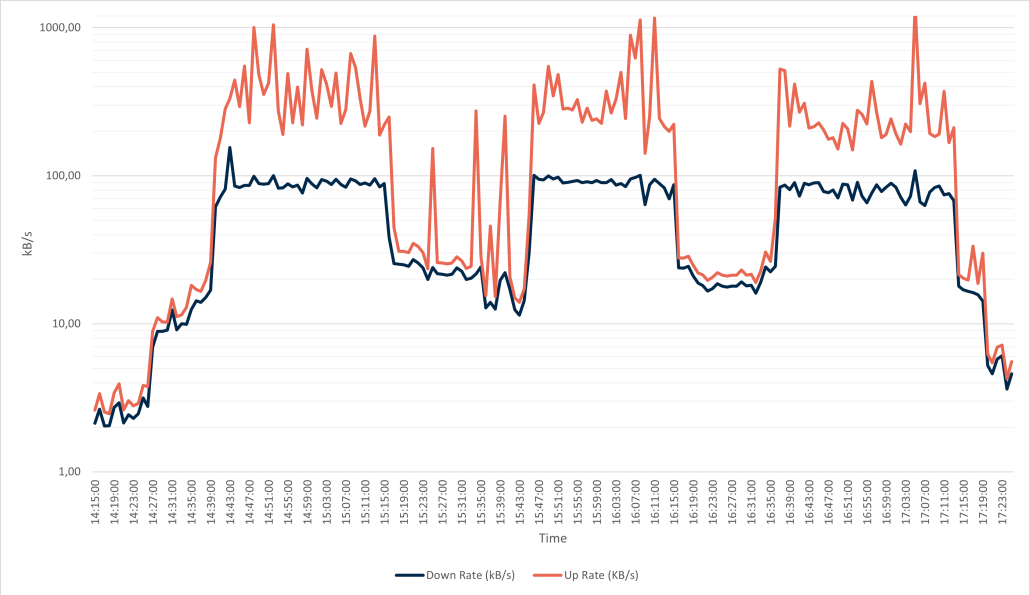


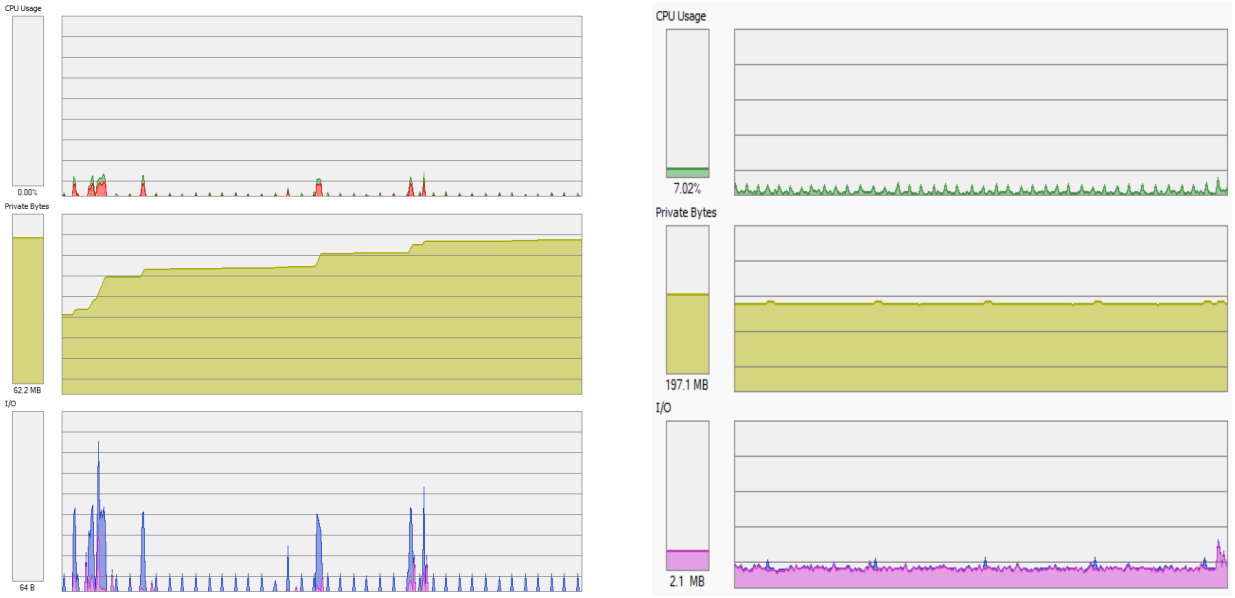
Figure 5.5: Network usage during Operations training using logarithmic scale

5.2.2. Performance

The second analysis is relative to the performances that the server and clients have on their machines through time. The test has been carried out using a half dozen clients connected to a server while receiving telemetry. The kind of data collected for either the server and the client are the CPU usage, the memory occupation, and the I/O activity which in this case is associated with network activity.

In *figure 5.6a* can be seen the results for the server. The CPU usage reaches almost zero percentage with some little spikes in correspondence to historical data requests, but it never exceeds 15 percent of the available computational power. For what concern the memory usage there is a proportional growth in correlation with historical data requests retrieved by the DHM. Although the bytes reserved by the DHM are not deallocated, it has been demonstrated that the memory utilization does not reach critical values during a nominal communication window and this phenomenon doesn't slow down the sever machine by never introducing disk swap operations. Regarding the

I/O a pattern is noticeable representing constant data coming from the spacecraft at regular time intervals and, as also mentioned before, the spikes represent historical data requests from Margot Clients. In *figure 5.6b* can be seen the resource utilization of one Margot Client. The CPU usage



(a) Margot Server (b) Margot Client

Figure 5.6: Performance results

always remains below 20 percent where the spikes are connected to the arrival of incoming data. This result demonstrates that Margot can be used on a vast range of machines with less computational capabilities. For what concern the memory utilization, through all the communication session the amount of bytes occupied never exceed 200 MB, even during historical data requests, avoiding possible RAM exceeding causing disk swap situations. Finally, considering the I/O activity, the amount of data received is proportional to the information coming from the spacecraft and the number of requests made by the user, but it remains below the maximum throughput of a classic wi-fi connections considering 2.4 GHz and 5 GHz frequencies.

6. Further Work

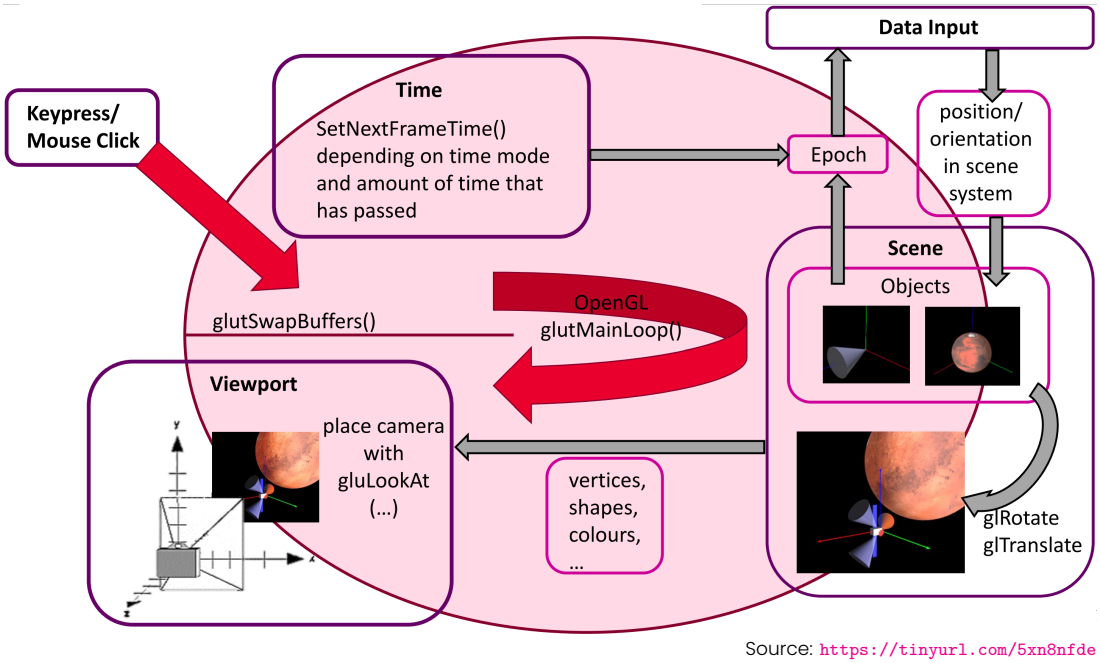


Figure 6.1: 3D-Visualisation of Flying Satellites - Step By Step

During these months of development, Margot has improved FCT's productivity during spacecraft operations, demonstrating that it can give a solid contribution in data monitoring and analysis. It has become a fundamental tool during training by allowing the FCT to develop displays capable of showing in real time the behavior of the spacecraft simulator and easily "zapping" between telemetries really fast.

But there is still a lot to do and improve. A good starting point, which came out from the test and validation phase, is the optimization of the amount of resources locked by the server. The objective is to keep the mean value of the private bytes (referring to the *figure 5.6a*) as constant as possible. The DHM source code has already been analysed in order to find when the program should release data. The outcome of this analysis is that the Microsoft SQL connector retains data inside the SQL cursor, but it is not yet clear how to dispose of its resources so further effort must be carried out.

Standard telemetries, taken from the MIB, are not always enough and there are some synthetic data that can be provided. A good example is the download percentage of images coming the spacecraft payload or unit/time conversions of the on-board time. Testing Margot during operations training allow receiving a lot of feedback from colleagues. From these sessions, the necessity of having a dedicated section where telecommands can be visualized came out as well as the possible integration of Margot together with Argotec's Mission Planning Tool. The MPT can manage the communication schedule saved in the database and update it accordingly during the mission. Also, the MPT can benefit from some real time telemetries like the battery current and

voltages which can be used to precisely calculate communication windows duration before the battery need recharging, or the spacecraft operating mode telemetry to quickly know when it is in communication mode. Furthermore, Argotec's FCT is evaluating the introduction of a real-time rendering of the spacecraft in space following attitude and orbital attitude telemetries [7]. The objective is to use a graphical engine (like OpenGL, Unreal Engine or Unity) that runs in a loop (*figure 6.1*) and constantly update the spacecraft model inside the environment represented in this case by the solar system. This functionality not only allows to have a better understanding of the spacecraft orbit and attitude but also allows, with the introduction of virtual cameras, to simulate and predict cameras framing that, for the scenarios considered here, LICIACube and ArgoMoon, are represented by the payloads. The implmentation can be integrated with a predictive model to be used when the spacecraft is not in communication and, by calculating the accumulated errors between two communication sessions, helping Argotec's team fine-tuning the system parameters for better predictions.

7. Conclusions

With this thesis work, a web-based ground segment software has been developed to support Argotec's ground operations, taking the ArgoMoon and LICIACube missions as case studies. The requirements identified as fundamental have all been achieved by allowing the user to display and analyse telemetry in real time and non-real time scenarios. The software can store data into a dedicated database and can retrieve this information when requested, also it can withstand work loads coming from dozens of clients at the same time. To validate the project objectives Margot elements were tested, and the results analysed through test with hardware in the loop. The remaining requirements and future developments will help to implement other features in the framework.

It should be noted that this work can be used not only for the missions reported in this thesis, but also for future activities. Using external files, such as configurations file described in the previous chapters, will allow the initialization of Margot without changing the source code. At the time of writing, the LICIACube mission is on its journey to Dimorphos, the opportunity of testing Margot on a real-time communication is closer every day.

References

[1] Steven Bellovin. Guidelines for Specifying the Use of IPsec Version 2. RFC 5406. Feb. 2009. DOI: 10.17487/RFC5406. URL: https://rfc-editor.org/rfc/rfc5406.txt.

[2] Carolyn E. Connolly Thomas M.; Begg. Database Systems – A Practical Approach to Design Implementation and Management (6th ed.) Pearson, 2014.

[3] "Deep Space Network". In: https://it.wikipedia.org/wiki/Deep_Space_Network (2021).

[4] U. Erikson. "Functional vs Non Functional Requirements". In: https://www.analyticsvidhya.com/blog/2020/05/artstory analytics-data-science/ (2021).

[5] Eclipse Foundation. "Dynamic Security Plugin". In: https://mosquitto.org/documentation/dynamic-security/ (2021).

[6] Vandermolen R Fulton R. Airborne Electronic Hardware Design Assurance: A Practitioner's Guide to RTCA/DO-254. CRC Press, 2017.

[7] Lena Fürnstall. "3D-Visualisation of Flying Satellites – Design, Implementation and Surprising Use Cases - FDSSO". In: 2021.

[8] "IEEE Standard for System and Software Verification and Validation". In: IEEE Std 1012-2012 (Revision of IEEE Std 1012-2004) (2012), pp. 1–223. DOI: 10.1109/IEEESTD.2012.6204026.

[9] ISO, ed. Information Technology—Open Systems Interconnection—Basic Reference Model: The Basic Model. 2nd ed. International Standard. Vol. ISO/IEC 7498-1:1994. 1994.

[10] Schatz - Daniel - Bashroush - Rabih - Wall - Julie. Towards a More Representative Definition of Cybersecurity. Journal of Digital Forensics Security and Law, 2017.

[11] "MATLAB". In: https://en.wikipedia.org/wiki/MATLAB (2021).

[12] "MQTT Version 5.0 OASIS Standard". In: https://docs.oasis-open.org/mqtt/mqtt/v5.0/os/mqtt-v5.0-os.html (2021).

[13] NASA. OpenMCT. GitHub Repository.

[14] Sean Parent. "Better Code Concurrency". In: https://vorbrodt.blog/2019/02/26/better-code-concurrency/. Ed. by NDC London. 2017.

[15] Paul Ralph and Yair Wand. "A Proposal for a Formal Definition of the Design Concept". In: Design Requirements Engineering: A Ten-Year Perspective. Ed. by Kalle Lyytinen et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 103–136. ISBN: 978-3-540-92966-6.

[16] CCSDS - Consultative Committee for Space Data System. Space Link Extention - Internet Protocol for Transfer Services.

[17] "TCL Developer Xchange". In: https://www.tcl.tk/ (2021).

[18] “Thread Pool”. In: https://en.wikipedia.org/wiki/Thread_pool (2021).

[19] Michael Schmidhuber homas Uhlig Florian Sellmaier. Spacecraft Operations. Springer, 2015. ISBN: 978-3709118023.