Ground Station Distribution and Automation for Low-Earth Orbit Cubesats

Rafael Paixão Branco
rafael.p.branco@tecnico.ulisboa.pt

Instituto Superior Técnico, Lisboa, Portugal

November 2022

Abstract

It is not possible to maintain a persistent connection between a low Earth orbit satellite and a ground station. For this reason, different techniques are employed to increase the connection availability, and data throughput. In this work, the ground segment for the ISTSAT-1 is studied, its architecture revised and improved, to ease its distribution, automation, and operation. While the suggested improvements and solutions are directly applicable to the ground segment of ISTSAT-1, they should be general enough to be employed on other ground segments.
Keywords: Distribution, Automation, Graphical User Interface, Satellite

1. Introduction

Nowadays many satellites orbit our planet and while their missions diverge from one another, most of them need to broadcast information and receive commands from Earth in order to fulfil their missions. This information can, typically, be subdivided into mission related data, telecomands, and telemetry which allows the detection of possible anomalies with the spacecraft.

To allow this communication between the two parties a ground segment is needed. A ground segment is the part of the satellite's communication system that resides on the ground [4], allowing the monitoring and control of the spacecraft from Earth. The structure of this segment varies according to the satellites they are supporting. However, every ground segment needs at least one ground station which provides a physical layer interface to communicate with the satellites. In order for ground stations to be able to transmit and receive data, they need a line of sight (LOS) to the target spacecraft. Therefore, the connection availability is limited by the satellite's orbit and the ground station's location.

Traditionally, when a satellite is in LOS, satellite operators are entrusted with the task of sending the appropriate commands to achieve some previously stipulated goal. However, recently, there has been an increase in the automation of repetitive tasks, not only to free staff for more important endeavours but also to reduce the costs associated with the mission.

1.1. Motivation

Today a great number of satellites are launched into low Earth orbits (LEOs), orbits normally at an altitude of less than 1000 km above Earth. The low altitude of these orbits allows high data bandwidth and low latency communications. However, satellites in LEOs travel at a speed of around 7.8 km per hour, taking approximately 90 minutes to circle Earth [3], nearly 16 times the rotational period of Earth. Therefore, resorting to one LEO satellite and a ground station, it is not possible to persistently maintain a LOS and, consequently, a connection between them.

When operating a LEO satellite, one has to be aware of when the spacecraft is above the horizon and thus accessible for communication. This is known as a pass. The number of passes, as well as their temporal distribution, varies according to many factors such as the orbit's altitude and inclination, and the ground station's location.

Communication is central to a satellite's mission success. Therefore it is important to employ techniques to increase the number of available passes and their throughput. This problem needs to be addressed by most LEO satellites and the ISTSAT-1 is not an exception.

ISTSAT-1 is a cubesat (1U) that will orbit Earth at 550km of altitude, in a sun-synchronous LEO. Its mission is to validate that a small satellite, with a planar antenna, and a general microprocessor (instead of a Field-Programmable Gate Array (FPGA) board) is able to collect Automatic Dependent Surveillance-Broadcast (ADS-B) messages sent

by planes [1]. In order to accomplish its mission, the spacecraft needs to be supported by a ground segment.

The ground segment of ISTSAT-1 was developed alongside the spacecraft, while its main task was to support the satellite's needs and test it. Now, that the spacecraft is near launch, it is necessary to audit the state of its ground segment, assessing flaws, suggesting improvements, and implementing solutions. While the suggested improvements and solutions are directly applicable to the ground segment of ISTSAT-1, they should be general enough to be employed on another mission's ground segments.

1.2. Objectives and Contributions

The main contributions of this work were the reduced latency, increased goodput, Graphical User Interface (GUI) design and implementation, and the ability to schedule operations. These contributions were motivated by the objectives extracted from the analysis of the initial ISTSAT-1's ground segment.

Since the ground segment was developed at the same time as the spacecraft, its structure grew organically to support the needs of the satellite. This resulted in design decisions that poorly influenced latency and goodput. Thus, one of the objectives was to improve both metrics through the modularization of the ground segment.

During the evolution of the ground segment, operators resorted to a Command Line Interface (CLI) to interact with the spacecraft. While this decision suffices for the development process, it is important to create an User Interface (UI) easier to use. Therefore, another objective was to improve the ground segment so that its operation becomes more user friendly.

Throughout the development of the satellite, it endured long lasting tests. However, these tests did not require any scheduling, they were simply left running. This motivated the objective of employing a system capable of scheduling operations.

2. Related Work

In order to properly design a ground segment for a LEO satellite, it is important to research about how ground station distribution, ground segment automation, and interfaces are typically handled. This investigation aims to provide insight about solutions that can be applicable to problems the ISTSAT-1 may endure.

2.1. Ground Station Distribution

Ground stations are responsible for acting as a gateway between the ground and space segments. A connection between them is established through the transmission and reception of electromagnetic waves which need to be modulated and demodulated in order to allow the exchange of data between both parties.

For a ground station to be able to communicate with a satellite, it must be in LOS. However, for LEO satellites, this only accounts for a relatively short period of time. Therefore, controlling just one ground station might not fulfil the satellite's mission data budget. There are two main approaches to increase communication availability.

One technique is to increase the number of ground stations accessible to the ground segment. These are referred to as Ground Station Networks (GSNs).

Another way to solve this issue is by relaying data through a pre-existing network formed by satellites in a constellation. Typically referred to as Space Relay Networks (SRNs). However SRNs require the installation of specific hardware and/or software prior to launch in order access these networks.

ISTSAT-1 does not contain the components required to take advantage of SRNs, therefore it is only possible to increase communication availability through ground segment solutions.

In recent years, with the reduction of LEO satellite's launch prices, there was an increase in the demand for readily available ground stations and companies started providing Ground Stations as a Service (GSaaSs). These services allow a client to rent ground stations from a GSN for some period and only pay for the scheduled time.

2.2. Ground Segment Automation and Interface

Traditionally the ground segment is operated by humans, known as ground operators. They are responsible for monitoring and controlling the state of the satellite. To achieve this goal, they analyse the received telemetry and mission data, devise a strategy to reach the target state, and according to that plan, issue commands to the satellite.

Since it is only possible to communicate during a pass and that passes are typically short for LEO satellites, the time it takes a ground operator between the analysis, planing and commanding phase is critical to the amount of work it is possible to produce during each pass.

This limitation can be mitigated by automating reactions based on received satellite data, and by providing ground operators appropriate interfaces. The first measure ensures ground operators do not have to execute repetitive tasks, which reduces the probability of operation errors and increases data processing and reaction time. While the second, guarantees that when a ground operator is required to interact with the satellite, they can do so efficiently.

There are many ways of quantifying the automation level of a ground segment. One common approach is to rate them according to the light level of the operations room [9]: lights on, refers to a manual system; lights dim, to a semi-automated system,

where some of the operations are automated; and lights off, defines a system where almost all functions are automated.

2.2.1 Mission Operation software

The software that enables the ground segment has a crucial role in the success of the mission it supports. It is typically composed of multiple systems that interact with each other, requiring a big effort and investment to develop. In order to mitigate these costs, it is usual to resort to generic solutions, which are developed taking into account common needs across multiple satellite missions.

Major Tom Major Tom is the cloud based command and control solution provided by Kubos. In addition of providing an interface to send commands to the spacecraft and display the received data, it also affords the possibility resorting to the GSNs pre-integrated into their system for communication.

3. Ground Segment of ISTSAT-1

During the development phase of ISTSAT-1, the primary focus was on the satellite development. Therefore, typically, the feature implementations in the ground segment were motivated as a way of testing new satellite functionalities.

The organic growth of the ground segment around the spacecraft's needs led to an unstructured design, whose architecture and flaws will be exposed. After evidencing the shortcommings of some of the approaches taken, a solution will be proposed.

3.1. Initial Architecture

The ground segment is divided into the modules shown in Figure 1. The coloured blocks represent hardware systems while the white blocks represent relevant processes running inside them. The solid links between blocks represent Transmission Control Protocol (TCP) connections while dashed links represent Universal Serial Bus (USB) communications.

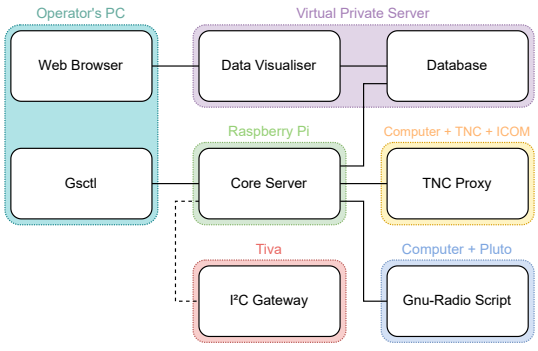

Figure 1: Ground Segment Architecture

3.1.1 Protocols

Typically the ground segment communicates with the satellite via radio, but, during the development and testing phases, it is also possible to communicate with each ISTSAT-1 subsystem through a serial bus.

The protocols used for radio communication between the ground and space segments are ISTNanosat-1 Command Protocol (INCP), Reliable Data Protocol (RDP)[8], Cubesat Space Protocol (CSP)[6], and Amateur X.25 (AX.25)[2]. INCP is the application level protocol, used to issue communicate with the spacecraft.

When communicating though the serial interface, the INCP messages are directly sent in via the I^2C bus to the target subsystem, easing the development and testing of the spacecraft.

3.1.2 GSCTL

GSCTL is a client program with a CLI. It is the interface between the operator and the system, providing access to the functionalities exposed by the Core Server.

Using this UI it is possible, not only, to command the satellite, but also to run built-in operations. Operations is the designation used to define a dynamic flow of commands whose behaviour depends on the satellite responses. The available operations cover the most used scenarios. Currently these operations are mostly used to automate functional tests on the spacecraft.

This interface, while effective for proficient operators due to its flexibility, can intimate those trying to learn to operate the spacecraft as it requires the user to constantly recall the names of commands, steepening the learning curve. On the other hand, GUIs organise information in a way that facilitates its mastering. They also account for the majority of UIs, therefore, users are typically more comfortable using them. For these reasons, it is important to create a GUI that enables the ground segment operation.

3.1.3 Core Server

The Core Server exposes an Application Programming Interface (API) that enables client programs to operate the satellite. This module also processes mission and telemetry data received from the spacecraft, storing it persistently in a Database. To send information to the satellite, INCP messages need to be generated and formatted according to each gateway specification. When data is received, the inverse process has to take place.

Gateways are nodes that form a passage between two networks operating with different protocols. In the specific case of ISTSAT-1, there are two gateway

types, ground stations and the Electrical Ground Support Equipment (EGSE).

3.1.4 EGSE

EGSE is an integrated system of electrical testing solutions to ensure that the satellite is operating according to the specification.

In the specific case of ISTSAT-1, the EGSE can be used to program the satellite and to interface with the spacecraft's I^2C bus. The EGSE Gateway is the module responsible for receiving INCP messages via USB and sending them to the satellite's I^2C bus.

3.1.5 Ground Station

Each ground station is comprised of at least a radio, an antenna, and a controller that provides an API to exchange data with the satellite. The ground segment has one ground station using a traditional radio transceiver and another using a Software Defined Radio (SDR). Both ground station controllers only allow the sending and receiving of frames. It is not possible to change modulation or the pointing of the antenna programmatically.

To communicate using any of these ground stations, the operator must request the Core Server to establish a TCP connection with the appropriate controller. Since the Internet Protocol (IP) address and port of the service running in the ground station controller are necessary to open a TCP connection, and the operator cannot directly command the Core Server, this is usually done using GSCTL.

This approach is not ideal, as the ground operator needs to know when each ground station is up, and what is its IP address. If the ground station is not hosted on the same network as the Core Server, it might be necessary to configure the ground station's network to allow foreign connections. However, depending on the network, it may not be possible to configure it in such way.

Traditional Ground Station In this ground station the controller receives the address, control, and information fields of the AX.25 frames via the TCP socket. This information is then sent to the Terminal Node Controller (TNC) which frames it according to the High-Level Data Link Control (HDLC) protocol, modulates them using Audio Frequency-Shift Keying (AFSK), and sends the audio to the radio transceiver. The transceiver then frequency modulates the audio signal into the Radio Frequency (RF) band and transmits it using the antenna. When receiving information, the reverse operations are done by the same ground station components.

SDR Ground Station Differently to how the previously discussed ground station operates, this controller receives the AX.25 frames. It then, processes and forwards them to the SDR. The SDR ensures that they are transmitted using the antenna. As with the traditional ground station, when receiving information, the reverse operations are done by the same components.

Due to the nature of SDRs, it is possible to change the used frequency modulation. This is defined by the GNU-Radio script running in ground station controller.

Ground Station Interface The approach defined in both ground station descriptions establishes a RDP connection between the Core Server and the satellite. Since the RDP connection is established with the Core Server and not with the Ground Station, the throughput of the network is influenced by the distance between these nodes, as every RDP segment needs to be sent through a TCP tunnel, eventually crossing many internet nodes.

3.1.6 Gateway Interface

The Gateway interfaces to send INCP commands to the satellite are not standardised. This lack of standardisation is notable not only trough the distinct utilized connection types, but also from the used data formats. The Core Server resorts to TCP when establishing a connection with either ground station and to USB when connecting to the EGSE gateway. The data formats used are AX.25 fields, AX.25 frames or INCP messages depending on whether the target gateway is a traditional radio ground station, a SDR based ground station or a EGSE gateway, respectively.

This design decision is tied to an increased program organization complexity, leading to less maintainability. It also results in the unnecessary proxying of data affecting the goodput between the Core Server and the satellite.

3.1.7 Automation

In order to execute a satellite operation, a connection between a GSCTL instance, the Core Server, a gateway and the satellite must be established. This connection is required since the automation's control flow is decided in the GSCTL.

Even though the capabilities provided by GSCTL can be leveraged to achieve lights-out operation, the process to reach such goal is not practical nor robust. As an example, the operator could schedule the operation to run in their computer during a satellite pass. However, this would require their computer to be on, and connected to the internet at the specified time and during the full pass duration.

It would be preferable to allow the schedule of operations directly in the Core Server, entrusting

it with the chore of executing the appropriate operation at the correct time. This solution is more robust as it reduces the number of points of failure.

3.1.8 Data Visualiser

The data visualiser complements GSCTL, allowing the operator to visualy analyse the telemetry and mission data stored in the database. This is achieved through a web based GUI that enables the creation of different panels, according to the data organisation.

3.1.9 Security

Connections between the Core Server and ground stations or GSCTL happen through TCP sockets as such all the data exchanged between both parties is in plain text.

3.2. Proposed Architecture

While re-designing the ground segment of ISTSAT-1, the initial architecture was taken into account trying to reuse most of the components in order to ease the development efforts. The proposed solution aims to solve the exposed issues with the previous design. Figure 2 depicts the proposed changes to the ground segment architecture.

The Core Server should be hosted in a dedicated machine, along with the Data Visualiser and database, providing an increase in processing power and availability to the service.

Each gateway should initiate the connection process with the Core Server, ensuring that there is no need for the ground operator to monitor the state of these connections. As soon as the system is available, it's controller will establish a connection with the Core Server. This controller also exposes the system's API.

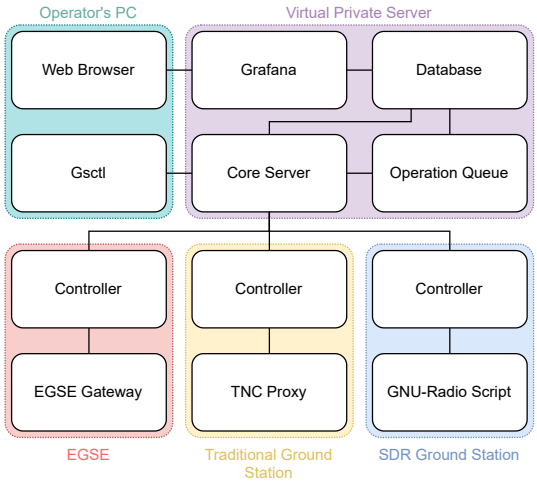

Figure 2: Ground Segment Architecture

3.2.1 Communication Protocols

Instead of the previously exposed heterogeneity in the message format accepted by each gateway, every gateway handles INCP messages.

3.2.2 Stand-alone Electrical Ground Support Equipment (SEGSE)

The presented architecture assumes that there is always an Internet connection. There are cases where this assumption might not be correct, preventing the operation of the satellite. Therefore, it is crucial to define a system that still allows the debug of the satellite even without an internet connection.

This system is referred to as SEGSE. It is comprised of a general purpose computer and a microcontroller board, connected via USB. The computer is running the Core Server, EGSE Controller, Data Visualiser, and a local database. While the microcontroller is running the EGSE Gateway.

In order to operate the SEGSE, the operator's computer is linked to the computer directly, using an RJ45 cable to connect both devices via Ethernet.

3.2.3 Operation Queue

The proposed design keeps the functionality that allows GSCTL to execute operations, while allowing those same operations to be scheduled in the Core Server. This is achieved by the Operation Queue, a software module, that ideally runs in the same machine as the Core Server and is responsible for establishing the connection between them.

To schedule an operation, the operator, notifies the Core Server about its name and desired timestamp for the execution. Then, if the Operation Queue is connected, the Core Server forwards it the request. The Operation Queue registers the information regarding the execution of this operation, and saves it in the database. As soon as the time for execution arises, the Operation Queue executes the desired operation, behaving like a GSCTL instance.

3.2.4 Graphical User Interface

In order to incorporate a new UI into this ground segment, there are two main approaches. One is to create a program that runs on the operator's computer and directly calls the endpoints exposed by the Core Server's API. This is the technique employed by GSCTL. Another approach would be to host a service, like a web server, that provides a UI, translating user requests into calls to the Core Server API.

4. Implementation

At the time of implementation, the satellite was at the end of the final test campaigns. As such, mod-

ifications to the current design took into account the already written test procedures and, where possible, respected them, avoiding changes to already approved documents. That being said, the proposed solution was implemented in two distinct phases.

During the first phase, the ground segment's architecture was altered to encompass the necessary changes to the working features. These changes take priority over the implementation of new functionality because they may invalidate already written procedures. Solving these issues first, ensures that if backwards compatibility is broken, there is more time to correct the affected documents. It also means that the documents that still need to be written will already take into account the new architecture.

The second implementation phase, consisted on adding new functionality to the ground segment. A GUI was included and the automation process improved with the development of the Operation Queue.

4.1. Core Server Interfaces and Gateway Behaviour

Core Server's gateway interface had to be modified to allow the sending and receiving of INCP messages, as well as the commanding of each gateway, depending on its capabilities. Ideally this traffic would be sent over one connection, reducing the number of connections, and thus the load on the server.

The Core Server, as the vast majority of the ground segment, is implemented in the Python[1] programming language, and uses the Trio[2] library to manage asynchronous input and output.

The current client API provided by Core Server resorts to Trio-RPC. Trio-RPC is a Remote Procedure Call (RPC) library, developed in-house, that takes advantage of Trio in order to process requests asynchronously. However this library only allows communication through a traditional client-server model, where the client sends a request and the server answers with a response. Although this approach is acceptable in the case of GSCTL, it is unacceptable in this particular circumstance, as gateways must be able to push and receive INCP messages as soon as possible.

Since Trio-RPC was developed in-house, it was altered to allow servers to call functions registered in the client, thus enabling it to be used for in this scenario. It was also improved to support Transport Layer Security (TLS) and execution of multiple requests concurrently.

4.1.1 Trio-RPC

This library allows the execution of asynchronous remote functions. The most important component of this system is the class RPCNode. It enables the sending and receiving of requests and responses, as well as their handling. Since the interface with this class provides a lower level interface than most applications will need, classes to abstract the intricacies of the system were developed.

These classes are named RPCClient and RPCServer. While both expose methods to register functions that can be remotely called. The RPCServer contains a method to start waiting for new connections, and the RPCClient one for establishing them.

Whenever a connection is established between a RPCServer and a RPCClient, they start a new task to handle it. This task is controlled by an instance of the class RPCNode, lives for the duration of the connection, and starts at least two child tasks, an handler and an executer. The number of executor tasks is the same as the number of concurrent requests the RPCNode is able to process.

Each instance of RPCNode exposes a method that allows the execution of functions in its remote counterpart. This is done by sending a request message to the connected node, which processes it and responds with a response message. Messages follow the javascript object notation (JSON) format.

Concurrency In order to handle requests concurrently, each RPCNode makes use of executor tasks. This implies that there are a maximum number of requests that can be executed concurrently.

Another solution would be to create a new task per request. However, the time it takes to launch a new task would increase the processing time of each request. This solution also allows a malicious client to spawn an arbitrary number of tasks, eventually reducing the available time to handle other requests. This attack is known as a denial-of-service.

Syntax The way a user interacts with a library greatly influences its usability and adoption. This library's API tries to accommodate most use cases by providing a generic client and server implementation. While this implementation should fit most use cases, it is expected that some applications may require changes to the default behaviour. For this reason, its inner workings are exposed, allowing the user to implement their own version of either the client or server, to better suit their needs.

Security Both the RPCClient and RPCServer support TLS encryption and peer authentication facilities. To resort to these features, the appropriate SSLContext should be given as an argument to the "connect" method of RPCClients and to the constructor of RPCServers.

[1] `https://www.python.org/`

[2] `https://trio.readthedocs.io/en/stable/`

Backwards Compatibility The interface to invoke remote calls has been kept unchanged. When porting code using the old version of this library, the only sections requiring intervention are related to the RPCClient's connection process.

4.1.2 Core Server

The Core Server exposes two endpoints, the previously defined client API, and a new API to handle incoming connections from gateways.

The client API allows client programs to issue commands to the satellite, while the gateway API enables the handling of gateways. When a gateway establishes a connection with the Core Server, it must provide its identifier and capabilities. The identifier is the label used to uniquely identify each gateway. The capabilities are a list of functionalities available for the connected gateway. Since different gateways might provide different functionalities, it is necessary to inform the Core Server about which functionalities are exposed by which gateway. The available capabilities are "gateway", "ground_station" and "tiva". Each gateway can implement any combination of the three.

The "gateway" capability ensures that the gateway allows remote calls to the function "send" that sends INCP messages, and that as soon as it receives an INCP message from the satellite, it pushes it to the Core Server by invoking the function "deliver_incp_message".

Unlike "gateway", the "ground_station" and "tiva" capabilities do not notify the Core Server as soon as some event happens. Instead they just handle requests made by the Core Server. The "ground_station" capability exposes functions related to ground station control, while "tiva" implements functions to manage the Tiva-C device utilised by the EGSE to debug the spacecraft.

When defining a new function to be invoked through the client API, the developer needs to annotate which capabilities it requires. This allows Core Server to inform the client when they issue an API call using a gateway that does not expose all the required capabilities to execute the desired function.

4.1.3 Ground Station

This gateway implements the "gateway" and "ground_station" capabilities. It is composed of three main software modules, the controller, GNU-Radio script and modulation changer.

The GNU-Radio script resorts to a SDR to communicate with the satellite via radio. The sending and receiving of messages to this script happens through a TCP connection, exploited by controller to transmit and receive AX.25 frames to and from the spacecraft. In order to change the modulation in use, the running GNU-Radio script must be stopped and a new one started. This is done via the modulation changer, that exposes a TCP socket allowing an external program to stop the currently running script and start a new one. The modulation changer also detects if the modulation script running has crashed and restarts it. The controller takes advantage of the modulation changer to allow the Core Server to change the modulation used by this ground station.

4.1.4 EGSE

This gateway implements the "gateway" and "tiva" capabilities and is composed of a single software module, the controller. This module interacts with the Tiva-C and allows the debug of the spacecraft.

4.2. Graphical User Interface

Instead of implementing an in-house interface, the cloud based command and control solution, Major Tom, was integrated into the ground segment. This decision aimed to reduce the development and maintainability efforts while allowing the possibility of utilising their pre-integrated GSNs.

As a bonus, Major Tom, provided an academic program, where it allowed university projects to take advantage of their service free-of-charge.

When integrating Major Tom into a new ground segment, the only required module is a gateway. Based on their notation, a gateway is a software component that receives messages in JSON and translates them into a format capable of being interpreted by the satellite. This gateway is responsible for establishing the connection with Major Tom, and the data it produces is forwarded either to Major Tom, in order to be sent using an integrated GSN, or to a specific gateway directly connected to the Core Server.

Instead of altering the Core Server to establish connections with Major Tom, a new software module, Major Tom Translation Layer was created. This module establishes a connection between Major Tom and the Core Server, and translates the communication between both. This decision allowed the integration of Major Tom, without requiring changes to the already stable Core Server, by leveraging Core Server's client API (the same used by GSCTL).

The process of integrating Major Tom into the ISTSAT-1's ground segment can be decomposed into two parts, the command definition, and the development of Major Tom Translation Layer.

4.2.1 Command Definition

In order for Major Tom to provide a GUI capable catering to the different needs of multiple missions,

it needs to be configurable. The process of configuring Major Tom's UI is achieved through the upload of JSON objects to Major Tom's website. These objects, refered to as command schema, define the commands that are available for a specific satellite or ground station.

ISTSAT-1's commands Communication with the satellite happens through the INCP protocol. The ISTSAT-1 is composed of subsystems and each subsystem has different data variables, reports, configurations, commands, and enumerations. These are described in a YAML Ain't Markup Language (YAML)[3] file that is processed to generate subsystem specific libraries, defining the available instructions.

Data variables represent the value of a metric at some point in time. Reports are groups of data variables, they allow the request of multiple data variables in a single call. Configurations are parameters that determine the behaviour of the satellite, and can be altered in runtime. Commands are parametrisable procedures that execute on a subsystem.

The Core Server provides, through its client API, functions to interact with each spacecraft subsystem. These functions are "data_req", that requests the value of some data variable or report, "config_get" and "config_set", that, respectively, get and set the value of some configuration, and "cmd", which issues a command. These procedures are the basic building blocks that enable all satellite operations.

In order to create the full command definition schema, a program was written to parse the INCP description file, and extract the required information. The command definition schema was then uploaded to Major Tom in order to customize its GUI to the mission requirements.

4.2.2 Major Tom Translation Layer

After successfully configuring Major Tom, a service to establish a connection between their servers and Core Server still needed to be developed. This service is the Major Tom Translation Layer. It establishes a WebSocket[5] connection with the Major Tom's servers, allowing them to send the requests issued by the client via their GUI. When it receives the issued command, translates it, and calls the appropriate function through Core Server's client API. As soon as response is received from the Core Server, it is translated to Major Tom's format and sent.

Telemetry As is, this module already enables the utilisation of Major Tom's UI to communicate with the spacecraft. However, the Grafana interface integrated into Major Tom is only being populated when responses to commands, issued from this client, are received. As such, telemetry data is also not being used to populate this Grafana's database.

In order to solve this issue, a new command was created in the Core Server, enabling a client to request to be notified as soon as new satellite data is received. From this moment on, whenever the Core Server receives data variables or reports from the spacecraft, it forwards it to all the interested clients. To implement this behaviour, it calls the function "on_data" on the clients that wish to be notified.

The Major Tom Translation Layer was then changed to request the Core Server to be notified when data is received from the spacecraft. When this data is received, it is translated into the Major Tom format and sent to their servers to be stored in their database.

4.2.3 Impediment

Despite the working condition of the implemented GUI, due to contractual changes, Major Tom stopped providing their software free-of-charge for university projects. As such, this command and control service is no longer a viable solution for the ISTSAT-1's ground segment.

4.3. Alternative Graphical User Interface

Since the previously integrated UI stopped being viable, a new GUI was developed, in-house. The decision to develop a new interface, instead of integrating another one into the ground segment, was motivated from the experience with Major Tom. Integrating a general command and control mission software into the ground segment does not provide the ideal flexibility nor customization levels, and entails the loss of control over the decisions made by the software provider. By developing the interface internally complete control over these factors is guaranteed.

The developed interface was a web server. It runs in the same machine as the Core Server and the operator accesses the interface through its browser.

In order to ease the development process, this module was programmed in Python using the pglet[4] framework. This framework allows the creation of GUIs through the use of pre-built components.

This UI focuses on providing a simple interface to ping, request data, edit configuration values, and issue commands to the satellite. Throughout this document, this interface will be referred to as "Smooth Operator".

4.3.1 Interface

The GUI is subdivided into two parts, the navigation bar, on the left side, and the operational interface on

[3] https://yaml.org/

[4] https://pglet.io/

the right. Figure 3 displays the interface to request data variables and reports from the satellite.

Figure 3: Smooth Operator's Data Request Interface

Each page contains a response section at the bottom of the operational interface where the last issued request is displayed. The "History" tab contains a list with information about all the requests issued in this session.

4.3.2 Security

Pglet provides built-in support for authentication through the sign in with a GitHub, Google, or Microsoft account. This functionality was used to block unwanted access to the application.

However, traffic between the client's browser and the web server still happens over an unencrypted connection. During the deployment phase, this needs to be taken this into account, proxying the data through a TLS termination proxy.

4.4. Operation Queue

The Operation Queue was developed to enable the scheduling of operations. When this module boots, it automatically establishes a connection with the Core Server's client API. After the connection is established, it notifies the Core Server that this client is an Operation Queue.

When another client wants to schedule an operation, it issues a remote call to the Core Server providing the operation name, and the date and time for when this operation needs to take place. The Core Server then proxies this message to the connected Operation Queue that registers it in its queue of operations and waits for the next timestamp. When it is time to execute the operation, the Operation Queue, runs it.

5. Validation

ISTSAT-1 is part of European Space Agency (ESA)'s Fly Your Satellite! (FYS)[7] programme. As such, in order to launch the spacecraft, it has to endure a panoply of tests. Some of these tests, while assessing the functionality of the satellite, also test the ground segment, as they require a close interaction between both parties.

At the end of the first development phase, both the satellite and the ground segment, enrolled on a testing campaign (Mission Test (MT)), aimed at verifying that the ISTSAT-1 is able to perform its intended mission and withstand contingency scenarios. Since the satellite needs the ground segment to be able to complete its mission, by extension, it was also tested.

After the Mission Test (MT), the satellite enrolled on another test campaign, the Vibration Test (VT), that intended to verify that the spacecraft can withstand the random vibrations expected for the launch phase. Despite the ground segment not being the main target of this test campaign, it can once again be used to validate its behaviour.

5.1. Mission Test

In order to verify that the satellite is able to fulfil its mission, this test simulated the interactions with spacecraft during its first orbits.

Both the nominal and contingency operation phases were operated remotely, due to COVID-19 restrictions.

This test campaign took place in Instituto de Engenharia de Sistemas e Computadores - Microsistemas e Nanotecnologias (INESC-MN)'s clean room. To perform this test, all the required equipment to operate the satellite had to be transported to this location. The necessary equipment was the EGSE and the SDR ground station.

5.2. Vibration Test Campaign

During this test, the spacecraft was supposed to be vibrated on its three axis, assuring that it will not be damaged during the launch. Before and after vibrating the satellite, it had to be subject to an Reduced Functional Test (RFT), to ensure that the vibration did not harm the spacecraft.

This test campaign took place in the CubeSat Support Facility (CSF), an assembly integration and testing facility for CubeSats, located at the ESA Education Training Centre, based at the ESEC-Galaxia facility in Transinne, Belgium. As with the MT, the EGSE and the SDR ground station were transported to the site, because they are required to execute the RFT.

The ground segment architecture followed was the same as the one used in the MT.

5.2.1 Challenges

After configuring both gateways to connect to the local Wi-Fi network, none was able to establish a connection with the Core Server. Due to the lack of time, there was no chance to properly investigate the root cause of the issue. However, it is possible

that the router was configured to block outbound traffic to the port where the Core Server is hosted.

This issue was solved through the use of a mobile hotspot. Another possible solution, would be to setup a SEGSE, avoiding the need for an internet connection.

5.3. Results

The MT verified that both gateways were able to connect to the Core Server without any kind of network configuration. As soon as both components had internet access, provided by INESC-MN's Wi-Fi, they established a connection with the Core Server, enabling any client to use them to communicate with the spacecraft.

Due to the extensive use of ground segment functionality, this test also ensured that none of the previously implemented features were broken, and that the ground segment is able to be controlled remotely.

Despite the challenges described, the Vibration Test (VT) campaign validated once again the working condition of the ground segment.

6. Conclusions

Throughout this work, changes to the ISTSAT-1's architecture were proposed and implemented, aiming to improve the goodput of the satellite, as well as the usability of the UI leveraged to operate the ground segment and, by extension, the spacecraft.

To this end, the connection mechanisms and interfaces between gateways and the centralised server where standardised with the purpose of reducing the network load, increasing maintainability, and providing a clear commanding interface. The design decisions made to accomplish these goals, motivated the extension of an in-house RPC library, to allow servers to issue remote calls on clients, which greatly simplified the implementation process. These changes were validated during two test campaigns.

The work on the UI for the ground segment resulted in the integration of the cloud based, command and control solution Major Tom. While this service proved to be an adequate solution to interface with the functionality provided by the ground segment, due to contractual problems, the partnership with this company did not proceed; motivating the development of a new GUI. This UI provides a simpler interface, more tightly coupled with the needs of this specific ground segment.

On the automation front, an approach to the scheduling of operations was discussed and implemented. Enabling users to calendar the execution of operations directly through the centralised server.

6.1. Future Work

In order to improve upon this work, the GUI developed should be subject to user trials, evaluating its performance when compared to the CLI, both for new and proficient users.

Another way of expanding on this work would be to allow interested people to contribute to the telemetry data collection process. This can either be achieved by integrating the ground segment with a GSN, or by openly distributing the software for the ground stations. If the latter option is chosen, any person could use their ground station to collect telemetry, and automatically send it to the Core Server. In order to incentivise contributions, the satellite data could be displayed in a public manner, with a leaderboard listing the most active contributors.

References

[1] *Instrument procedures handbook*. Aviation Supplies and Academics, Inc., 2017.

[2] W. A. Beech, D. E. Nielsen, and J. Taylor. AX.25 Link Access Protocol for Amateur Packet Radio. page 143.

[3] G. D. Considine and P. H. Kulik. *Van Nostrand's scientific encyclopedia*. Wiley-Interscience, Hoboken, N. J, 10th ed edition, 2008.

[4] B. Elbert. *The Satellite Communication Ground Segment and Earth Station Handbook, Second Edition*. Artech House space technology and applications library. Artech House, 2014.

[5] A. Melnikov and I. Fette. The WebSocket Protocol. RFC 6455, Dec. 2011.

[6] M. Pessans-Goyheneix, J. Bønding, M. B. Tychsen, and K. F. Jensen. AAUSAT3. Publisher: Citeseer.

[7] J. Vanreusel. Fly Your Satellite! The ESA Academy CubeSats programme. In *Proceedings of the ITU Symposium & Workshop on Small Satellite Regulation and Communication Systems, Santiago de Chile, Chile*, pages 7–9, 2016.

[8] D. Velten, R. Hinden, and J. Sax. Reliable data protocol. Technical report, RFC-908, BBN Communications Corporation, 1984.

[9] J. Wertz and W. Larson. *Space Mission Analysis and Design*. Space Technology Library. Springer Netherlands, 1999.