Development of the ISTnanosat-1 Ground Segment

Alexandre Silva
alexandre.s.silva@tecnico.ulisboa.pt

Instituto Superior Técnico, Lisboa, Portugal

May 2017

Abstract

In this report we propose a solution for the ground segment of the ISTNanosat-1.

The ISTNanosat-1 is a CubeSat, a type of artificial satellite, currently under development by students and faculty from IST along with amateur radio experts from AMRAD and AMSAT-CT. As with any spacecraft, it will need to be monitored, controlled and results from scientific experiments have to be retrieved. The ground segment is the necessary infrastructure on Earth that will permit these tasks.

The ISTNanosat-1 Ground Segment presents a system capable of telemetry retrieval and issuing instructions to the satellite. All of the relevant information is then shown to the users using a web application. Furthermore, properly authorized users have the capability to issue the aforementioned instructions. The developed ground segment is composed by multiple ground stations controlled by Core server. In this centralized design, the Core communicates with the satellite trough the ground stations, which serve as gateways, using the designed and implemented ISTNanosat Command Protocol. In the Core, several services permit for the for data and error retrieval, command and diagnostics execution and security of the ground-space segment link.

Keywords: satellite, CubeSat, ISTNanosat, ground segment, ground station.

1. Introduction

The ISTNanosat-1 is a CubeSat, which is being developed by students and faculty from IST/ULisbon of various different engineering programmes, along with amateur radio experts from AMRAD (AMSAT-CT), CEIIA and EDISOFT. The nano-satellite itself is composed by different subsystems that fit together inside the cubic case. These modules can be divided in two groups, the so called flight subsystems and the payload. The former are subsystems pertaining to the required control and communication functions for the correct satellite operation, while the latter perform the satellite's four missions (One primary and three secondary ones). Additionally, this first iteration of the ISTNanosat project will serve as a stepping stone to gain valuable experience for future more ambitious missions and CubeSats.

The primary mission involves the Automatic Dependent Surveillance – Broadcast (ADS–B), which is a surveillance system for tracking aircraft using periodically broadcast beacons containing their positions. While today this system is optional, it will become mandatory in 2020. Currently, only ground stations track aircraft, and therefore, remote areas such oceanic routes remain unmonitored. The tracking of aircraft from space became particularly interesting, as airline industry safety in such regions became the topic of debate after the incident regarding flight 370 from Malaysia Airlines.

The ISTNanosat-1 will thus build upon previous spacecraft missions regarding ADS-B technology and test a wide Field-of-View (FOV), small form factor (patch) ADS-B antenna, installed either on the nadir pointing face of the satellite or on a deployable mechanism. Once operational, the ISTNanosat-1 will collect aircraft tracking data, in particular from remotes areas, and transfer it to the ground whenever possible. This data will then be analyzed in order to characterize the "Cone of Silence" when a aircraft is at the nadir of satellite and several performance metrics including the probabilities of target aquisition, detection and identification.

An artificial satellite system has three major operational components, the Space Segment, the Ground Segment and the Users. Together, the ground and space segments are what permits a satellite to be communicable from Earth, which then permits its control, data collection and overall usage by the Users to accomplish its missions, see figure 1.

The Space Segment comprises the satellite constellation or, like in the ISTNAnosat-1's case, a singular satellite and the uplink and downlink satellite links. The satellite itself is composed of sev-

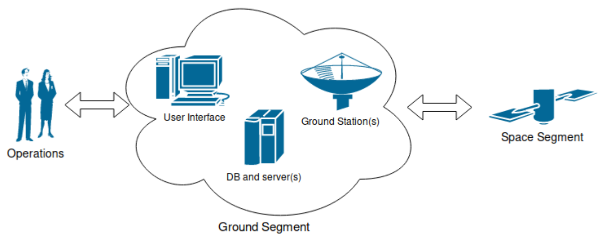

Figure 1: Ground and Space segments.

eral subsystems. The Command & Data Handling is the module in the satellite responsible for monitoring and coordinating all other subsystems. The Electrical Power Subsystem handles the batteries, solar panels and its function is to provide power to the satellite and its subsystems. The Attitude Determination and Control System is responsible for determining the satellite's position and to make orientation corrections if needed. Finally the Communications module assures the satellite's communications.

The ground segment is the necessary infrastructure on Earth that supports the communications, monitorization and control of the satellite. Ground-Space communications are support by what is called ground station, a computer connected to a radio capable of communicating with the satellite using a predefined set of protocols. Using one or more of these ground stations, the Ground segment can then receive telemetry, data produced by the satellite's subsystems, and issue of commands as to instruct the satellite to perform various tasks. Once, the Ground Segment can monitor and command the satellite it must provide a interface such that the users can view it and issue commands. These users invariably include the Mission Control Crew (MCC), which will take care of the housekeeping related tasks, and those interested in the missions. Note however, that this two groups are not necessarily composed with different individuals.

As with other satellites, the ISTnanosat-1 will need constant monitoring and adjustments to keep it in good working conditions. Not only to command the satellite flight functions but also to retrieve important data concerning the scientific experiments and applications performed on-board. The focus of this dissertation will then be the design and development of the ISTNAnosat-1 Ground Segment. It is this system that will communicate with the satellite and therefore perform its monitoring, control and data retrieval.

After communication and diagnostics are assured, an interface must be defined and implemented to permit project members to control and monitor the satellite, as well as, any other user to view and retrieve data from the scientific subsystems. Since the ISTNAnosat-1 is being produced in an academic context, all data is to be publicly available trough this interface such that it can be viewed and analyzed.

The first and main objective, is to create and provide a system that allows to communicate with the satellite. This system will then be used by the MCC to retrieve data and issue commands. It will also serve as a base for performing the housekeeping of the satellite and to retrieve scientific data.

In order for the MCC to perform its duties, they must be able to issue commands that change the state of the satellite. These commands, which are transmitted over radio in insecure links, can in the hands of a ignorant or malicious user render the satellite completely inoperable. Therefore, this system must be designed in such a way that **only** the actual MCC's commands are executed.

Once communications are assured, a user interface must be produced, which will be used to interact with the satellite. This interface must permit for two distinct uses. First, it must permit for the ISTNAnosat-1 project members to view all collect telemetry and to issue commands to the satellite. In other words, allow for the overall management of the satellite and it's flight subsystems. Secondly, due to the academic nature of this project, all of the data should be publicly available to anyone who wishes to see it. This the intended use for anyone who is not part of the MCC but is interested in the data produced as a result of the project's missions. Due to the two different uses for this interface, we must strike a balance between creating an interface that is as functionally rich as possible for the MCC and user friendly everyone else.

2. Architecture

The Ground Segment of the ISTNanosat-1 exists to fulfill three major goals: i) the retrieval of data from the satellite, both in the context of maintenance and the project's missions; ii) the posterior analysis of this data by personnel on the ground; iii) assure the delivery, execution and result retrieval of instructions, i.e. commands and diagnostics, to the satellite. From these goals we distilled the following fundamental requirements.

First and foremost, the Ground Segment must assure the communications with the satellite as to support telemetry retrieval and instruction transmission. The challenge of this issue is the limited bandwidth and communication window any one ground station. Therefore, the end design must be able to support and use multiple distinct ground stations. As an added bonus, several ground stations also serve to improve the redundancy, and thus reliability, of the Ground Segment. However, the usage of several ground stations leads to the possibility of collisions in the ground-space radio link. Which can occur when multiple ground stations are in LOS of the satellite at the same time. Therefore, some sort of mechanism must be employed to eliminate, or reduce to the absolute minimum, these collisions.

Secondly, and as we previously mentioned, each subsystem produces telemetry and has a set of operations it can perform. However, some mechanism must exist in order to transfer the telemetry and instruct the execution of these operations. Therefore, a control protocol must be specified and implemented which will allow for just that. I.e. the retrieval of telemetry (data and error logs) and the issuing of the instructions (diagnostics and commands). Using this protocol and the communications capabilities assured by the above requirements, the Ground Segment will collect and persistently store all telemetry and instruction results.

Figure 3 shows an overview of the Core's architecture. In it, the center block contains the various services which handle the internal logic for data retrieval and instruction execution. The Ground Station Handling block provides a simple API for sending and receiving messages from the Satellite. It Abstracts the internal details of the Core-ground station and ground station-satellite communications. The messages are constructed, serialized and unserialized using the libINCP block. The REST interface uses interacts with to 1) retrieve collected telemetry and executed instructions; And 2) issue instructions. The database is used to persistently stored all telemetry, instruction exections, user data and other Ground Segment related informations.

The data service is responsible for the retrieval of produced data from the satellite's subsystems. This data is organized in variables were each has an indentifier unique per subsystem. Thus each variable is can be uniquely indentified for the entire satellite by the *(subsystem, ID)* tuple. Additionally, each of these variables has a type specified by the subsystem's author such that thse can be used, stored and displayed to the users.

Consider for example, the variable which describes the charge level of the satellite's battery. This variable is characterized by a certain ID of the EPS and its type is a float which corresponds to the charge percentage. Internally the battery sensor may be polled several times a second, however, the subsystem reduces this raw data into a single *(value, TIMESTAMP (TS))* tuple to be transfer retrieved using the ISTNanosat Control Protocol. The TS should match the value in question. For example, if produced values are the result of the simple average of all polled values over a minute, then the TS could correspond to its middle instant.

The rate at which this values are produced are defined by the subsystem, taking into account several factors such it's important, value production rate, the reduction algorithm (if used), available space and the overall link budget (the dedicated throughput for the variable). This rate is however, inconsequential to the Ground Segment whose minimal logical unit is the *(value, TS)* tuple by the radio link. The ground segment only assures that each value transferred is stored and then visualized.

The other kind of telemetry are errors. These are similar to the data but have a few differences. Firstly, these are produced infrequently and typically can have no value at all. Meaning, the Ground Segment only knows that it occur and when.

Each subsystem has a set of commands which can be executed if so instructed by the a authorized user whenever the ISTNanosat-1 is in range. Each command is identified by *(value, opCode)* tuple and has input arguments and output values. These arguments and values, are specified by the subsystem and have well defined types.

Like errors and data variables, diagnostics are uniquely identified by a *(value, TS)* tuple. AS previously mentioned, the goal of diagnostics is to actively test a component and verify its status. Once issued, the resulting test can take a long to be performed and may even finish once the connection window is closed. In such an occasion, the Core retrieves the results on the next window. These result are similar to command results but also include the start and end TS of the diagnostic execution. In addition to on the fly issuing of diagnostics, these can be scheduled to be issue at the next connection window with the satellite.

Commands and diagnostics need to be send securely. But simply put, whenever the command or diagnostic services are to communicate with the satellite a Secure Session must first be established. Only then, can critical messages, those relating to commands and diagnostics, be sent/received through the Secure Session service, as

The ground stations serve as gateways for the Core to communicate with the satellite. Furthermore, and to support ground stations in remote areas with unstable internet connections, these must store all received data while not connected to the Core. Data which is then forwarded to the Core once a connection is reestablished. Additionally, and for similar reason, the Core may send messages to the ground station while not connected to the satellite for it to later relay to the satellite. These are diagnostics that users scheduled to be sent to the satellite when it enters into Line of Sight (LOS).

The command and control of the ground stations

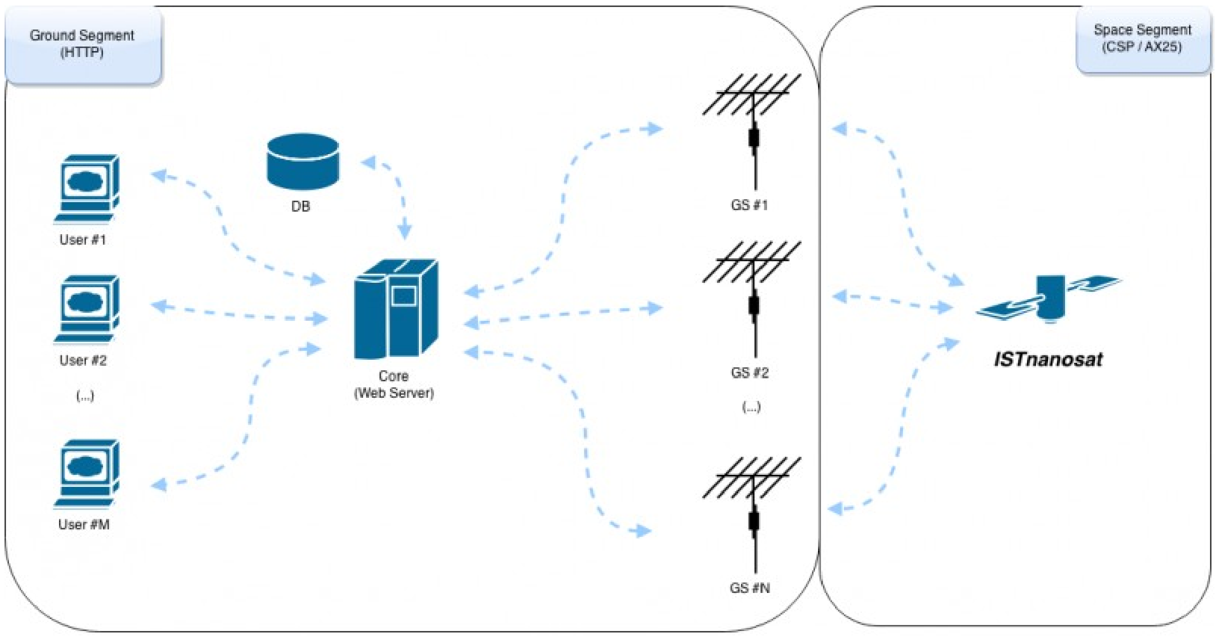

Figure 2: ISTNanosat-1 ground segment topology.

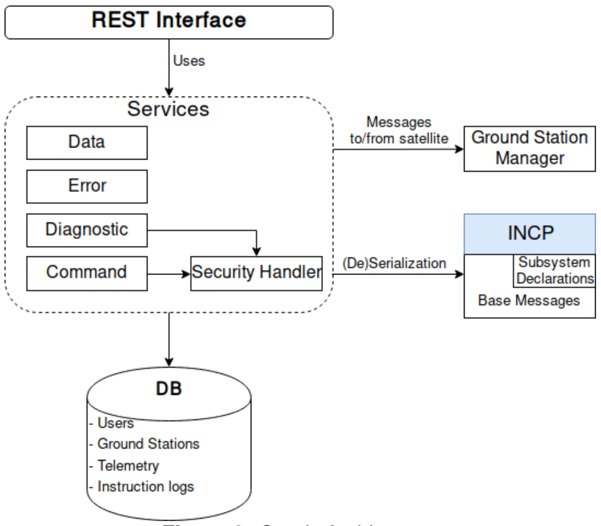

Figure 3: Core's Architecture

has three major components, the Core's ground station handling logic, the ground station logic itself and a, simple, control protocol for communications between the two. In the Core server this logic component can be seen in the *Ground Station Manager* of figure 3. The overall protocols of the Ground Segment can be see in figure 4 including the Designated ground station (DGS).

The aforementioned DGS is a message passing scheme used for communications and control of the ground stations by the core. This is what allows for the core to instruct the ground station to forward messages, order it to connect with the satellite and other control logic.

The Core side ground station management, seen in figure 3, is what allows for the ground stations to connect. Then once, properly authenticated, it's trough it that the services transmit and receive messages with the satellite. In particular, it's what handles the multiplexing of the Core-satellite communications trough the ground stains abstracting this issue from the rest of the Core server.

To avoid collisions in the ground-space link the chosen approach is for the Core's to select a DGS. The selection process is relatively simple and is done by first filtering all ground stations in range of satellite, known by received beacons, and then selecting the one with the highest priority. If the resulting ground station has a different priority than the current DGS, if one exists, a handover process is initiated between the two. This selection is triggered whenever a event occurs which can affect the selection. Specifically, when a ground station connects or disconnects from the Core or if one comes into/out-of the satellite's LOS. The handover process is simply informing the current DGS to terminate its connection, and instructing the new to connect to the satellite.

Selection of the designated DGS is based on its profile. Each of the above profiles has a priority matching its capabilities were the main ground station has the highest priority.

3. Implementation

Since the ISTNanosat-1 is an ongoing project involving multiple people, each working on different components and subsystems, we still do not have a physical satellite with which to test the Ground Segment. Even if we did, to perform all testing during development with it, would pose a serious lo-

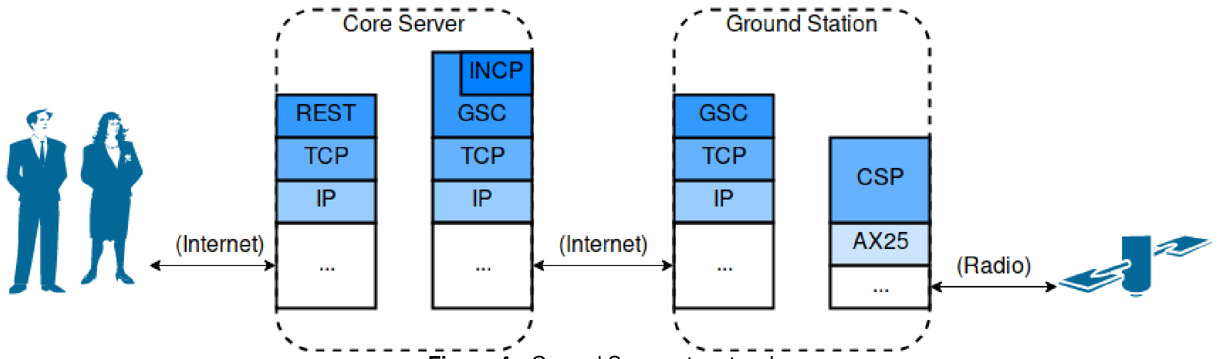

Figure 4: Ground Segment protocols.

gistical challenge due to the many people involved. Therefore, we implemented a emulated satellite in order to perform the development and testing of the Ground Segment.

Since without a space segment the ground segment serves no purpose, the emulator's value is obvious. However, to properly test the ground segment, the protocol stack must be the one shown in figure 4 with the INCP, CSP and AX25 protocols. Furthermore, the external behavior of the satellite must be one that, from the Ground Segment's perspective, is the same as the final satellite. Therefore, the emulator fully responses to the various services which the INCP enables, such as data and error log retrieval; diagnostic and command execution.

We opted for using C for implementing the INCP after taking into consideration the overall ISTNanosat-1 project's context. Firstly, a C implementation allows for a single implementation for both the ground segment and for the satellite's subsystems, thus reducing the code surface area and consequently the likelihood of bugs. Secondly, implementation effort is reduced for the entire team at the cost of we having to 1) implement the protocol in a lower level language and 2) having to handle the binding between the C implementation and the Core server written in Python. Thirdly, and if the reader recalls from section **??**, the INCP requires some information about the *IDs* and data types of the telemetry and instructions for its normal functioning. By having a single lib, we can also "codify" the meta-data about the variables, errors, commands input/output arguments and diagnostics results. Therefore, a single source of truth exists across both the ground and space segments.

A consequence of having a shared lib which is that it must run both in the satellite's subsystems, i.e. an embedded system running FreeRTOS, and in the Core, i.e. Linux. Therefore, we must also respect the computation constraints of the lowest denominator, in this case the subsystems and the two very distinct operating systems. Furthermore, NASA's guidelines for safety critical software must be taken into consideration.

For the reasons detailed above, the libICNP is written in C. However, the Core server which depends upon this library is written in Python. Therefore, some sort of mechanism for binding the two language was needed. In order to test the possible alternatives for doing this binding a small C code snippet was written with some base constructs (e.g as functions, macros and structures).

After comparing two possible alternatives, SWIG and Python ctypes, we concluded that SWIG has a superior type checking mechanism which does not require manual intervention, unlike in ctypes. Furthermore, it creates wrapper classes automatically and allows access to defined constants of the header files. Therefore, SWIG appeared to be the solution which requires the least overall effort with the best set of functionalities. Unfortunately, it the up-front cost of configuring the wrapper generation. However, this was done once and we expect not having to touch it again.

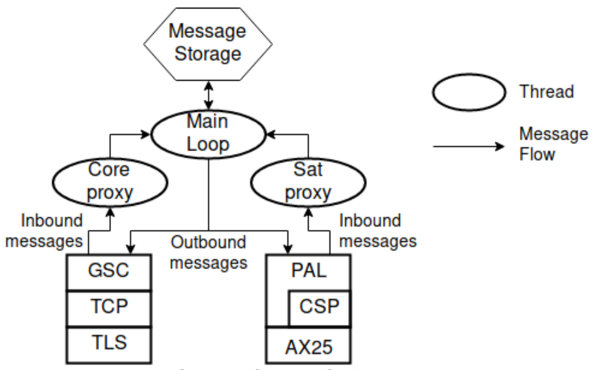

Figure 5: Ground Station Software Architecture

The ground station is essentially an event driven application, were the events are received messages. Since the messages can be received from two different hosts, the Core server and the Satellite, we opted to divide the program into three threads namely, the Main Loop, the Core and Sat proxies. The main thread (i.e. Main Loop) is were the business logic is handled.

The satellite proxy, like the core proxy, unserializes and places any received message into the aforementioned message queue. However, it does not establish connections with the satellite on its own. Instead, the *Main Loop* thread establishes the connection, and only then, does the proxy block itself waiting for messages. The beacons however, which are constantly sent by the satellite, are always received and place in the message queue. The *Main Loop* then retrieves these messages, and either stores them or sends them to the Core.

The *Main Loop* thread, is the one that actually performs the operations expressed in figure 6. On the *wait for message*, it's blocked waiting for messages to be inserted into the thread safe queue, which are placed there by the two proxies.

The satellite-ground station protocol stack has two protocols. Specifically, the AX25 protocol, provided by the Linux's libAX25, and the CSP, provided by a libCSP, modified in order to support AX25.

However, these two protocols are not interacted with directly. Instead, the Protocol Abstraction Layer (PAL) serves as an adapter between the protocols' libraries and the rest of the application. This adapter serves three major goals:

Firstly, since libCSP is designed to be used in a both Linux and FreeRTOS it does not follow the Posix Socket design and API. For example, it uses an internal buffer for storing packets and thus avoids memory allocations during run-time, as is best practice for embedded systems. Furthermore, it uses both its own API for creating packets and *csp_send()/csp_recv()* functions, rather than the Linux's *send()/recv()* system calls. The PAL then provides a single API which hides both protocol's APIs.

Secondly, and as detailed in section 3, the used stack is different depending on whether the satellite is in safe or normal mode. Thus, the *pal_send()/pal_recv()* calls to the appropriate protocol by maintaining a state machine for just this purpose. Furthermore, whenever a state transition occurs, the PAL handles the necessary configurations of each protocol.

Finally, the libAX25 implementation as a few limitations when dealing with connections, detailed in section 3. Thus, the implemented solution to this issues is also handled by the PAL.

The Core is composed by several services. Since multiple users may simultaneously invoke functionalities of the service, each must handles its own concurrency issues. This way, the exposed API, used by the RESTfull interface, is oblivious to such issues.

In order avoid manually handling the low level details of interaction with the database we opted to use SQLAlchemy. In part due to its object-relational mapper (ORM), which allows for the data mapper pattern, where classes can be mapped to the database. Which is how SQLAlchemy handles the laborious and tedious work of creating queries for insertion and retrieval and tables and rows. Furthermore, it does this without hiding the underlying SQL related processes. Instead, it provides abstractions transparently built on top of these processes. It's this transparency what we leverage in order to the create the data, errors, commands and diagnostics tables with SQL types equivalent to the ones found by introspection of libINCP. Furthermore, it permits to chose from several DB backends (sqlite, MySQL, PostgreSQL, etc). Thus, we can have one backend for the final version, and one for unit testing were we want speed and simplicity (i.e. no server configuration required). In the latter case, an in memory version of sqlite is used.

The external interface which the core exposes to the users as four main components and functions, as shown in figure 7. Firstly, it provides the static content needed for building the web application interface. Secondly, it provides a JSON file (i.e. the mentioned INCP.json) such that the interface knows which data and functions the Core supports regarding its various services. Thirdly, it provides a REStfull interface providing access the to the Core's services. And fourthly, authenticates and maintains user sessions.

4. Validation

We designed and performed several sets of test. Each had an increasing level of granularity regarding what is tested. These tests were designed by following the guidelines in NASA's Software Safety Guidebook. Specifically, we have five sets:

Firstly, we detail how we used static code analyses tools to diagnose issues with the developed code. Whole classes of errors are eliminated without having to dedicate any effort other than an initial configuration and setup phase.

Secondly, we briefly discuss how, which and what unit test were implemented. These units tend to either be classes, or types and functions logically connected. Note that unit tests can be used both for testing and development purposes. However, both ends depend on the ability to test one and only one "unit" in isolation and with as much thoroughness as desired.

Thirdly, we detail the employed integration tests used to validate the end-to-end functionalities of the entire Ground Segment. This tests, builds upon the unit tests to both validate areas which the latter do not cover, and these work as intended when combined. Note that another class of tests are the so called functional tests. Although, there are some differences between the two, we combined the two and address them only as integration tests.

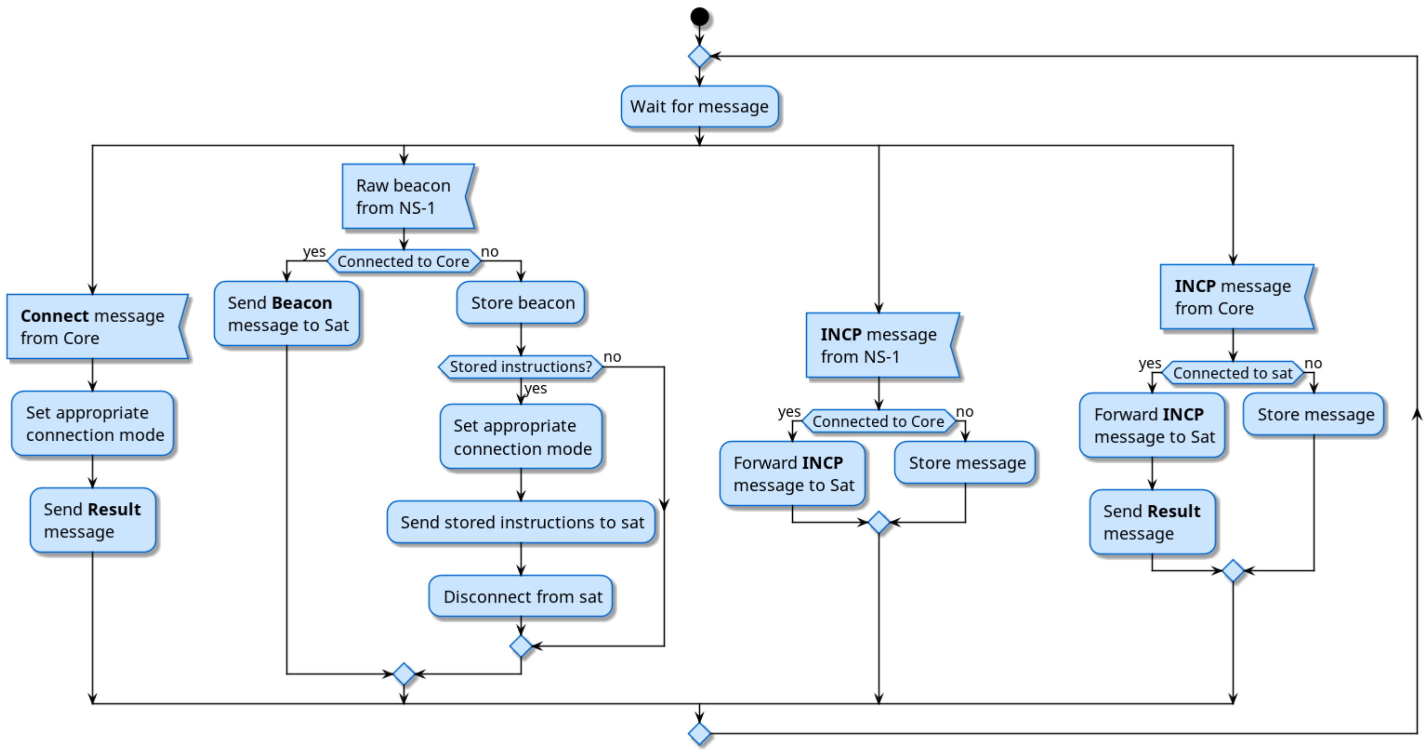

Figure 6: Ground station execution flowchart.

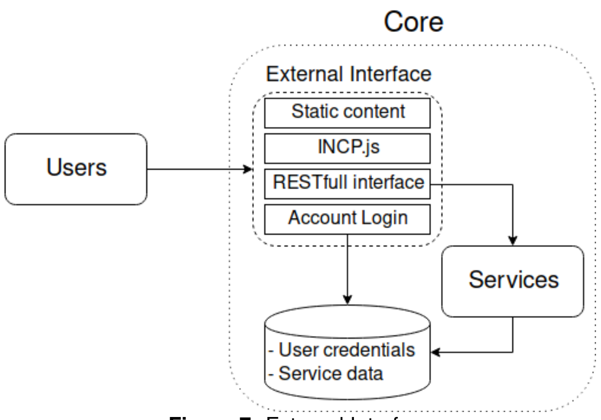

Figure 7: External Interface

Fourthly, we performed a test were the operation conditions were beyond the expected range. The goal is essentially to stress the system in such a way that if there is a flaw, it's revealed by the test.

Finally, we go over the checklist provided in NASA's guidebook. This checklist basically has a series of checks which cover the most common bugs and errors found in critical software.

5. Conclusions

The ISTNanosat-1 Ground Segment presents a system cabable of telemetry retrieval and issuing instructions to the satellite. All of the relevant information is then shown to the users using a web application. Furthermore, properly authorized users have the capability to issue the aforementioned instructions. The developed ground segment is composed by multiple ground stations controlled by Core server. In this centralized design, the Core communicates with the satellite trought the ground stations, which serve as gateways, using the designed and implemented INCP. In the Core, several services permit for the for data and error retrieval, command and diagnostics execution and security of the ground-space segment link.

The INCP was designed to permit the retrieval of only the desired telemetry, produced while no ground stations were in Line of Sight (LOS) of the satellite. This is performed in such a way that minimizes, as much as possible, the re-transmission of repeated information, by specifying the desired time interval for the desired information. The implementation of the protocol was done in such a way as to permit it's use in the satellite's subsystems. Consequently, the context and limitations of developing software for embedded systems of a satellite were taken into consideration. Since the protocol is to be used by other developers, we took great care in order to maintain a simple the API and make the implementation simple to use. Additionally, by "codifying" the commands, data, errors and diagnostics in this implementation, we created a single source of truth for the entire ISTNanosat-1 project.

Unfortunately, we could not test the libINCP on one of the subsystems' boards. Something, which must absolutely be performed. Furthermore, due to the various introspection information some work may be performed in order to, by compile time flags, exclude unnecessary portions of the implementation. This is especially critical for those subsystems which minimal working memory such as the EPS.

In the Core server, we leveraged the high level nature of Python and it's mature ecosystem in order to produce a solution which is easy to maintain and extend. In particular, the usage of the libINCP introspection information to drive the telemetry and instructions of the services. All of the services functionalities are then exposed via REST interface in a simple way. Although the user interface still needs some polish.

Regarding the ground stations and how these are controlled. The presented solution is both simple and effective. In particular, the Protocol Abstraction Layer (PAL) hides the complexity of the AX25-CO and libCSP handling. Furthermore, we also present the implementation of the AX25-CO interface for the libCSP.

The security scheme was designed to minimize the overhead both computationally for satellite's subsystems and in transmission throughput. For the user and ground stations existing solutions were employed taking into consideration common practices.

Currently, the Core uses the periodically transmitted beacons as indication that the satellite is in Line of Sight of a ground station. Then, it opportunistically selects one DGS which serves as the gateway between it and the satellite. Although in the tested scenarios this scheme works without any major issues, we suspect this will not always be the case when using the radios for ground-space communications. The most obvious case, is the detection of lost communication between the ground station and the satellite when another ground station is in range. In the current approach there will be a time period were the DGS will be out of LOS but before the AX25 connections times-out and a new DGS is selected. If a preemptive handover was performed, this wasted communication time may not be wasted. One possible solution may be using the known orbit and ground stations positions. With this information, one may try to predict which is the best DGS at a certain time and when to perform a handover. Although, this may be just possibility that never materializes, this is an issue that needs to be investigated.

Regarding the performed tests and overall validation, it consists manly in functional requirements validation. The load test however, showed without question that the ISTNanosat-1 is a network bound system. Event trough several used libraries and Python itself are known to have significant overheads, when compared to naively compiled solutions. Additionally, the performed validation is mainly a functional tests. Therefore, we took the opportunity to automate most of the tests, such that future work is as painless as possible.