


Experiences and lessons learned developing a next-generation ground segment prototype

M.Sc. Eng. Pablo Soligo

*Departamento de Ingenieria e Investigaciones
Técnicas - Grupo de Investigacion y Desarrollo
de Software Aeroespacial.
Universidad Nacional de La Mantaza
Buenos Aires, Argentina
psoligo@unlam.edu.ar*

PhD. Jorge Salvador Ierache

*Departamento de Ingenieria e Investigaciones
Técnicas - Grupo de Investigacion y Desarrollo
de Software Aeroespacial.
Universidad Nacional de La Mantaza
Buenos Aires, Argentina
 <https://orcid.org/0000-0002-1772-9186>*

Index Terms— Artificial satellites, Small Satellites, Ground support, Ground segment, Software design, Space technology, Telecommands, Telemetry, COTS.

Abstract - In this paper we discuss how, using general purpose software tools and advanced programming techniques, it is possible to create a multi-mission ground segment software application that can be applied in almost (virtually in) any mission. We developed a functional prototype and worked with real satellite telemetry and commands. Different alternatives were explored and several analyses have shown the advantages of this approach in terms of productivity, maintainability and cost. One of the main goals was to use modern general-purpose tools instead of old or spatial industry classical ones, providing a cost-effective and multi-platform solution. We used general purpose databases, a well-known Object Relational Mapper (ORM), popular programming languages, libraries, frameworks and advanced programming techniques. We avoid any tool, protocol or strategy not used in software industry. Telemetry data from several sources is processed, stored and finally showed through Open Source Mission Control Software (OPENMCT) application by National Aeronautics and Space Administration (NASA). External software units can interact with the system using (REST)/JavaScript Object Notation (JSON) and web sockets interfaces. The interfaces are available to receive telemetry from different external systems or to publish telemetry to different external clients. In particular in this work, the

published telemetry interface is used by OPENMCT. An initial prototype was developed to support the flight segment of a cubesat engineering model provided by Innovative Solutions In Space (ISIS) in the context of the academic mission Formador Satelital 2017 (FS2017), as part of Master in software development for space application (MDIAE). Currently we test compatibility (without commands) not just for cubesat but also for medium or large missions. The system evolved from a traditional client-server architecture to a distributed system in order to improve the horizontal scalability and, in its most recent version, data mining and machine learning techniques were incorporated. Using well-known libraries this telemetry processor can predict future telemetry values and compare them with the real ones in order to detect anomalies or unknown patterns.

I. INTRODUCTION

This paper elaborates on the experiences and results of the development a cost-effective satellite ground segment system prototype. The prototype design and implementation is based on the experiences carried out during MDIAE with ground segment systems currently being produced and their relationship to general purpose software systems widely used in the software industry. During the design phase, the use of widely accepted and proven robustness tools and solutions was prioritized as

opposed to common proposals of the space industry. Part of the goal of this and previous papers is to collate these implementations, with prevalence in three domains:

A. Decoding, command and control languages

The use of general-purpose languages and interpreters is proposed to decode telemetry and develop command scripts.

B. Data Persistence

An architecture where data persistence¹ is performed on relational database engines solely accessed through an ORM is proposed. Both the data in all processing levels and the definitions are stored in a single central repository.

C. Interfaces

Interfaces with internal and external systems for module communication as well as data publication and ingestion are performed through extended and accepted protocols.

Ad-hoc or low-penetration standards, formats and protocols in the massive use industry are marginalized from the solution and COTS solutions, libraries, components and frameworks are prioritized. Even though part of the goals were related to a proposal different from the collected systems, the recommendations published in AIAA [1] have been taken into consideration, for being in accordance with good practices of software engineering.

In chapter 2, the background encouraging the creation of the GIDSA research group and the established technological premises are explained. Chapter 3 describes the approach adopted to control the satellite, decode its telemetry and implementation details. Chapter 4 presents the storage methods used and the restrictions imposed in order to achieve a better compatibility in different scenarios. Chapter 5 describes some characteristics of the implemented interfaces and the currently integrated systems. Chapter 6 describes the latest system adaptations; whose goal is to achieve a predictive behavior in terms of state of health. Finally, chapters 7 and 8 list results and conclusions.

II. BACKGROUND

During the experiences undergone in the first

cohort of MDIAE (2015-2017), the first functional ground segment prototype was developed, now called UNLaM Ground Segment Prototype (UGS) for mission FS2017 [2]. A 2U cubesat engineering model from the ISIS company (<https://www.isispace.nl/>) was used as flight segment.

The first prototype developed was based on a client/server architecture which resulted in the current distributed architecture [3]. In order to develop this first prototype, it was necessary to reverse engineer the model since it did not have manufacturer support.

With these premises and prioritizing commonly used solutions in general purpose systems, we have opted for:

- Using general purpose interpreters for decoding telemetry and command scripts instead of developing our own interpreters.
- Using relational (Relational Database Management System (RDBMS)) and non-relational databases for data and metadata storage.
- Using data access layers (ORM) enabling portability between RDBMS motors and other storage technologies.
- Using maximum penetration tools for task distribution and execution.
- Using communication protocols at a level of applications commonly accepted within the general-purpose IT field.

III. DECODING, COMMAND AND CONTROL LANGUAGES

In the space field, when dealing with controlling a satellite or decoding a telemetry frame, a common practice is the use of proprietary languages for specific purposes. Generally, command sequences are applied to control structures in these languages, as happens with telemetry interpretation in multimission systems. Space companies and agencies around the world have developed their own languages and interpreters in order to achieve this goal [4]:

STOL: Satellite Test and Operation Language. It has been developed by Nasa and widely used in

¹ It refers to the data attribute for them to survive somehow.

several missions.

PLUTO: it has been used in some ESA missions (Satellite Control and Operation System 2000).

Others: developed or used by different companies SOL (GMV), CCL (Harris), PIL (Astrium), SCL (ICS).

These languages are typically proprietary and incompatible with one another [5], hindering migrations between systems and the possibility of sharing procedures between different missions.

FS2017 [2] was the first flight segment used as experimental test for developing the systems presented in this paper, subsequently broadening to telemetry in Lituanicasat2, Bug-SAT Tita and SAC-D missions. We opted for a general purpose language instead of creating a specific language or using the existing ones in CONAE, academic partner in MDIAE.

This approach has many advantages. The following table compares pros and cons of each implementation type.

	Specific purpose	General purpose
Advantages	They can be more user-friendly for non-programmer users. They can offer specific adaptations to satellite operation problems.	Portable, more powerful than specific purpose versions. Great number of users.
Disadvantages	It is a proprietary technology. Portability issues. Own development. Poor tools and lack of documentation	Too many options, Too powerful, Could be less readable.

In the case of FS2017 ground segment, the idea of a proper mission language was rejected due to lack of time and resources needed for developing and validating a specific purpose interpreter. This condition is particularly true in small satellite missions, which lack this capability due to the cost of incorporation, resulting in projects without this capability. Python was the language used due to its popularity [6], ease of learning and reflexive characteristics, even though the concepts presented here are applicable to other interpreters with similar characteristics.

By using a general-purpose interpreter, we can also build scripts with better debug capabilities, community available documents and specially avoid the need to take or update the interpreter in several platforms.

A. Telemetry processor

The system allows setting up telemetry variables showing type of frame, position, number of bytes, and form of interpretation, and it is possible to assign a calibration function that changes raw information into engineering variables. This calibration function is developed by python independently from the system by the specialist. When including new telemetry, the system distributes the processing and the unit in charge of processing the telemetry variable will look for the applicable calibration method in the configuration (Figure 1 - Calibration Method Selection). If not previously uploaded, it will be uploaded in runtime and called up to perform the processing.

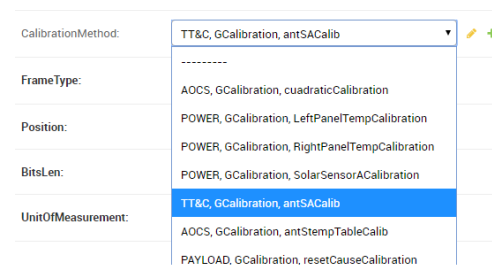


Figure 1 - Calibration Method Selection

A distributed process regularly monitors the calibration methods that have been included to perform the load and checks the methods that have modified their shape to perform a re-load of the updated version. Figure 1 - Calibration Method Selection shows the selection of a calibration method for a specific telemetry variable. Methods are uploaded and updated on runtime.

The software reflection technique used for uploading and running calibration functions is generally costly in processing time. Two types of optimization are used to improve performance:

- If no changes occur, functions are uploaded once. Any change to the source files means re-uploading the function.
- The function only runs if the new raw datum is different to the previous one or if the calibration function has been recently uploaded, which implies a possible change.

By applying these simple optimizations, the tests performed proved that it is possible to

calibrate/decode around 5000 engineering variables between 2.5 and 3.5 seconds -a capability enough to provide a satellite with characteristics similar to the ones of SAC-D with a desktop computer (Intel® Core™ i5-5200U Dual Core, 2.20 GHz). Figure 2 - Decode Time show the decoding time for 16 samples of 5000 telemetry variables, each of them with a 10% exchange rate mostly using parameterized linear conversions. It can be observed that the first sample takes slightly over 25 seconds. The first decoding requires uploading and running all calibration functions. As from the second sample, the optimizations previously described are executed.

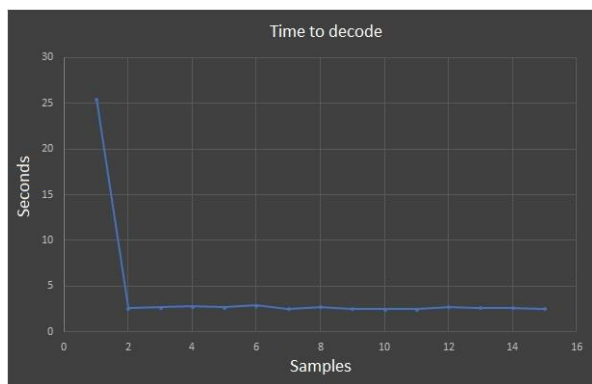


Figure 2 - Decode Time

B. Command processor

Similarly to how telemetry is processed, command scripts are uploaded and called up on runtime. The command processor has only been tested with the FS2017 model. The system allows to set up specific commands for each satellite. The execution of a certain command type can trigger a call to an adaptor, or the execution of a code snippet related to manufacturer's specifications.

Particularly in the case of FS2017, the execution of a command type produces the output of the code defined by the manufacturer in a TCP/IP port working as communication interface with radio software.

```
satellite = Satellite.objects.get(code=="FS2017")
```

```
#Get a
tt = TlmyVarType.objects.get(code="obcT1")
#Get the last five temperatures
tmps = tt.vars
    _order_by('-created')[:5]
    .values_list('calFValue', flat=True)
#Average, using numpy, an external library
med = np.average(tmps)
#Check a condition
if med>TT.max:
    #Condition is true, I'm going to send a command
```

```
ct = satellite.getCommandType()
    .get(code="telemetryOBC")
cmd = satellite.newCommand(ct)
    satellite.sendCommand(cmd)
```

Source 1 – Example of a conditional command script shows a conditional script, where a “telemetryOBC” command is sent if it is detected that the obcT1 telemetry is over the maximum limit. It is also observed that the limit is controlled with a mean value estimated with an external library (numpy), showing the use of external libraries in a command script with an allegedly simple example.

```
satellite = Satellite.objects.get(code=="FS2017")
#Get a
tt = TlmyVarType.objects.get(code="obcT1")
#Get the last five temperatures
tmps = tt.vars
    _order_by('-created')[:5]
    .values_list('calFValue', flat=True)
#Average, using numpy, an external library
med = np.average(tmps)
#Check a condition
if med>TT.max:
    #Condition is true, I'm going to send a command
    ct = satellite.getCommandType()
        .get(code="telemetryOBC")
    cmd = satellite.newCommand(ct)
        satellite.sendCommand(cmd)
```

Source 1 – Example of a conditional command script

IV. DATA PERSISTENCE

In the general-purpose software industry, the use of relational or non-relational DBMS has spread from large banking and accounting systems to small embedded devices. Even though it is not an absolute rule and there exist papers stating experiences on the implementation of formal storage, search and recovery strategies ([7] and [8]) in the space industry, it is not difficult to find complex and advanced systems avoiding the use of these tools [9]. During MDIAE, all the ground segment systems studied avoided storage on a DBMS. Instead, data or metadata were available in binary or flat files. The absence of DBMS hinders effective and standardized data storage and recovery, eliminates a concept of mandatory security such as referential integrity and overrides the capability of several systems working on data in a safe and concurrent way.

In the implemented prototype, both data definitions (alarms, telemetry variables, available commands, etc.) and data itself are run by a relational database motor at any processing level. In satellite telemetry, there are entities for raw data storage independently from the satellite they belong to, or the

source of ingestion. As this data is processed, they are moved to tables where the engineering variables are already processed and available for any module that requires them, including the command processor.

For future versions, implantation of tables that enable efficient access to historic data, KPIs and denormalizations to allow for long term storage is foreseen. All data is exclusively accessed with ORM. The use of this intermediate layer offers productivity improvements as well as independence from the database motor provider.

V. INTERFACES

In terms of internal and external interfaces, there are plenty of ad-hoc solutions or implementations of low-penetration technologies in the space field. CORBA is a remarkable example because it has not been massively adopted and has been quickly changed for HTTP-based implementations [10], even though it is a standard presented by several players in the software industry. The interfaces of the proposed system are fully based on this latest technology. All telemetry is ingested with a REST service which can be used by both third-party modules and internal modules that adapt formats, technologies or characteristics from former implementations. Task distribution, synchronization and general communication are also done with http-based brokers.

VI. STATE OF HEALTH

Particularly, in order to check the state of health, a limit control system, which is one of the most simple and generally used techniques for checking state of health, was initially used ([11] and [12]). During MDIAE, experiences with several systems from the space field that use this technique were carried out; and as from the first version, the prototype checks the satellite's state of health with a limit control (1), for direct and derivative variables (these derivative variables are artificially created from a combination of the others). Even though it is simple to implement, this technique is context-insensitive and the history value [13] is completely unknown. As it was explained in Chapter 3, the use of a general-purpose language for telemetry decoding-calibration allows to use the available libraries to enhance scripts. In the

case of UGS, this capability is used with machine learning scikit-learn libraries [14] in order to find detours during regular running by using historic behavior and contextual variable data.

In its latest version, the system has included entities into the data model to allow decoding-calibration scripts to modify maximum and minimum limits previously configured according to predicted values. The system analyzes historic information and context variables to create a model allowing to predict future behaviour. As from a prediction, the system modifies one-sigma limits to enhance control and avoid false positives [13].

VII. RESULTS AND LESSON LEARNED

A. Graphical User Interfaces

Even though user interfaces were not the main goal during this first project phase, it was necessary to create a telemetry administration and visualization tool that would allow to visualize and share results. The evolution of the prototype, for research use, to an available system with production output capabilities also requires a user interface. As a current solution, we decided to develop interactive web interfaces that would run in any device (PC, Tablet, Cellphone). The development of desktop applications was avoided due to the cost and difficulty experienced when porting to different platforms, such as Windows, Linux and Android.

For administration purposes, the development of static pages was enough. This fast development was enough for the research team to administer data, even though it is not suitable for a production system. To visualize historic and real-time telemetry, it was necessary to use web development frameworks.

1) First prototype

To visualize historic and real-time telemetry, user interfaces were initially developed to run on the browser interacting with the backend through packages JSON. Experienced developers (even though they did not have experience with the used tool) invested more than 120 hours achieving poor results, which only accounted for a fraction of the needs and had serious adaptability issues with different screen sizes. Figure 3 - First Prototype UI shows this first prototype.

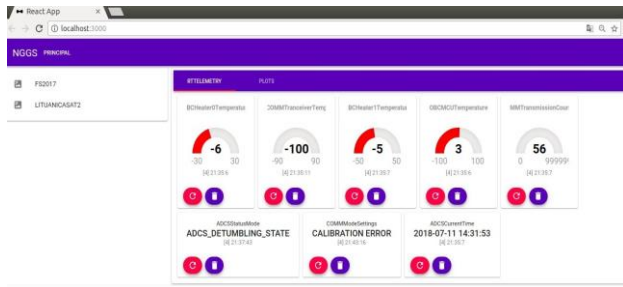


Figure 3 - First Prototype UI

2) NASA Open MCT

As an alternative, an integration with NASA Open MCT (<https://github.com/nasa/openmct>) was evaluated and finally, was partially implemented. The Open MCT proved to be an excellent, modern alternative with a wide range of possibilities but lacks application examples that would help developers, even though it is well documented. Figure 4 - Open MCT Showing telemetry published by UGS shows 2 using UGS data.

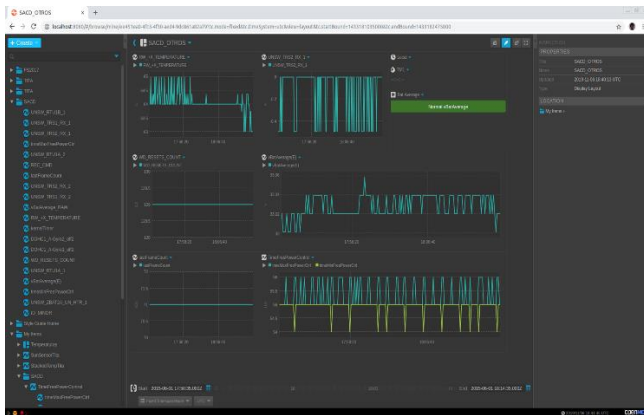


Figure 4 - Open MCT Showing telemetry published by UGS

For the partial integration, which does not cover mandatory aspects necessary for a production output, with NASA Open MCT, no less than 90 man-hours of development were needed.

The initial development (5) was discontinued since it jeopardized other research tasks due to its high development cost. The current state of the integration with NASA Open MCT only includes historic and real-time telemetry and telemetry metadata partially. Even though it is an already developed tool, it needs a developer heavily involved in the technology being implemented. To conclude, modern user interfaces can take more than 40% of total development time

[15].

B. Commands and Telemetry with general purpose languages

At present, general purpose languages have characteristics, capabilities and tools outweighing specific purpose ones, which can hardly be regarded as complex. Several, large-scale projects support their robustness. Well-proven options do not justify a proprietary language development; not having a user database means lacking documentation, support and tools. Small satellite missions do not need to do without capabilities available for larger missions. The use of software reflection allowed to only include capabilities previously used in larger missions at a low cost. The tests performed showed that it is applicable even in general purpose hardware.

C. Telemetry mining

The latest prototype version showed a practical, alternative, and low-cost implementation to a simple limit control. The importance of this implementation regarding early fault detection is yet to be proven and it is one of the future lines of research presented.

D. Other results

The use of mature ORMs has allowed to develop the prototype independently from the database motor. The use of relational databases helps to process distribution and referential integrity maintenance. However, denormalizations allowing to work with data from past years for large satellite missions have not been applied to the prototype. These design adjustments are necessary to get relatively constant response times for large amounts of data. The denormalizations to implement are known and there are excellent style guides [16].

VIII. CONCLUSIONS

The experiences carried out show an experimentation platform that is realistic regarding space systems. They allow researchers and students to test solutions, control limits, compare alternatives and set decision criteria. The possibility of working

with several missions, from small college missions to scientific satellite data, allows to specifically answer to the premise of developing a system transparent to an orbiting satellite. The experiences, progress made, and results show that a promising path has been chosen. The prototype showed that it is low-cost and adaptable, and offers some functionalities even outweighing larger-scale systems. Products such as NASA OPENMCT, directly integrable with the developed interfaces or with papers such as [17] support the results. Future lines of work will continue exploring the proposed alternatives exclusively based on techniques and tools of massive use in the software industry, specially applied to the space industry within the ground segment context. Lines of research will focus on applying data exploitation and automated learning techniques and tools for determining the state of health of the satellite platform.

ACKNOWLEDGMENTS

This research was funded by PROINCE C211. We would like to thank the Engineering and Technological Research Department (DIIT - Departamento de Ingeniería e Investigaciones Tecnológicas) of UNLaM for supporting research. We would also like to thank CONAE for allowing access to the telemetry of the Argentine mission SAC-D.

IX. REFERENCES

- [1] K. Galal y R. P. Hogan, «Satellite Mission Operations Best Practices,» 2001.
- [2] P. Soligo, E. González, E. Sufán, E. Arias, R. Barbieri, P. Estrada, A. Montilla, J. Robin, J. Uranga, M. C. Valenti y others, «Misión CubeSat FS2017: Desarrollo de Software para una Misión Satelital Universitaria,» *WICC 2017*, p. 843.
- [3] P. Soligo y J. S. Ierache, «Segmento Terreno Para Misiones Espaciales de Próxima Generación,» *WICC 2019*.
- [4] G. Garcia, «Use of Python as a Satellite Operations and Testing Automation Language,» de *GSAW2008 Conference, Redondo Beach, California*, 2008.
- [5] G. Chaudhri, J. Cater y B. Kizzort, «A model for a spacecraft operations language,» de *SpaceOps 2006 Conference*, 2006.
- [6] T. I. O. B. E. Software, «TIOBE Programming Community Index, September 2017,» 2017. [En línea]. Available: <https://www.tiobe.com/tiobe-index/>.
- [7] J. Houser y M. Pecchioli, «Database Administration for Spacecraft Operations-The Integral Experience,» *ESA BULLETIN*, pp. 100-107, 2000.
- [8] P. Cruce, B. Roberts, M. Bester y T. Quinn, «A database centered approach to satellite engineering data storage, access, and display,» 2007.
- [9] O. Montenbruck, M. C. Eckstein y J. Gonner, «The geocenter control system for station keeping and collocation of geostationary satellites,» 1993.
- [10] M. Henning, «The rise and fall of CORBA,» *Queue*, vol. 4, pp. 28-34, 2006.
- [11] T. Yairi, M. Nakatsugawa, K. Hori, S. Nakasuka, K. Machida y N. Ishihama, «Adaptive limit checking for spacecraft telemetry data using regression tree learning,» de *2004 IEEE International Conference on Systems, Man and Cybernetics (IEEE Cat. No. 04CH37583)*, 2004.
- [12] T. Yairi, Y. Kawahara, R. Fujimaki, Y. Sato y K. Machida, «Telemetry-mining: a machine learning approach to anomaly detection and fault diagnosis for space systems,» de *Space Mission Challenges for Information Technology, 2006. SMC-IT 2006. Second IEEE International Conference on*, 2006.
- [13] P. Soligo y J. S. Ierache, «Arquitectura de segmento terreno satelital adaptada para el control de límites de telemetría dinámicos,» de *XXV Congreso Argentino de Ciencias de la Computación (La Plata, 2019)*, 2019.
- [14] *scikit-learn Machine Learning in Python*.
- [15] G. Calvary y J. Coutaz, «Introduction to model-based user interfaces,» *Group Note*, vol. 7, p. W3C, 2014.
- [16] T. Morel, G. Garcia, M. Palsson y J. C. Gil, «High Performance Telemetry Archiving and Trending for Satellite Control Centers,» de *SpaceOps 2010 Conference Delivering on the Dream Hosted by NASA Marshall Space Flight Center and Organized by AIAA*, 2010.
- [17] L. A. C. Brighenti, A. B. D. Evans, M. R. D. Moretto y M. C. F. Ferrari, «Advances in Context Aware Spacecraft telemetry Checking,» de *2018 IAC*, 2018.
- [18] T. Yairi, N. Takeishi, T. Oda, Y. Nakajima, N. Nishimura y N. Takata, «A Data-driven Health Monitoring Method for Satellite Housekeeping Data based on Probabilistic Clustering and Dimensionality Reduction,» *IEEE Transactions on Aerospace and Electronic Systems*, 2017.

X. ACRONYMS

AIAA	
Institute of Aeronautics and Astronautics	2
CONAE	
Comisión Nacional de Actividades Espaciales.....	3

COTS	
Commercial Off-TheShelf	1, 2
DBMS	
Database Management System	4
DIIT	
Departamento de Ingeniería e Investigaciones Tecnológicas	7
FS2017	
Formador Satelital 2017	1, 2, 3, 4
HTTP	
Hypertext Transfer Protocol	5
MCT	
Mission Control Software	6
MDIAE	
Maestría en Desarrollos Informáticos de Aplicación Espacial	1, 2, 4,
5	
NASA	
National Aeronautics and Space Administration	1, 6
ORM	
Object Relational Mapper	1, 2, 4
RDBMS	
Relational Database Management System	2
REST	
Representational State Transfer	1
SAC-D	
(Spanish	
Satélite de Aplicaciones Científicas-D, meaning Satellite for	
Scientific Applications-D)	4
STOL	
Satellite Test and Operation Language	2
UGS	
Unlam Prototype Ground Segment	2, 5, 6
UNLaM	
Universidad Nacional de La Matanza	2