SpaceOps Conferences
28 May - 1 June 2018, 2018, Marseille, France
2018 SpaceOps Conference

10.2514/6.2018-2634

A Lightweight and Efficient Control Center Based on Modern technologies

Nicolas Humeau* and Thibault Gateau*

*Institut Supérieur de l'Aéronautique de l'Espace (ISAE-SUPAERO), Toulouse, France
E-mail: **{nicolas.humeau,thibault.gateau}@isae.fr**

Nicolas Humeau and Thibault Gateau are both ground segment engineer on CubeSats projects at ISAE-SUPAERO which is part of the Toulouse University Space Centre (CSUT).

I. Nomenclature

1U, 3U,12U	*Number of « cube » unit of the nanosatellite (one, three or twelve)*
AIT	*Assembly Integration Test*
CNES	*Centre National D'Etudes Spatiales (French Space Agency)*
CNRS	*Centre National de Recherche Scientifique (French National Scientific Research Institute)*
CSUT	*Centre Spatial Universitaire de Toulouse (Toulouse University Space Centre)*
ECSS	*European Cooperation for Space Standardization*
ENAC	*Ecole Nationale de L'aviation Civile (French Engineering School)*
GUI	*Graphical User Interface*
INSA	*Institut National des Science Appliquée (French Engineering School)*
ISAE-SUPAERO	*Institut Supérieur de l'Aéronautique et de l'Espace (French Aerospace Engineering School)*
LAAS	*Laboratoire d'Analyse et d'Architecture des Systèmes (French Resarch Lab)*
ONERA	*Office National d'Etudes et de Recherche Aérospatiales (The French Aerospace Lab)*
OOP	*Object-oriented programming*
POC	*Component-oriented programming*
PUS	*Packet Utilization Standard*
SCC	*Simple Control Center*
SDB	*System Data Base*
UPS	*Université Paul Sabatier (Toulouse University)*
VKI	*Von Karman Institute*

II. Introduction

The Toulouse University Space Centre (**CSUT**) is a scientific interest group bringing together various training organizations (ISAE-SUPAERO, ENAC, INSA, Toulouse University Paul Sabatier) and research establishments (ONERA, LAAS-CNRS, IRAP), with the support of CNES. The CSUT hosts space projects (nanosatellites, CanSat, micro launchers...) carried out by students, from the **upstream study** phases to the **realization and operation** of orbital systems. In particular the following CubeSats projects are at study right now:

1. **Entry-Sat – 3U**
    a. Study of atmospheric reentry
    b. Phase D
    c. ISAE-SUPAERO / VKI
2. **Eye-Sat – 3U**
    a. Study of zodiacal light
    b. Phase D
    c. CNES / ISAE-SUPAERO / ENAC / Paris Sud Cachan
3. **Nymph – 3U**
    a. Study of optical fiber aging
    b. Phase B
    c. ISAE / LAAS / UPS / Thales Alenia Space
4. **Atise – 12U**
    a. Study of boreal aurora
    b. Phase B
    c. CSUT / CSUG (University Space Center of Grenoble)
5. **Spectra – 3U**
    a. Monitoring of radiofrequency spectrum
    b. Phase 0
    c. CSUT / ENSTA-Bretagne

Moreover, the CSUT hosts a generic command/control center for nanosatellites. This control center is in charge of generating orders for the satellite, monitoring its visibility from the ground and computing orbital position. It will be used for the first time to control and monitor the **Eye-Sat**[1] nanosatellite, developed by **CNES** interns, which is currently in the integration phase.

Eye-Sat is a triple (3U) CubeSat planned for launch in early 2019 that will provide a survey of the **zodiacal light** (the light scattered by interplanetary dust cloud) leading to a better understanding of the properties and origins of its particles. It is a modern platform using the **latest technologies** (see chapter 4) from the CNES, with **academics and pedagogical** purpose.

To fulfill the mission, Eye-Sat has high expectation specification in regard with CubeSats standards on-board software: based on the Packet Utilization Standard (PUS) from ECSS the flight software has many requirements like **high speed** downlink for housekeeping telemetry, on-board **scheduling** for telecommands, on-flight software **reloading**, etc. The on-board software is using the latest CNES **embedded command-control** software (the libPUS library). Users and students should also be able to operate it easily: **light** installation manual, **light** user manual, **great** graphic user interface. Thus the ground segment control center need to match this **features** to make this on-board requirements **possible**, **testable**, and **operable**. Literature reveals few existing solutions (Open MCT[2], Cosmos Ball Software[3], QuantumCDM[7]) often proprietary and /or mission specific and to our knowledge, no implementation of control center with such features currently exists at CNES. Solutions for classical satellite operations are not adapted either (proprietary, high complexity, cost to deploy and maintain).

III. The need and the users

Consequently, after studying various solutions and carrying out a number of tradeoffs, we decided to develop a new software. But before embarking on a new development we did a thorough analysis of the need and users.

The initial need is quite different from "big satellite" project since this control center software is not meant to be used just for operation but also for flight software validation and verification, for assembly, integration, test (AIT) phase and for Test Benches validation. This is a key point: on this type of small project, we don't have enough resources to develop specific software for each phase. Using the same software for OBS, AIT, and operation is a response for it.

It means that when we first thought about the architecture of the software we had to take into account different user needs and not just operational and we had to think out-of-the-box in order to anticipate what could be the perfect architecture in order to not redevelop and share a maximum of features. Here is a list of the user that we identified:

1. Students (that can take or assist all the followings roles)
2. Flight software developer
3. Flight software integrator
4. Assembly, test, integration engineer
5. Attitude control law developer
6. Test bench developer
7. Test bench validator
8. Ground software developer
9. Operator

So what we first did before starting any development activity was building an architecture for the back-end and the front-end that would match the needs for all these **profiles**. avoiding re-developpments as features are common between many users. Therefore, main efforts were lead during initial application components architecture design in order to make it flexible and modular.

The key was really to **anticipate** as much as possible when we built the application components architecture and object model. Redevelopment cost increases exponentially as you go to through the classic phases: development, unit test, integration test, and production. So anticipation was a key in order to start a project on the right way and create a coherent architecture that best reflects the **business logic**.

Having a flexible and modular architecture may seem nonconcrete and over-abstract, but it is really the **key** that has allowed us to build coherent base software that reduced as much as possible redevelopment and evolution costs. So actually it was really important to understand user needs at first in order to put all the chances on our side.

The software also had to be as generic as possible as it will be used in multiple CubeSats projects (not only Eye-sat). That leads us to use standards in an extensive manner:

1. XTCE standard for the System Database (SDB) - CCSDS
2. Space Packet Protocol as a base for core implementations - CCSDS
3. Packet Utilization Standard for monitoring and control definition - ECSS
4. Tm Transfer Frame standard - CCSDS
5. Tc Transfert Frame standard - CCSDS

There is a gap between respecting a standard and real world implementation. Some elements were not induced by standards (such as length of secondary headers, type of CRC, etc.) but developments were made to let them configurable in order ensure that future projects would not be restricted by it.

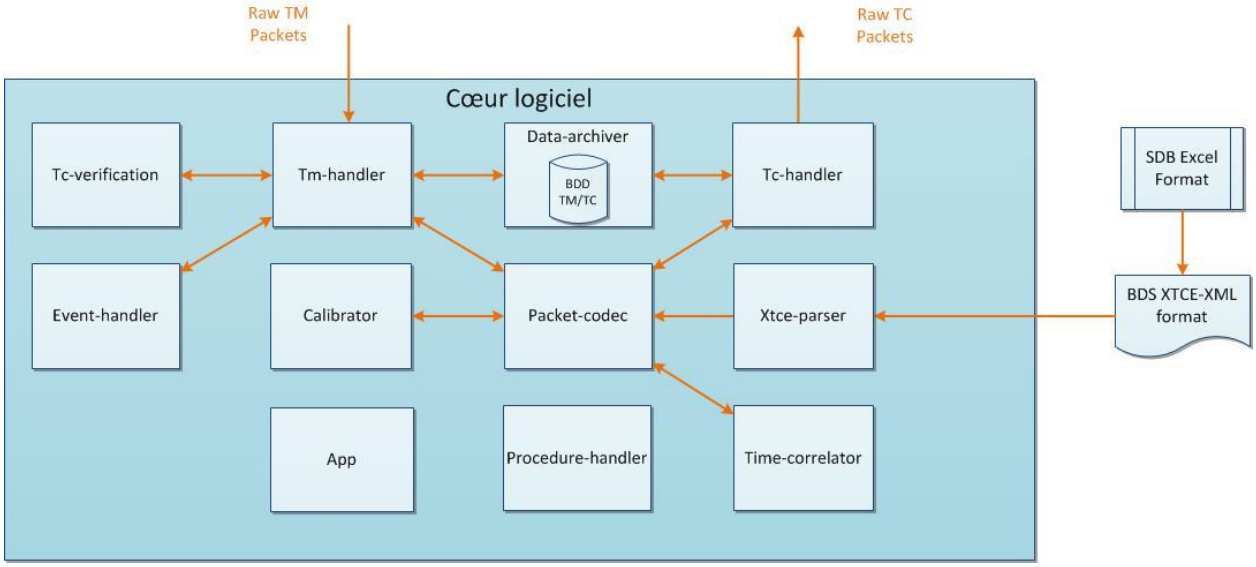
IV. A solution : Simple Control Center (SCC)

As explained before, we choose to develop a new control center software named SCC based on modern technologies in order to fulfill the needs explained above. Everything is built around **Dropwizard**[4] framework: it provides us light and production proven web server with **Jetty**, easy central configuration management system, Restful API with **Jersey** to communicate with the front-end and to export data, powerful logging system, and some other java features: **Google** Guava framework, **Joda Time** library with powerful time converters, Java Database Interface (**JDBI**) for easy database SQL access. Dropwizard is not just an assembly of existing java libraries. Dropwizard ensures the integration of this toolkit so that our applications remain permanently production-ready and stable and so allowing us to focus on the business logic. Moreover using java as numerous advantages for this type of "industry" level project:

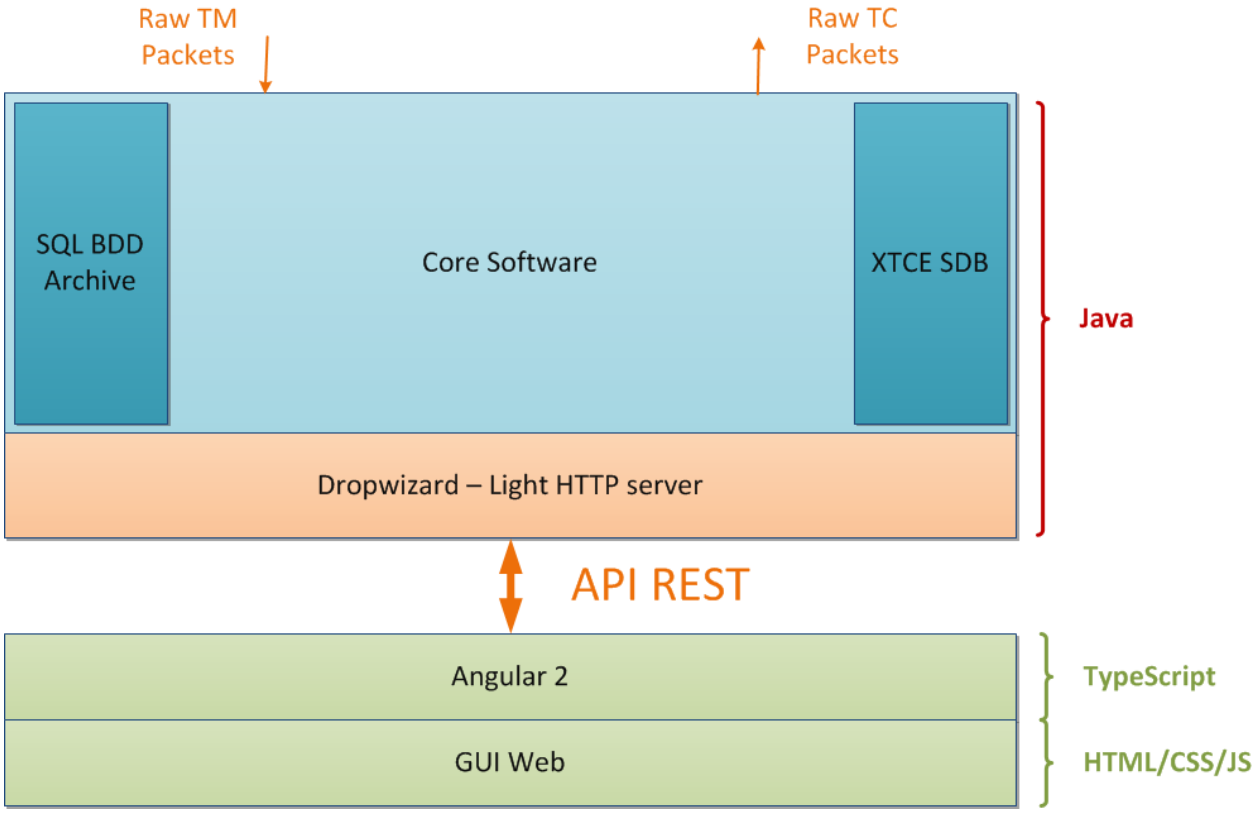
1. **Strong typing** give us a lot of security in the development process
2. **Scientific** production ready libraries richness, in particular for **space-mechanics**
3. Huge community and experience return on the internet
4. Richness and added value of industrial **level quality** tools and test framework that no other languages has reach nowadays
5. **Multi-threaded** easy to use framework
6. Project management tools like **maven**

SCC's architecture is composed by different components. Each component provides a major functionality expected by the tool. Component-oriented programming (POC) is widely used in computing. The principle is simple: the software is divided into logical elementary bricks. In this way the readability and maintainability of the code is considerably improved. Indeed, by making each logical element abstract and transparent in relation to the others, we ensure that the evolution or modification of one of the components will not impact the code of another component. This is where we get **modularity** and **flexibility**.

The POC should not be confused with the object-oriented programming (OOP), which have similarities but do not operate at the same level. The OOP is equivalent to defining modules at the level of code and computer language, while the POC intervenes at the level of the overall architecture of the system/project. Component-oriented programming is very often used in conjunction with object-oriented programming. Here is a summary of our component oriented architecture:



We are using two main technologies: **Dropwizard** with java for the back-end, **Angular**[5] with Typescript for the front-end. This architecture has proven to be very **productive** and **saves** us a lot of time. On the back-end we have the java object model which is representing the business logic, then with Dropwizard we expose the data seamlessly trough a REST API. The front-end or any other data user can consume the API. Here a summary of the technological architecture:



We are using the **MVVM** pattern (Model–View–ViewModel). It means that there is a separation between the graphical user interface (GUI) model and back-end logic (the data model). The view model is responsible for exposing the data objects from the back-end model in such a way that they are **easily managed** and **presented** for the front-end. In our case there are really small differences between the model and the ViewModel. The ViewModel is using a lot of the main **features** from the model and just turns it into object that are more easily **manageable** for the front-end.

The front-end has a Typescript object model based on the ViewModel and the back-end offers an API which is also based on the ViewModel but in java. Everything is going throw a JSON auto generation mechanism, thus the developer's work is really effortless because as long as the object from Typescript and Java matches the back-end and front-end can communicate natively without developing or adding an over-layer. The pros of choosing Typescript for the front-end are multiples:

1. **Automatic Object Mapping** makes communication with back-end seamlessly
2. Compilation language brings the force of **strong typing** to our complex application. It improves front-end debugging without having to go throw the browser to find tricky issues
3. As it compiles in JavaScript it can do native JavaScript so we can benefit from all its **richness**

V. Modern technologies in space projects

To our knowledge conventional space projects are always behind schedule in matter of **new computer science technologies** and so they make little use of it when it gets out. But there are logical reasons for that:

1. **Firstly** space projects for big satellites are usually long term project using the classic V-cycle, meaning that technologies in use on the project may have been selected years/decades before. For them it is not a problem if a software is already 20 years old, as long as it does the job.
2. **Secondly** on these projects stability and quality have always been the number one constraints and budgets have always adapted to these constraints.
3. **Thirdly** human resources have also never been a constraint and it always adapt to projects, so the high complexity of implementing and operating in production traditional old software solutions has never been an issue.

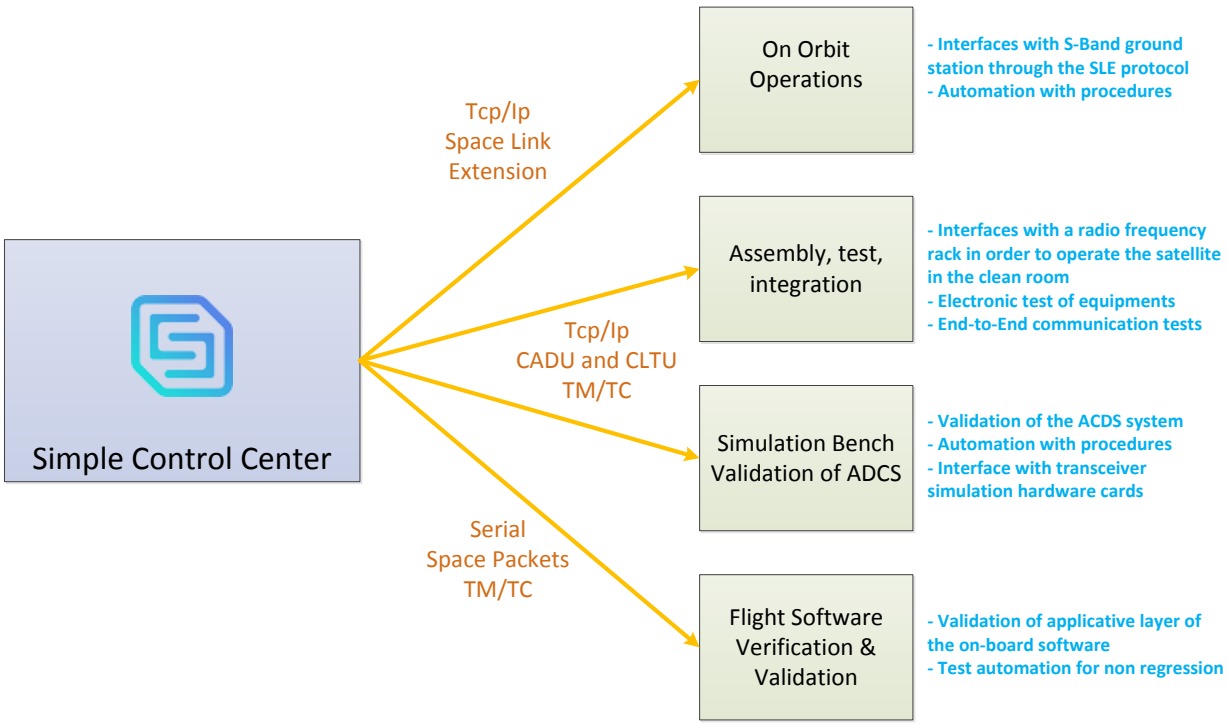
The point is that with CubeSats and nanosatellite the **paradigm** is fully reversed. The **philosophy** for CubeSat projects is the **opposite**: the main drivers and constraints on the development are to do **cheap**, **fast**, and **efficient** and with **fewer human resources**. Stability and quality are still are study of course but they are not the main drivers. This changes everything if we deal with **computer science**. Nowadays ground computer science technologies evolve at an **impressive speed**, but above all each evolution brings each time a great **added value**. That is why there is a huge interest in term of in terms of **cost reduction**, **quality** in software development, **human resources limitation** and **user experience improvement**. All these factors make even more sense when you associate them with CubeSats, because they are seeking for the same **features.**

What we learned on our projects is that space missions are really missing something huge if not using latest computer science technologies. Why? Because it brings so much more in the following fields: quality, testing environment, user experience, development environment, production environment, that using them really makes your project going to another level. It makes a huge difference both on the **mission success** probability and the **cost reduction** of the project.

Here is a non-exhaustive list of these computer science technologies we are using:

1. **Dropwizard** : with the benefits mentioned above
2. **Angular** : with the benefits mentioned above
3. **Maven** : for dependency and project management
4. **Bootstrap** front-end framework: this great library is the world's most popular front-end component library. It provide tested and ready to use component with classic dynamic behavior
5. **SonarLint** : This open-source tool automatically detects complex bugs and considerably improves code quality. It is one of the most advanced tools on this subject.
6. **GitLab Continuous Integration** platform: this is a key point, if you take the time at first on your project to setup proper continuous integration environment, it will without doubt save you more time than you could imagine. Of course setting up continuous integration is easier if you are working with the above tools like java, maven or angular.
7. **Docker:** it enables true independence between applications, infrastructure, developers, and operators. It gives the easily run continuous deployment both for **staging** and **production**.

You may wonder: we are just talking about the control center software, but why would this little part of the system have so **much impact** on the project? The answer is simple: as I said before this tool is not only used for operation, the **primary** main activity is also being a test mean for system. Actually there are numerous of tests that go through the control center software so it is actually the main testing mean of the system. Here are configuration examples of how we use it to test our satellite:



We've get to a point where the SCC software has really became a **facilitator** for us and really **saves** a lots of time and pain for our engineers. Whereas if you look to classic software on big project: some of them they really are white elephants that ask to the project more human resources, more time, and more budget than it gives you in return.

VI. Conclusion

The conclusion is a feedback about what we experienced.

1. Reusing **current knowledge** and computer science **modern technologies** in space project has a priceless value. It brings so many **added values** to the project that it should always be followed closely by space engineers.
2. The control center which can be the main testing mean must be a **facilitator** in the development process. It is highly recommended to use the **same core software** to ensure various activities: flight software verification and validation, satellite assembly, integration, and test, simulation test bench for attitude determination and control system validation, and operation once in orbit. **Sharing** these activities will both make the software **richer** and testing activities **easier**.
3. In order to make it **multi-mission** and **reusable** between projects you need to find the right balance between **functionality**, **configurability**, and **genericity**. This balance is really a very subtle point that must be addressed with patience and rigor when defining the system architecture.

References

Internet References

[1] Eye-Sat, https://janus.cnes.fr/fr/JANUS/Fr/eye-sat.htm

[2] Open MCT, https://nasa.github.io/openmct

[3] Ball Aerospace Cosmos, http://cosmosrb.com

[4] Dropwizard, http://www.dropwizard.io/1.1.2/docs

[5] Angular 2, https://angular.io

[6] QuantumCDM, http://www.kratostts.com/products/satellite-and-space/quantumcmd

[7] MVVM, https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93viewmodel