# Specification for Nanostar Software Suite (NSS)

## Nanostar Project

## Version 2.1




Thibault Gateau - Lucien Senaneuch

Thursday, $20^{th}$ June, 2019

# Contents

# 1 Introduction

## 1.1 Context: Nanostar Project

Nanostar project[1] proposition aims at supporting training and development of student nanosatellites in Europe: the challenge of the project is to provide students with the experience of a real space engineering process that includes all stages, from conception and specifications, to design, assembly, integration, testing and documentation. That is, the whole process through a network that combines high-level engineering careers and entrepreneurial ventures in the area of nanosatellites.

Nanostar will allow Southwest Europe to train students with a high level of skills in space engineering and project engineering, in order to become the future main players in the field of nanosatellites. Nanostar is funded by the Interreg Sudoe Programme[2] through the European Regional Development Fund (ERDF).

## 1.2 Context: Work package 2

The objective of this work package is to set-up the "Nanostar software suite", in order to define and implement the software tools for the CDF (Concurrent Design Facility). The definition of these "specialized" tools, after an inventory of existing software and expertise of academic partners, and testing or prototyping tools, will allow to define the needs for new tools, and the use of these tools for the preliminary design of nanosatellites. Nanostar project will provide a software suite in this sense. The development of interfaces and network /collaborative work tools will lead to the realization of different challenges. An additional contribution will consist in using design and analysis models of nanosatellites from requirements elicitation process to requirements validation process. Each academic partner will provide a set of software that can be used in the frame of this project, associated with its own expertise. It will be profitable to integrate these academic contributions with the CNES software base that is used everyday in the design real satellite projects. This requires the use of compatible languages, standardized and suitable interfaces for data compatibility and exchanges. A new contribution may require data from existing CNES software and will also provide additional results that must be compatible with other systems. For example, an academic software can provide ephemeris that have to be taken into account in IDM-CIC (CNES software) to design subsystems such as power. The integration of the overall platform will be supported by an IT firm according to the specifications

[1] `http://nanostarproject.eu/`
[2] `http://interreg-sudoe.eu/inicio`

established in WP2. Strong collaboration between academic partners will be required in order to allow an efficient process and concrete realizations.

## 1.3 *Nanostar Software Suite* Goals

In a nutshell, the *Nanostar Software Suite* (NSS) should allow a group of engineers, not fully specialized in space domain, to realize a pre-design analysis (0 phase, A phase) of a nanosatellite mission. This is usually considered as a feasibility study.

From some requirements deduced from the payload mission objective, engineers should be able to produce all classical system budgets:

- Mass budget
- Power budget
- Link budget
- Data budget
- Propellant budget
- Dissipation budget
- Radiation budget

They should also be able :

- to propose a preliminary Mechanical architecture
- to propose a preliminary Thermal architecture
- to propose a preliminary ADCS sizing
- to propose an Activity profile
- to propose a Launcher restriction choice
- to check LOS respect
- to check specific payloads constraints

A methodology has been defined to ease the process of generating these elements, Concurrent Design Facilities [Di Domizio and Gaudenzi, 2008]. Many implementations are proposed by space agencies (OCDT, IDM-CIC, CDP4, Valispace, Team-X, Virutal satellite... [ESA, 2019, Le Gal and Lopes, 2016, Rheagroup, 2019, DLR, 2019, Schreiber and Carley, 2005, Oberto et al., 2005, Valispace, 2019]).

However, to deploy them in a practical use fulfills our needs only partially. Some methodological limits must be taken into account [Braukhane and Bieler, 2014] and no tool is providing "as is" from an academic point of view all the features we are looking for. Developments are therefore required, specifically on two axis:

- the space mission database management, concerning interaction and visualization designed as **NSS-Core** Chapter 3 (p.15);
- the integration of domain specific software designed as **NSS Software Constellation**, i.e. how to make them work smoothly together Chapter 4 (p.25).

# 2 Nanostar Software Suite Overview

## 2.1 NSS concept: One Software Suite to Bind Them All

Designing a nanosatellite requires close interrelation between different fields, with respectively strong level of expertise, all the more so as development progresses. During Cubesats preliminary design many budgets are essential (mass, power, link, data, dissipation). Their local inputs, outputs and models are often intertwined. For instance, power budget rely on payload requirement, platform operational up-keeping, eclipses frequencies/duration, and batteries specifications. It will impact mass budget (e.g. number of required batteries, solar panels and wires, etc.), and reciprocally. But dissipation budget will also be concerned (batteries will only work between temperature extrema), heaters and radiators used will also impact mass budget and so on. All this process is, to our current knowledge, far to be unified with an ideal set of tools. Functionalities are often redundant between the different soft used, or even re-developed each time required rather than re-used. Fortunately some software bricks already exist, such as space mechanics libraries. Efforts on standardization are also undertaken (CCSDS) mainly concerning telecommunication protocols, and to a lesser extent, ephemeris formatting or equipment description. Even if they are paving the way for consistently interconnected suite of tools, proceeding end to end mission analysis lack of unified, consistent standards and open source tools. ESA and national space agencies are now proposing their own proprietary concurrent design facilities.

Here comes the idea for one tool to bind them all (and let's hope so not in the darkness bring them) or more precisely, a well defined suite of tools to help to get a strong consistency for a mission analysis preliminary design, which can follow the project to all it's live cycle thanks to a strong interconnection with experts tools. Optimally, each expert should be able to take (up to date) needed inputs on its own tools and provide to the team expected outputs, in a transparent way. Thanks to our previous and on-going nanosatellite projects, we have a more practical vision on specific nanosatellite project needs, and redundant software usage and developments we are used to be facing.

## 2.2 Requirements Overview

### 2.2.1 Requirements typology

- Requirement that should be respected "as is" will be designed with a bullet (•) in this document.
- Requirement that can be adapted, in accordance with the development team, will be designed with a circle (○) in this document (i.e nice to have).

### 2.2.2 Global requirements

This project is funded by SUDOE. Seven academic partners are involved. As a matter of fact, *Nanostar Software Suite* should be:

- Open source (AGPL v3)[3]
- Cross platform
- Ergonomic, with a student friendly user interface
- Use of Standards, insofar as possible
- Modular (each module should have a stand-alone version)
- Well documented (code API & quick-start guides)
- ○ Design responsive

### 2.2.3 Expected for each module of the constellation

In every module, we have these following requirements:

- independent library for any domain specific tools that can be easily called by third-party software;
- well documented API on this library;
- *Python* API for domain specific calculus;
- Python command line example using this API;
- Backend that implements this API in REST;
- *Angular* based front-end which implements the REST interface;
- modules must be open-source, with an AGPL v3 license[4];

[3]https://www.gnu.org/licenses/agpl-3.0.en.html - accessed March, $1^{st}$ 2019
[4]https://www.gnu.org/licenses/agpl-3.0.en.html - accessed March, $1^{st}$ 2019

- stand alone version of each module, working in a local way.
- code quality including test unit for each methods, integration and performance tests for whole module;
- ISAE-SUPAERO's Sourceforge platform will regroup source code deliveries and documentation, organized in one global project, subdivided in sub-projects. Each sub-project includes a git deposit.

### 2.2.4 Coordination Tasks

During whole development phases, and specially during last months we will need to:

- write integration tests for the whole NSS (core + constellation) for testing and validation purpose;
- write a global tutorial on a typical example scenario to illustrate NSS usage;
- check that CDF of each partners is operational:
  - room availability
  - all hardware installed check
  - NSS installed or at least remote NSS server accessible

### 2.2.5 Deliveries

In every module and for NSS core and modules, we have these following requirements:

- source code, organized in a standard schema:

```
.
├── doc/
├── README.md
├── LICENCE
└── src/
```

- working stand-alone example;
- quickstart guide;
- REST API documentation;
- REAMDE.md (markdown) file describing dependencies, compilation process, running example, running tests and limitations;

## 2.3 Development team

### 2.3.1 Available human resources by institution

A part of the development team is already formed, but not completed.

- One part-time IST staff member affected to the project (Paulo Oliveira);
- One full time ISAE-SUPAERO staff member is affected to the project (Lucien Senaneuch - 100%);
- One part-time ISAE-SUPAERO staff member is affected to the project (Thibault Gateau - 14%);

### 2.3.2 Sub-contractors available budgets

Sub-contractors will be hired, in accordance to workpackage 2's leaders, depending on their resources affected in workpackage 2 (see Table 1). The proposition of role affectated is visible on Table 2).

| Partner | Sub-Contractor allocated estimated budget |
|---|---|
| Bordeaux INP | 19,7 k€* |
| ISAE-SUPAERO | 65.0k€ |
| UM | 21.425k€ |
| UPM | 30k€ |
| UC3M | 20k€ |
| IST | 30k€ |

Table 1: Overview of provisional expense for sub-contractors

*: Bordeaux INP may have a bit less (not exceeding 15% of difference) according to March 2019 meeting in IST.

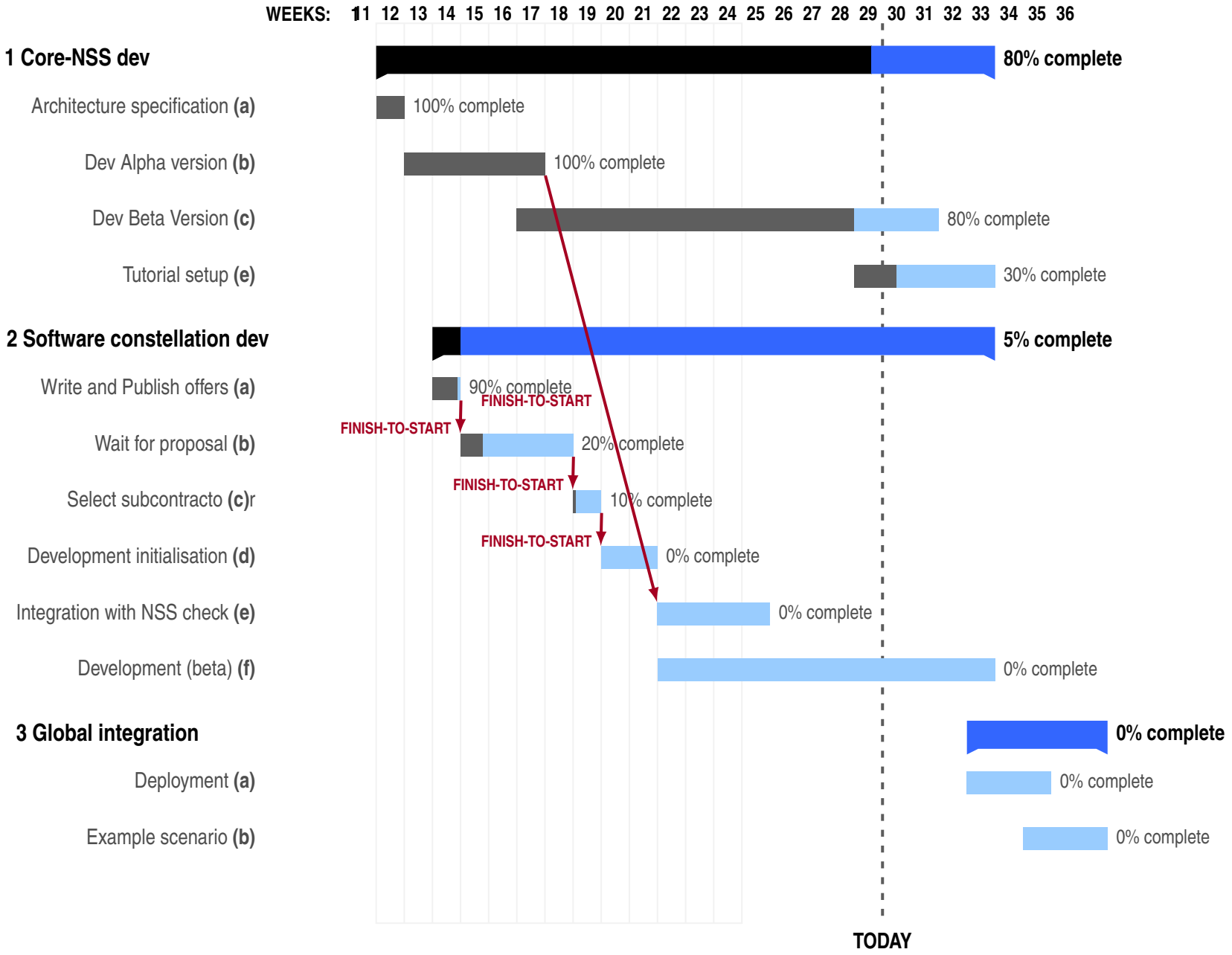
## 2.4 Development plan - NSS v1

### 2.4.1 Some key dates

- Public offers publication (April $1^{st}$);
- Subcontractors are selected (April $30^{th}$);
- Technical coordination with sub-contractors (around May, $15^{th}$);
- Technical review of on going development (in July, $31^{st}$);
- Development deadline is set to August, $31^{st}$;
- NSS availability ready for new challenges in September, $31^{st}$.

### 2.4.2 Workplan

The Gantt chart below summarizes proposed development steps. Development date start for NSS v1 has been set to February $15^{th}$ (2019). For visualisation purpose, "TODAY" is set to July $9^{th}$. Unit is counted in weeks, from February $15^{th}$ to August $31^{st}$


WEEKS: 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36
1 Core-NSS dev
80% complete
Architecture specification (a)
100% complete
Dev Alpha version (b)
100% complete
Dev Beta Version (c)
80% complete
Tutorial setup (e)
30% complete
2 Software constellation dev
5% complete
Write and Publish offers (a)
90% complete
FINISH-TO-START
FINISH-TO-START
Wait for proposal (b)
20% complete
FINISH-TO-START
Select subcontracto (c)r
10% complete
FINISH-TO-START
Development initialisation (d)
0% complete
Integration with NSS check (e)
0% complete
Development (beta) (f)
0% complete
3 Global integration
0% complete
Deployment (a)
0% complete
Example scenario (b)
0% complete
TODAY

### 2.4.3 Current Overview

Global architecture of NSS is visible on Fig. 1. Each institution is expected to have a currently working CDF Fig. 2. A set of modules have been defined as necessary for having an operational NSS (see Table 2).

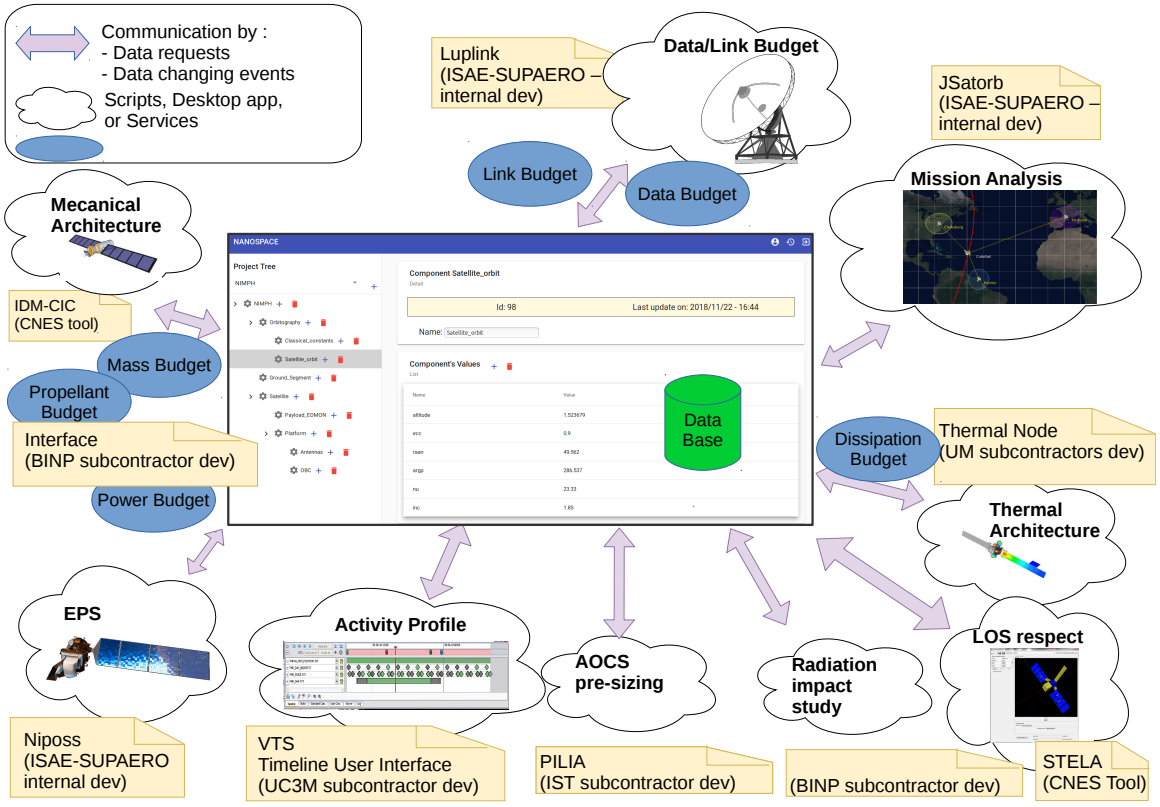



Figure 1: NSS current state, architecture overview.

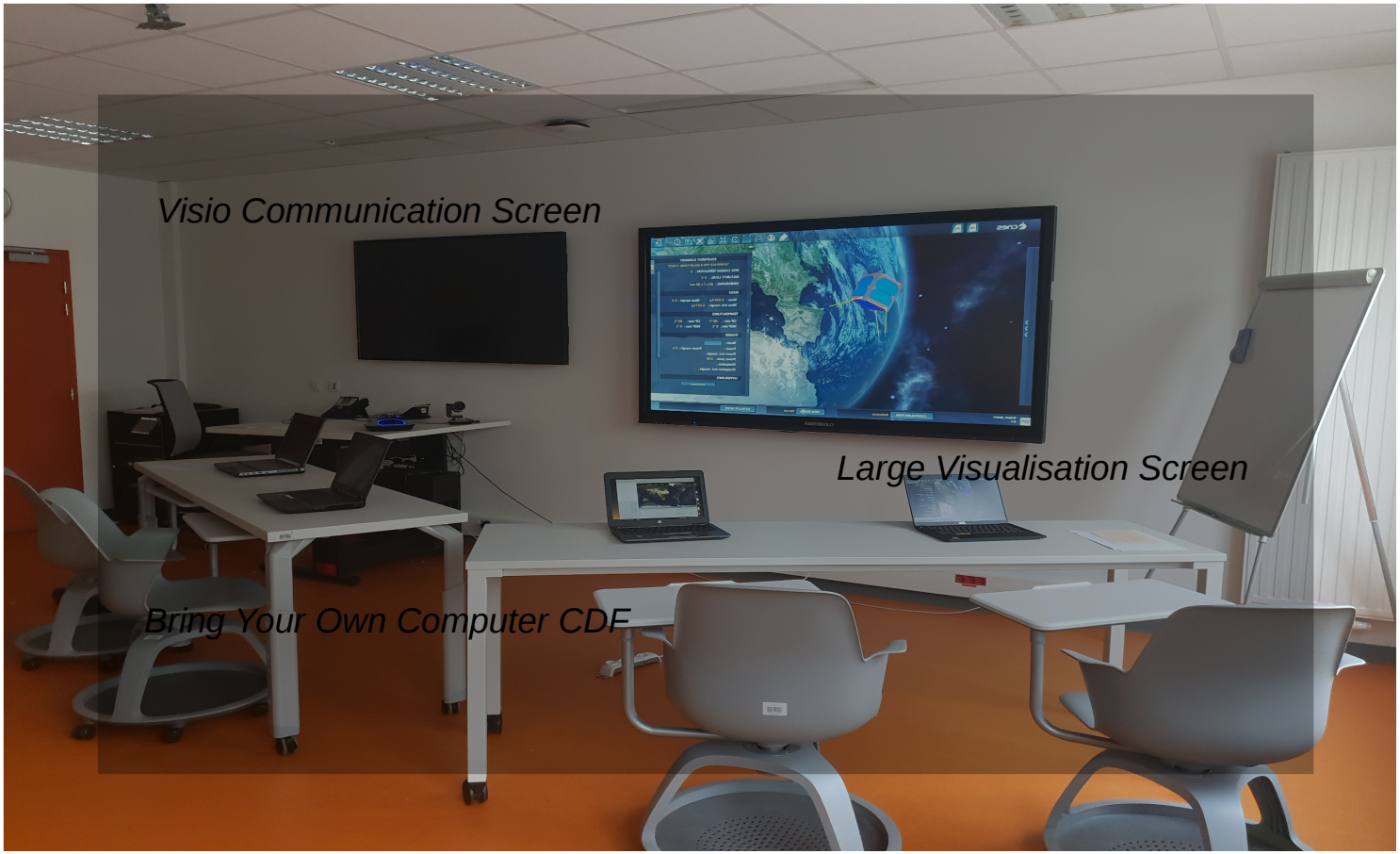



Figure 2: Example of current CDF room at ISAE-SUPAERO

| Id | Module | Base Tool | State | Responsible |
|---|---|---|---|---|
| 1.1 | NSS Core Front-end | Nanospace | dev on going [X] | SUPAERO |
| 1.2 | NSS Database imp. | based on neo4j | dev on going [X] | SUPAERO |
| 2.1 | Mass budget module | IDM-CIC | dependency [2.7] | - |
| 2.2 | Power budget module | IDM-CIC | dependency [2.7] | - |
| 2.3 | Propellant budget module | IDM-CIC | dependency [2.7] | - |
| 2.4 | Link budget module | Luplink (JSatorb) | dev on going [X] | SUPAERO |
| 2.5 | Data budget module | Luplink (JSatorb) | dev on going [X] | SUPAERO |
| 2.6 | Thermal architecture | None Yet | dev to do [X] | UM/UC3M |
| 2.7 | Interface with IDM files | REST API | dev to do [X] | BINP |
| 2.8 | Radiation budget module | None Yet | dev to do [X] | UPM |
| 2.9 | Mechanical architecture | IDM-CIC | dependency [2.7] | - |
| 2.10 | Preliminary AOCS sizing | None Yet / Pilia | dev to do [X] | IST |
| 2.11 | Activity Profile | Nanopower/VTS | dev to do [X] | UC3M |
| 2.12 | check LOS respect | Stela | done [✓] | - |
| 2.13 | Visualization | VTS/IDM-View | dependency [2.12] | - |
| 2.14 | EPS sizing module | Nanopower | dev on going [X] | SUPAERO |
| 2.15 | Mission Analysis | JSatorb | dev on going [X] | SUPAERO |
| 2.16 | Broker Choice | None Yet* | optional | - |
| 3.1 | Constellation Integration | - | dev to do [X] | SUPAERO |
| 3.2 | Partners Synchronization | - | dev to do [X] | SUPAERO |
| 3.3 | Full working scenario | Quickstart guide | dev to do [X] | SUPAERO |

Table 2: NSS modules state development overview

*: `isisspace.nl` is already providing a databases.

# 3 NSS-Core

## 3.1 Respect of existing standards

**Databases**:

A standard is proposed for data that are manipulated in a CDF: ECSS-ETM-10-25A. In practice, the document describing this standard is not directly accessible [5]

- Definition and vocabulary of ECSS-ETM-10-25A should be respected to be consistent with space agencies [European Cooperation for Space Standardization, 2019].
  - A prototype of the database management has been implemented in neo4j[6] (Fig. **??**).

**Protocol exchange**:

  - Domain specific softwares can be seen as REST components for NSS-Core;
  - JSON format for data exchange is favored.

## 3.2 NSS-Core Overview

NSS-Core includes a connected database that allow to store a satellite model. It's located on the Back-end (see Fig. 3). This database is accessible through a software component called "Model Manipulation Service" . A User Interface (UI) is also available to ease human interaction with this database. The main goal of the UI is to visualize and to modify the element stored. This set of tools form a self-sufficient software. It provides a full and a synthetic system view thanks to the centralization of the whole model's data.

## 3.3 Nanospace functionalities

This is a list of functionalities :

- the data model consists of:
  - *project* storing elements that will constitutes the model;
  - *components* composing Project;
  - *mode* describing the different functional options of a component;
  - *value* characterizing mode (and characterizing indirectly his component);
  - *user* who is responsible of the project or elements;
- 4 types of *values* are allowed:

[5] `https://ecss.nl/standard` - Accessed January 31$^{st}$
[6] `https://neo4j.com/` - Accessed January 31$^{st}$

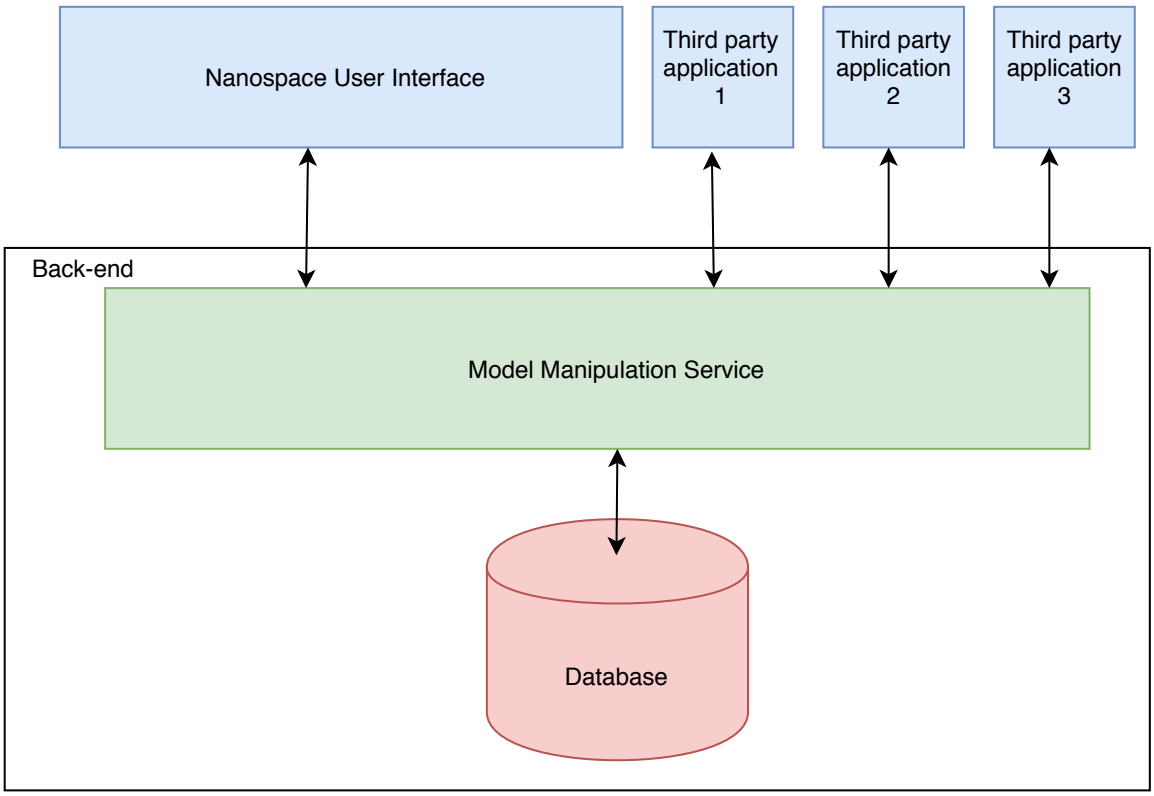



Figure 3: NSS-Core Overview.

  - formula storing calculations or numbers;
  - static value stored as text or string;
  - matrix storing formula or string values;
  - requirement storing a condition which can include formula or numbers; values ( boolean result);

- on the user interface, a user should be able to:
  - authenticate himself for accessing to the application and the projects he is responsible of;
  - add other user as responsible;
  - create and modify a project or a model (project composition of element);
  - characterize the model with values;
  - create requirements based on model characteristics;
  - use classic functionalities such as copy, drag-and-drop, auto-completion;
  - create one or several modes for each components;
  - access to all the requirements;
  - export or import the entire model on a system file;
  - be notified when a modifications is occurring on the data displayed by the UI;
  - access to an history of the past modifications;
  - be notified of a required change whether if the data are displayed or not;

- third-party programs can also access to the database:
  - authenticate themselves;
  - create, modify, or delete elements of the model[7];
  - cannot create requirements;
  - cannot create project.

A NSS-core use case is illustrated below (see fig Fig. 4).

## 3.4 Software Architecture description

The general architecture consists of 3 main components (see Fig. 5):

- The database (in the server side): data manipulated can easily be represented in the form of a hierarchical tree with variable depth. To increase performance when handling this type of data, it is stored in graph form. In addition, data must be write-protected in order to allow access competition. Neo4j fills these requirements. In

[7]Each modification of the database is logged

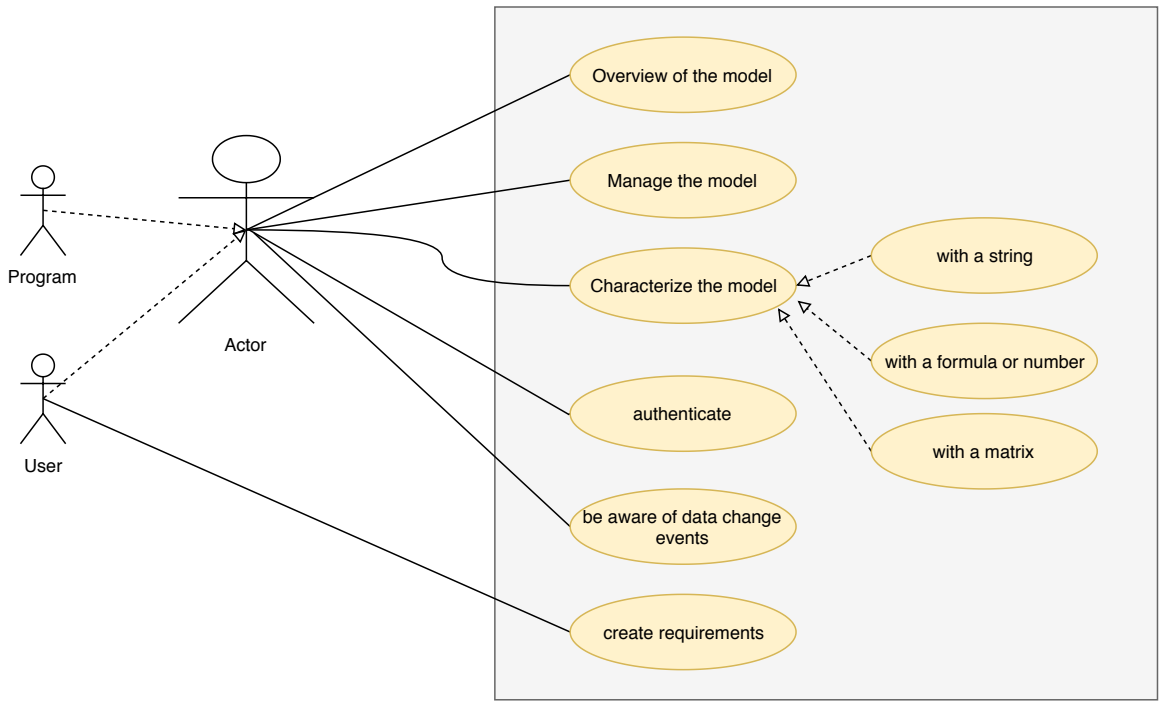



Figure 4: NSS-core use case

addition Neo4j has ACID[8] properties. It is a NoSQL database storing graphs. Neo4j has also a larger community than other databases such as OrientDB [Vergnes, 2015];

- The graphical user interface (GUI) : this is the main application for interacting with the model. It is based on web technology since we still want a cross-platform access without any installations from client side;
- A model manipulation service component that allows applications (webpage GUI, scripts, etc.) to manipulate the model. This service allows specific domain applications and user interface to communicate with the database. The technology chosen depends of the facility of the database manipulation. Neo4J's advise is to use Java which allow to manage it easier with a higher abstraction level. Spring java framework is therefore the most logical candidate.

Communications between third party services and the application need also to be taken into account, to ease access to the database for any type of program and language (Python, C, C++, java, Go, Octave, Julia...). An interface with a high level of abstraction is necessary. In this purpose, we have chosen to be oriented as web applications, using HTTP requests and respecting REST (REpresentational State Transfer) convention [9].

Additionally, we have to deploy additional components (see Fig. 5):

[8] Atomicity, Consistency, Isolation, Durability

[9] `https://www.ics.uci.edu$\sim$fielding/pubs/dissertation/rest_arch_style.htm`

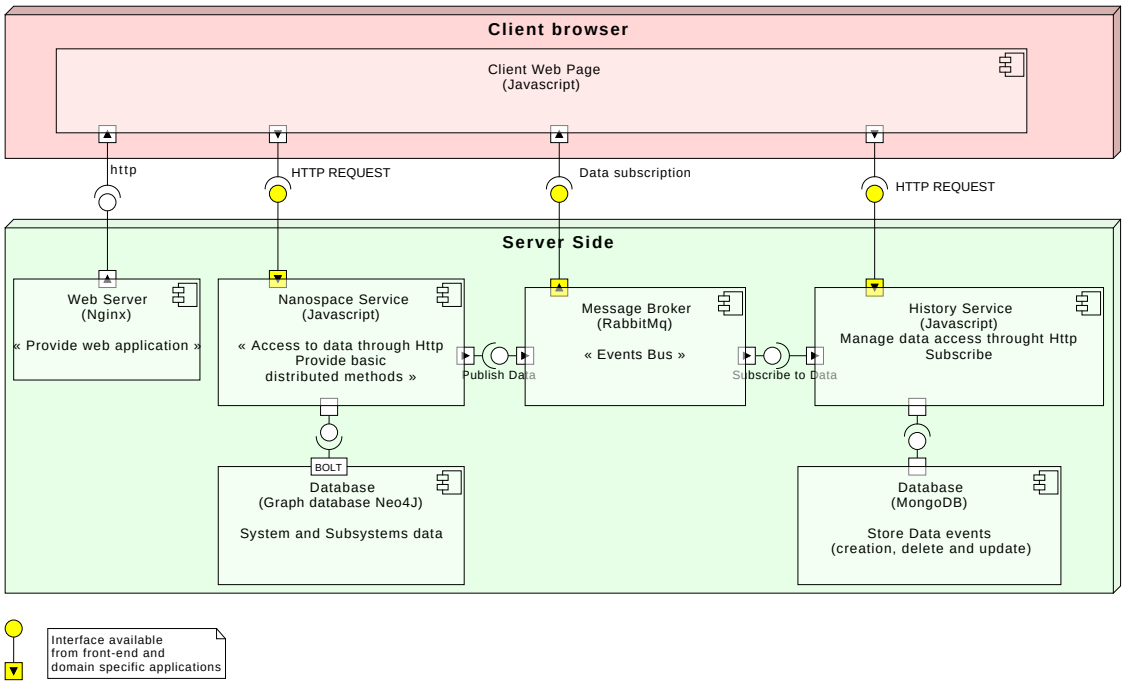



Figure 5: NSS Core architecture.

- an HTTP server (Nginx or Apache) allows to serve the main graphical interface;
- a Broker message deals with asynchronous events between each application (RabbitMQ);
- we choose to store the data on a MongoDb database which allows to store documents (each document corresponds to a dated event having circulated on the bus).

### 3.4.1 Prototyping and technical notes

A prototype has been developed in order to show the feasibility of the product on the following features:

- view a model as a tree;
- creation and Modification of project and model;
- sign in;
- be aware of data changes through historical.

Only the service layer was originally developed using Node.js[10] language. However, Neo4j does not offer a software layer to map requests as objects with Node.js. This feature,

[10] `https://nodejs.org/en/`

which will save time on future developments, is only available in python (community code) or Java (code managed by Neo4j).

### 3.4.2 Data model

In the proof of concept the initial choice was to be satisfied with a reduced number of entities in the database. Only four entities ("User", "Project", "Components", "Values") have been taken into account. These four entities are insufficient and it is necessary to make the model more complex in terms of database. Since data is stored in the database in a graph form, the graph below models the entities as nodes of the graph (see the model on Fig. 6 and an example of implementation Fig. 7).

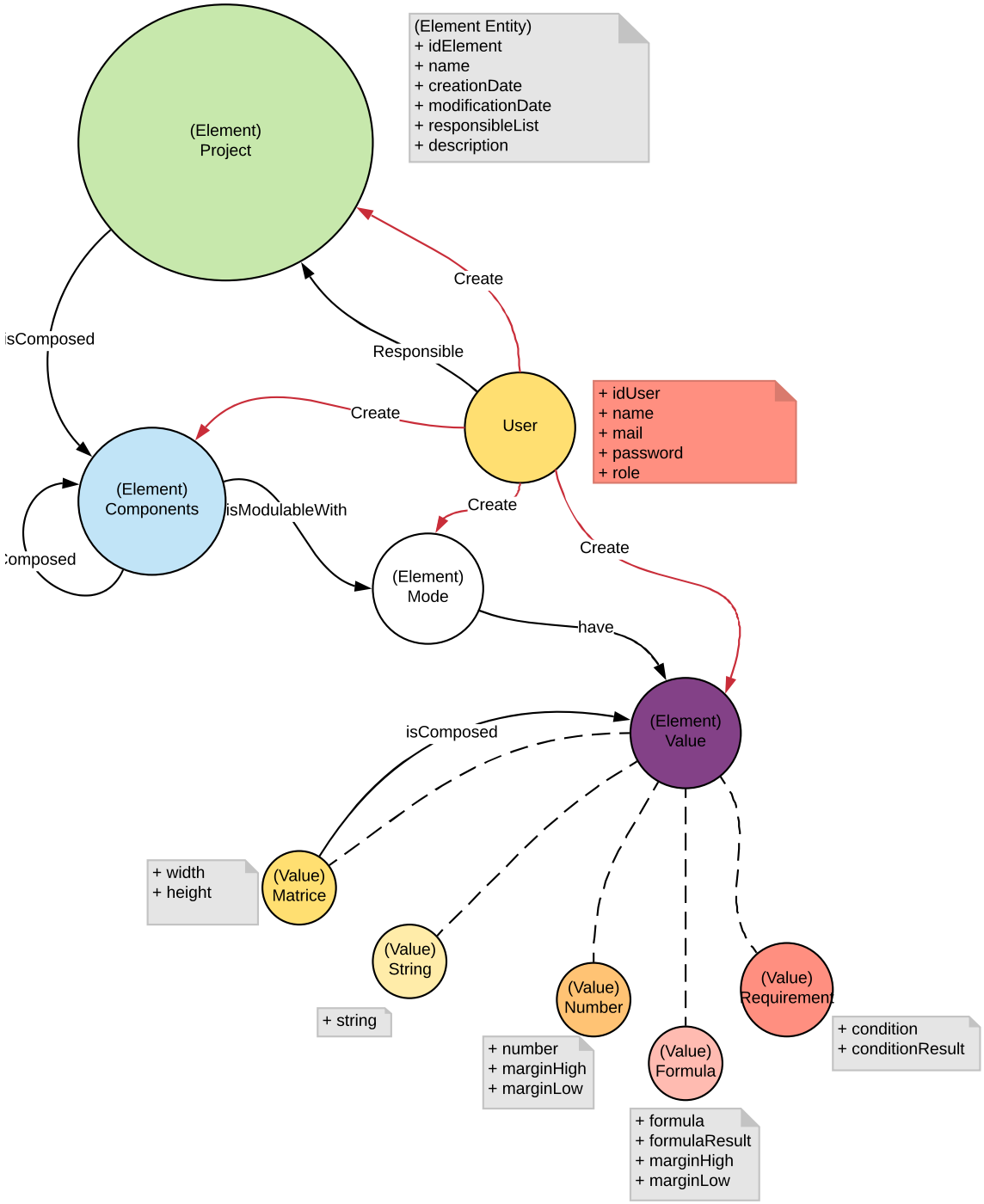



Figure 6: Neo4j based database model prototype.

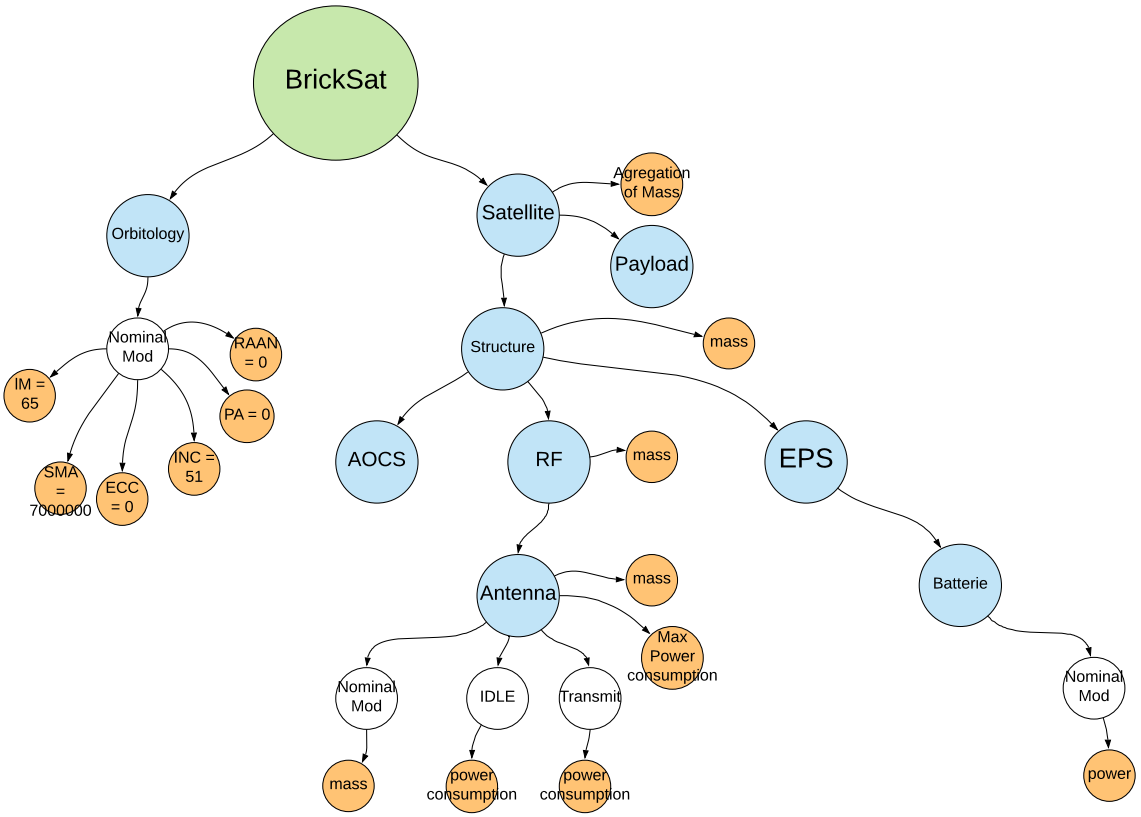



Figure 7: An example of implementation of the model.

### 3.4.3 User interface

A basic wireframe interface show the kind of interaction the user could have (see Fig. 8 to Fig. 11.


Figure 8: User has just loaded the project.


Figure 9: User can access to a context menu to modify the model.

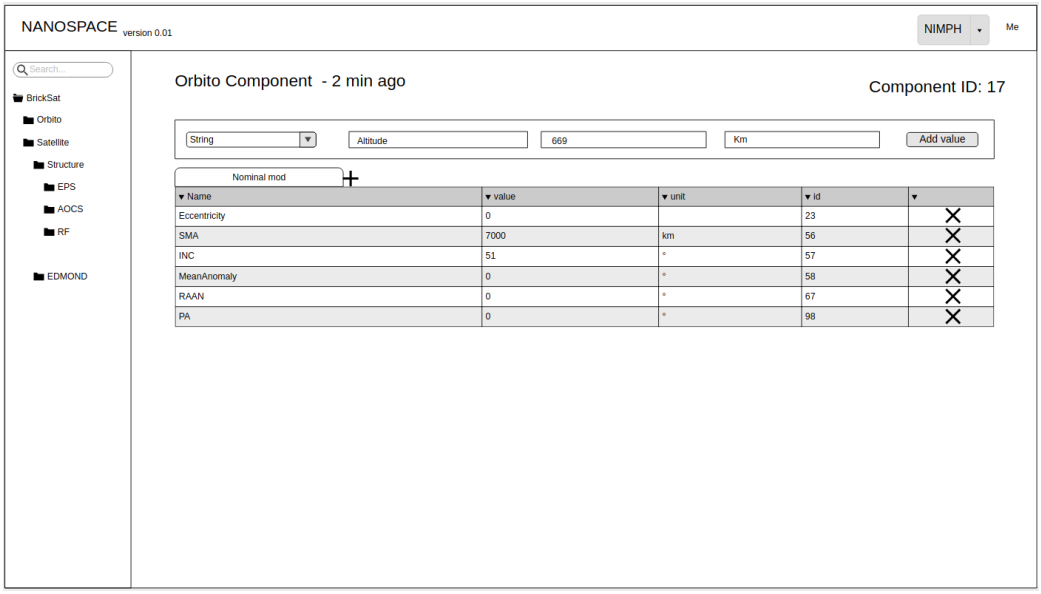



Figure 10: By clicking on a component a user can display characteristic values.

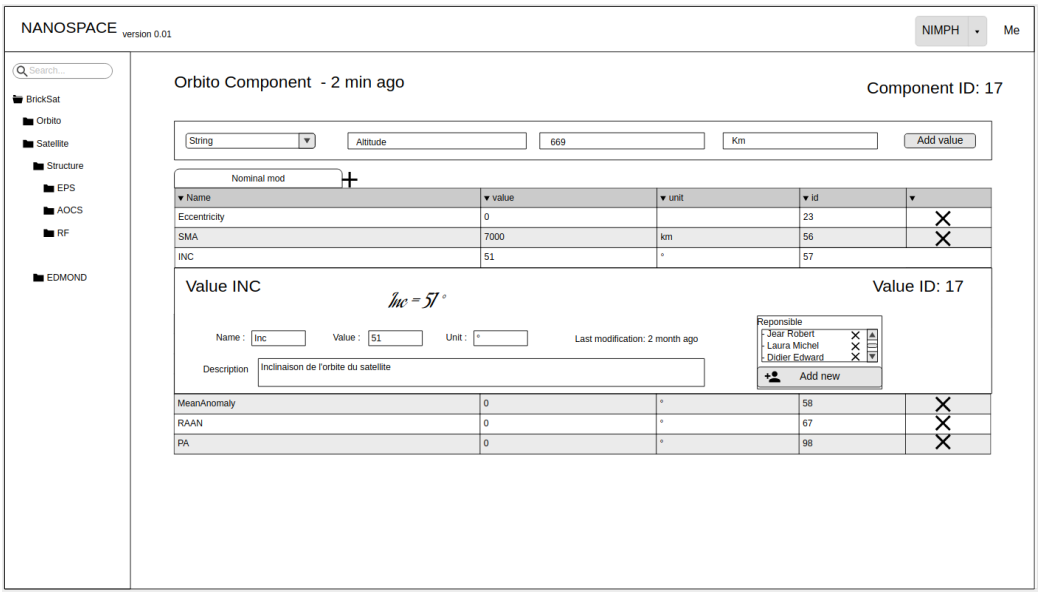



Figure 11: By clicking on a value the user toggle a menu to modify it.

# 4 Domain specific software constellation

## 4.1 Existing materials

### 4.1.1 Existing standards

- OEM files [Ephemeris - List (stellar object,date,position,velocity)]
  (CCSDS Orbit Data Message)
- AEM files [Attitude profile - List (date,quaternions)]
  (CCSDS Attitude Data Message)
- MEM files - xml format - description of spacecraft sub-components, mechanical structuring, modes, power consumption according to modes, etc.
  Not clear, but IDM-CIC files seem to respect this standard.
- ECSS-E-TM-10-25A [European Cooperation for Space Standardization, 2019]
- no real standard for geometric data but consensus around STEP file
- EDS (Electronic Data Sheet) for equipment.

### 4.1.2 Existing and operational software: NSS v0

**Mission Analysis**:

*Orbit definition, surface coverage, visibility windows, eclipse calculation, etc.*
* Recommanded
Celestlab - https://logiciels.cnes.fr/fr/content/celestlab
GMAT - https://software.nasa.gov/software/GSC-17177-1
VTS - https://logiciels.cnes.fr/fr/node/19?type=desc

* Alternatives
Orekit - https://www.orekit.org/
Patrius - https://logiciels.cnes.fr/fr/node/61?type=desc
PSIMU - https://logiciels.cnes.fr/fr/node/97?type=desc
Genius - https://logiciels.cnes.fr/fr/node/75?type=desc
Docks - https://cceres.psl.eu/spip.php?rubrique37
Astropy - http://www.astropy.org/
Satorb - https://sourceforge.isae.fr/projects/dcas-soft-espace/repository/raw/supportLight/Satorb_V07_setup.exe

**Structure - Power Budget - Mass Budget**:

*Allow to build a light 3D structural model, defining subcomponent and their respective power and mass. Spacecraft mode can also be detailed, impacting power consumption (and dissipation) of each subcomponents.*

* CNES IDM-CIC [Le Gal and Lopes, 2016] can be directly used to get mass and power budgets. Thanks to its internal subsystem component definition and Sketchup tools, it's also possible do design the Mechanical architecture.

*Access request must be done to your local supervisor (Access is restricted to Nanostar Members)*

**Dissipation Budget**:

*Allow to characterize temperature reached on each node of the spacecraft. Allow to check that temperature operating range of each equipment is respected.*

* Manual Scripting (Scilab, Python, Octave...)

**Link Budget - Data Budget**:

*Estimation of the margin for uplink and downlink connection with earth must be provided. That allow to size the useful data flow that can be exploited. Required on board data must also be sized according to last parameters and data budget.*

* Amsat Excel Sheet - https://amsat-uk.org/tag/link-budget/
* and Manual Scripting (Scilab, Python, Octave...) * Satorb - https://sourceforge.

isae.fr/projects/dcas-soft-espace/repository/raw/supportLight/Satorb_V07_setup.exe

**Visualization**:

*Visualization allow team to communicate internally, to check and validate mission analysis and scenario and to get pretty communication outputs.*

* VTS - https://logiciels.cnes.fr/fr/node/19?type=desc
* Celestia (included in VTS) - https://celestia.fr/
* CNES - IDM-VIEW
*Access request must be done to your local supervisor (Access is restricted to Nanostar Members)*

**End Of Life**:

*Nanosatellite sent in LEO should re-enter into the atmosphere in less than 25 years after the end of operational activity.*

* Stela - https://logiciels.cnes.fr/fr/content/stela

**Project Management**:

*Some recommendations for managing your project: work with a versioning system, often included in a Project Software Management tool that allow efficient and reactive teamwork... and communicate with any useful means...*

* Versioning tool
Subversion (SVN) - https://subversion.apache.org/
or Git - https://git-scm.com/

* Project Software Management tool
GitHub - https://github.com/
SourceForge - https://sourceforge.net/

* Internal communication
Slack - https://slack.com/signin

## 4.2 Specification for constellation software developments

### 4.2.1 Mission Analysis

**Usage**:

The module should allow to:

- set a list of spacecraft, with keplerian parameters or TLE
- ◦ provide a list of classical orbits (Geostationary, Sun synchronous, Critically inclined, Sun synchronous, Repeating ground trace, Molniya, Critically inclined, Circular...)
- import existing TLE from Celestrak `https://www.celestrak.com/`
- set a list of spacecraft constellation (groups of spacecrafts)
- export ephemerids data following CCSDS format of each spacecraft
  - in a local file
  - in json, through http REST
- calculate visibility windows for ground stations
- calculate eclipse (and penumbra) temporal windows for each spacecraft
- visualize spacecraft's orbit
  - in a 3D view (inertial frame)
  - on a planet projected 2D view (including satellite ground track, station visibility indicator, visibility cone)
- set a minimum elevation parameter for the ground station
- provide coverage estimation for planet observation missions
- all outputs must be available on a REST documented API, in json format
- allow to run and visualize the simulation: play, stop, pause, forward or backward. Start date and the time step (simulation speed) can be be selected.
- Set orientation vectors to Spacecraft/Stations, Earth centre, Sun, and/or Moon, or any celestial body center
- export attitude data following CCSDS format for each spacecraft
  - in a local file
  - in JSON format, through http REST service

**Technical proposition**:

ISAE-SUPAERO JSatorb mission analysis tool may be adapted for such needs. It's Open Source `https://sourceforge.isae.fr/projects/jsatorb-all` and already contains most of the features required.

JSatOrb is an ISAE-Supaero's software tool dedicated to orbital calculation and designed for pedagogical purposes, with professional level features outputs. It has been initiated to find a soft which would fill the gap between local teachers developed tools and professional tools, exploiting state of the arts algorithms concerning space mechanics calculus. Even if current provided open source libraries are not fully compliant with our pedagogical requirements (simplicity, flexibility, multi-platform and ergonomics), they provide complete and accurate calculus methods that are dedicated to a professional use. However, GUI part is not the main concern when used by space engineers which only require API access.

Concretely, JSatorb project is open-source (LPPG v3 license) and under development. It is inspired from current full-stack implementation methods. Ergonomic and intuitiveness are at stack concerning the front-end, which is mainly based on Angular (`https://angular.io/`) and Cesium (`https://cesiumjs.org`). Efficiency and correctness on calculus are provided by the back-end part, which relies on Orekit (`https://www.orekit.org/`).

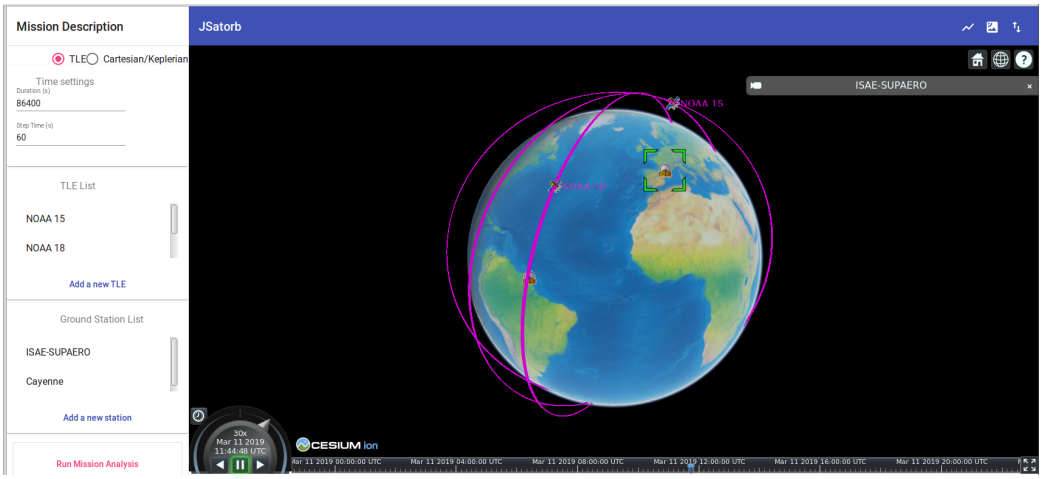



Figure 12: Illustration, JSatorb 3D view

**Tasks to be achieved**:

- test and validation of existing prototype, for each functionalities already developed
- surface coverage features to be developed, tested and validated (very classical problematic in earth observation mission)
- interplanetary trajectories feature to be developed, tested and validated

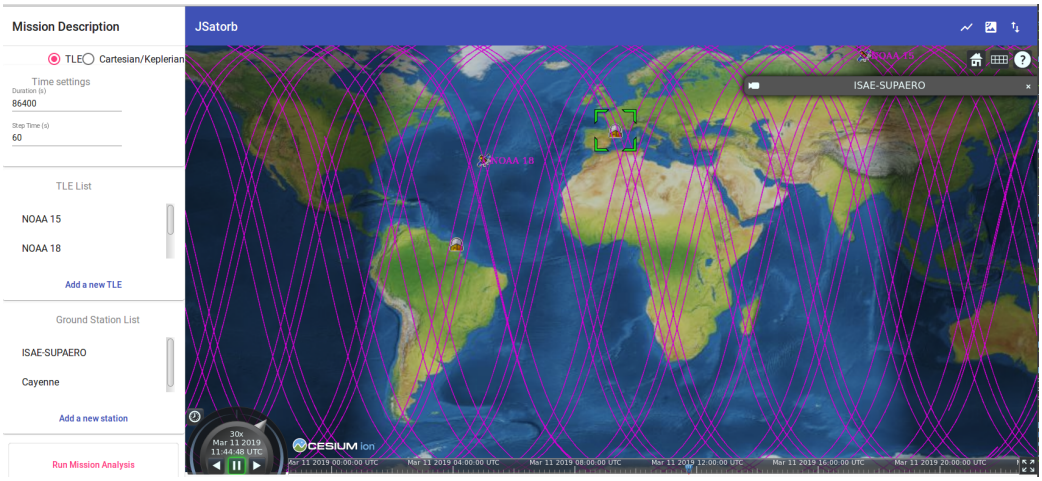



Figure 13: Illustration, 2D planisphere view with satellite traces, ground stations and day-night frontier visualization

- examples (SSO, geostationary, orbit around Mars...) should be available
- quick-start guide must be provided
- REST API must be formalized and provided
- visualization features should be improved (e.g. more frames)

### 4.2.2 Structure Module

**Usage**:

The module should allow to:

- build the first version of the 3D structure of the spacecraft and payload
- specify material types
- import already existing 3D structures

**Technical proposition**:

CNES's IDM-CIC tool.

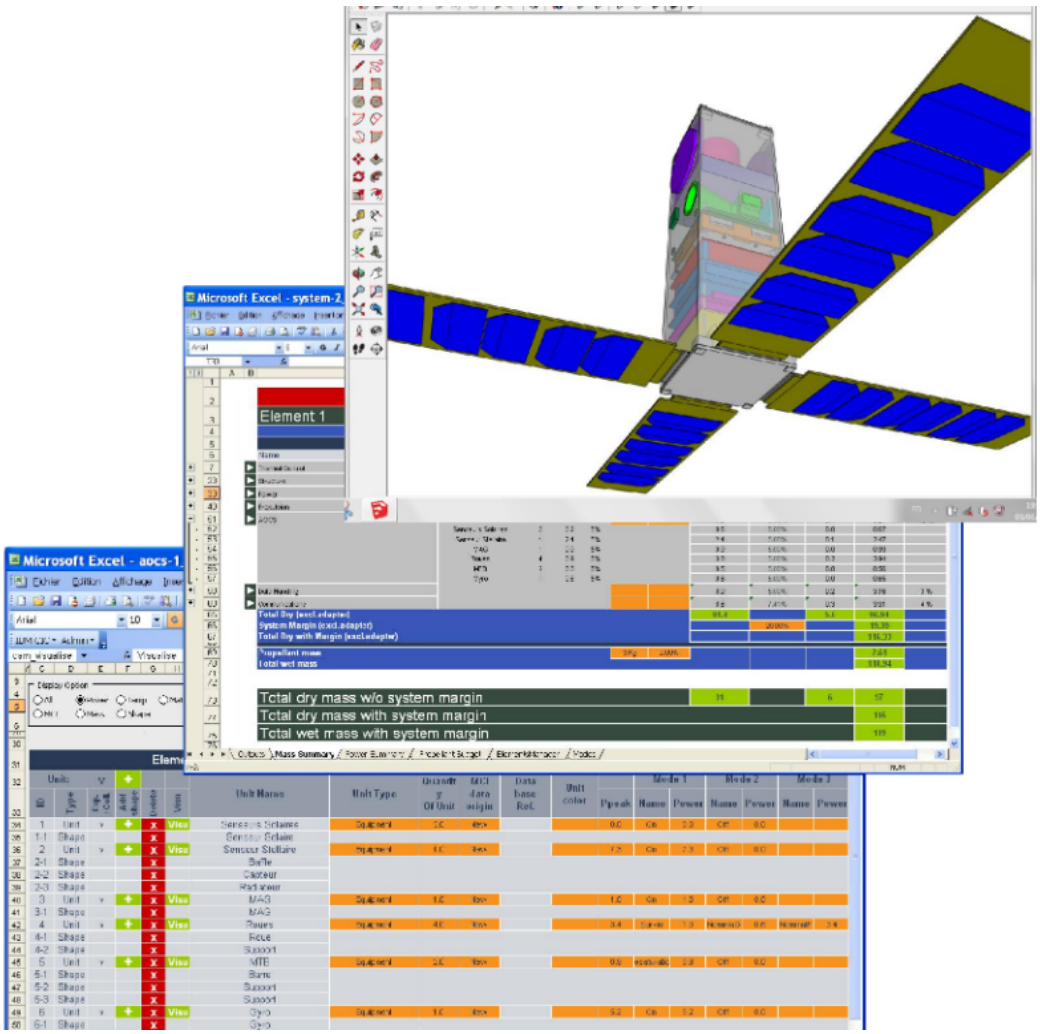



Figure 14: IDM-CIC excel interface

### 4.2.3 Link Budget Module

**Usage**:

The module should allow to:

- set the RF parameters for the
  - spacecraft
  - ground station
  - propagation path
  - orbital parameters

  in both cases:
  - uplink
  - downlink
- calculate a link-budget according to a BER requirement
- calculate a fade margin
- calculate raw data flow
- calculate useful data flow

**Technical proposition**:

Luplink project (ISAE-SUPAERO) [Ningaraju, 2018] has provided a set of libraries (Octave and Python) for basic and advanced calculus for link-budget. Space mechanics calculus are provided by Celestlab library (a Scilab CNES tool). A proof of concept prototype has validated the technical aspect regarding Nanostar global specification for modules (REST API, modularity, open source...) Light effort would be necessary to provide an operational module.

**Tasks to be achieved**:

- test and validation of existing prototype, for each functionalities already developed
- add fade margin calculus
- end the implementation of angular front-end part
- validate all calculus

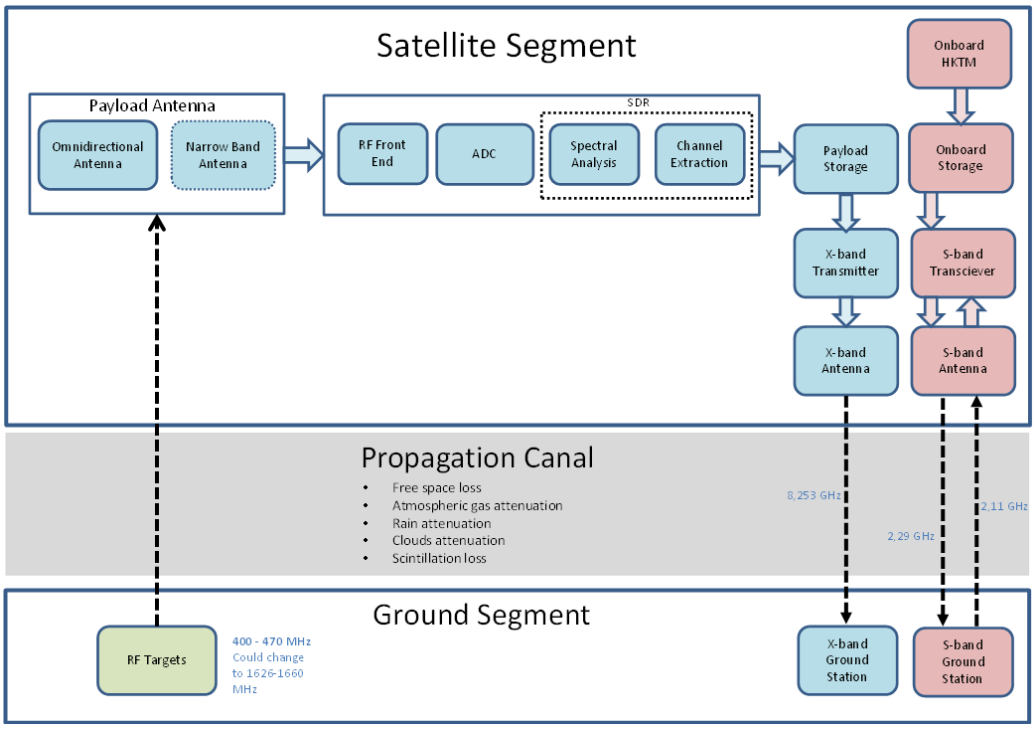



Figure 15: Nanosatellite Spectra project data chain - [Ningaraju, 2018]

### 4.2.4 Data Budget Module

**Usage**:

The module should allow to:

- set the useful data flow
- set the visibility windows
- update required on-board storage
- compute the data budget

**Technical proposition**:

Luplink project (ISAE-SUPAERO) [Ningaraju, 2018] has provided a set of libraries (Octave and Python) for basic and advance calculus for link-budget, which can be easily extended for data budget calculation. A proof of concept prototype has validated the technical aspect regarding Nanostar global specification for modules (REST API, modularity, open source...) Light effort would be necessary to provide an operational module.

**Tasks to be achieved**:

- test and validation of existing prototype, for each functionalities already developed
- end the implementation of angular front-end part
- validate all calculus

### 4.2.5 Thermal Architecture

Cf. UM's specification document: "NANOST-REQ-045_GT2 IDM-CIC_Thermica specs from UM_v2.0.docx", available on the Aerospace Valley sharepoint. We must take care of general specification that must be respected by all modules.

**Subcontractors possibilities**:

Subcontractor: **Clever Age**
Justifiction: developpers of IDM-CIC Excel Plugin
contact: <gchaffarod@clever-age.com>

Subcontractor: **Airbus**
Justifiction: developpers of Systema-thermica
contact:
<http://www.systema.airbusdefenceandspace.com/support.html>

### 4.2.6 Interface with IDM-CIC

**Usage**:

The module should allow to:

- provide whole or partial data values from IDM-CIC file for third party software
- update whole or partial data values in IDM-CIC file for third party software
- provide a front-end that make this manipulation available

**Technical proposition**:

- provide a library providing a REST API for IDM-CIC model (probably in .NET)

**Tasks to be achieved**:

The module should allow to:

- provide a REST API to manipulate IDM-CIC files:
  - access to stored data
  - modify stored data
  - create IDM-CIC from NSS database
  - update NSS database from IDM-CIC files
- provide an Angular front-end GUI

**Subcontractors possibilities**:

Subcontractor: **Clever Age**
Justification: developers of IDM-CIC (Excel Plugin and IDM .net library)
Contact: <gchaffarod@clever-age.com>

Subcontractor: **Virtual It** [11]
Justification: developers of IDM-VIEW

#### 4.2.7 Radiation budget module

**Usage**:

The module should allow to:

- read an OEM file
- provide an estimation of amount of radiation received by the spacecraft
  - read an AEM file
  - read a simplified 3D model of the spacecraft nodes, in the same idea as Thermal modeling (Cf. section 4.2.5)
  - calculate cumulated radiation received by each nodes
  - calculate required shielding for each nodes

**Technical proposition**:

[11] https://virtual-it.fr/en/portfolio-posts/idm-view-cnes/

- A qualitative estimation is a good start
- **Artenum** work with Onera lab on Open Source library
- Fastrad[12] software may offer a solution depending on license usage

**Tasks to be achieved**:

- Define the development strategy

**Subcontractors possibilities**:

Subcontractor: **Artenum** [13]
Justification: Scientific computing, library open source on radiations

Subcontractor: **Trad** [14]
Justification: Developpers of Fastrad, 3D radiation software

### 4.2.8 Preliminary ADCS sizing

**Usage**:

The module should allow to:

- read an OEM file
- read an Activity Profile file
- read an Activity Profile REST API (Cf. section 4.2.9)
- read a formal description of equipment (inertial wheels, magnetorquer...)
- provide a validation of mission feasibility in term of attitude control
- provide an attitude profile (AEM) file
  - read a structural model (Cf. section 4.2.2)
  - connect to a visualization broker such as VTS (Cf. section 4.2.9)

**Technical proposition**:

[12] http://www.trad.fr/
[13] http://www.artenum.com/EN/index.html
[14] http://www.trad.fr/

- ISAE-SUPAERO's PILIA internal tool can be made available

**Tasks to be achieved**:

- refer to the activity profile format (Cf. section 4.2.9)
- check mission requirements against material characteristics (required precision for attitude control, couple required for wheel desaturation, etc.)
- provide a attitude control model (instantaneous idealistic world, simple model of each element to provide a simulation )

### 4.2.9 Activity profile management

**Usage**:

The module should allow the user to:

- define different activity profiles for a same mission
- define the different modes of the spacecraft, and associated parameters
- allow to update dynamically parameters of these two types of data (e.g power consumption may depend of the mode, etc.)
- provide tools to visualize these profiles in an efficient and ergonomic way

**Technical proposition**:

- should be totally compatible with VTS broker (TCP sockets connections)
- CSV formatted files should be provided
- IDM-CIC is already taking into account mode definition

**Tasks to be achieved**:

- formalize an activity profile - list of couple (time, type of activity)
- formalize satellite mode data
- define an angular web interface making easy to describe an activity profile
- define an angular web interface making easy to describe the different satellite modes
- implement it in Angular
- provide a library implementing functionality for manipulating it
- provide a REST API according to this library

#### 4.2.10 EPS module

**Usage**:

The module should allow to:

- read an activity profile
- formalize a model of consumption according to satellite mode - list of couple (mode, data)
- define an angular web interface making easy to describe a model of power consumption according to satellite mode
- read a consumption model (power consumption according to mode)
- read material specification (characteristic of solar cells used, of battery power used...)
- read an AEM file
- read an OEM file
- propose a format to describe a model of power generation for solar cells
- read a model of power generation for solar cells
  - read a solar flux model
- provide a power budget (on each time step, power production minus power consumption)
- provide DoD (Depth of Discharge) budget (on each time step, DoD value)
- compute estimated number of batteries required (according to battery characteristic, activity profile and solar cells number and characteristic)
  - compute estimated number of solar cells required (according to solar cells characteristic, activity profile and solar cells number and characteristic)

**Technical proposition**:

ISAE-SUPAERO Nanopower mission analysis tool may be adapted for such needs: a Generic Cubesat Power Simulator [Perea, 2019].

**Tasks to be achieved**:

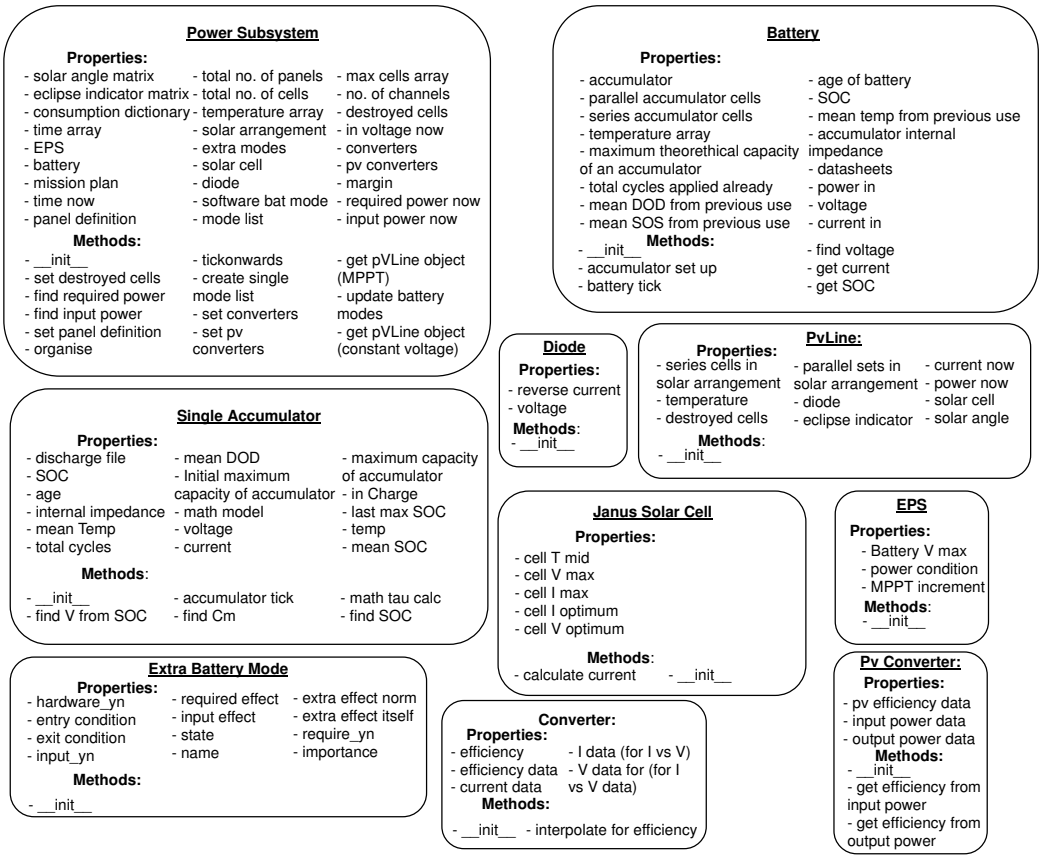



Figure 16: Structure of the Generic Cubesat Power Simulator [Perea, 2019]

- test and validation of existing prototype, for each functionalities already developed
- add fade margin calculus
- end the implementation of angular front-end part
- validate all calculus
- structure configuration file and call
- take into account penumbra model in code (optimization)
  - conf.ini management:
    - option to use a local propagator
    - unit test to do
  - fractions on power consumption
  - 3 modes on attitude control:
    - AEM file
    - Random rotation
    - emulation of ideal attitude control (celestlab)
  - structure of solar cell to be defined by user
  - service to get max eclipse duration
  - output du power budget
  - output DoD over time
  - get number of necessary accumulators (i.e. DoD along activity profile)
  - check if number of solar cell is consistent

# 5 Annex A: Input / Output format proposition

```
Notations:
(s): well defined and operationnaly used standard
(*): no standard defined yet, or not used in real life.
(^): single value, SI unit

========
= SCAO = IST module + interface (in dev)
========
```

```
IN
- OEM file (quaternions / time <CCSDS>) (s)
- MEM file (Activity profile, time dependant - CIC format) (s)
- MPM file (Mode definition, time independant - CIC format) (s)
- EDS files - (hw and sw specification) (*)

OUT
- AEM files (spacecraft, solar arrays, etc.)  (s)
- Attitude controle feasability (^)


=======
= LOS = - Stella (available) - no interface
=======
IN:
- Orbital parameters (*)
- Structure (*)
- Date (year) (^)
OUT:
- OK/KO (^)


===============
= Link-budget = - Dosa + interface (in dev)
===============
IN:
- Data flow (BER) (^)
- Orbital parameters (*)
- EDS files - (hw and sw specification) (*)

OUT:
- margins up/down (^)
- Raw Data Flow up/down (^)
- Usefull Data Flow up/down (^)


=============
= DataBudget = - Dosa + interface (in dev)
==============
IN:
- Usefull Data Flow (^)
- Visibility Window (*)
- Ground Stations List (*)
```

```
- On-board storage capacity (^)

OUT:
-available data flow (up) (^)
-available data flow (down) (^)


====================
= Mission Analysis - JSatorb + interface (in dev)
====================
IN:
- Orbital parameters (*)
- Ground Stations List (*)

OUT:
- OEM file (s)
- Visibility Window (*)
- Eclipses (*)

=============
= Structure = - IDM-CIC/Sketchup (available) + interface - INPB (in dev)
=============
IN:
- Number of Units (^)
OUT:
- 3D structural model (*)


===============
= Mass Budget = - IDM-CIC (available) + interface - INPB (in dev)
===============
IN:
- 3D structural model (*)
OUT:
- Mass Budget (^)


===============
= Activity profile - UC3M module + interface  (in dev)
===============
IN:
- user inputs (Payload details/cosntraints) (*)
```

OUT:
- MEM file (Activity profile, time dependant - CIC format) (s)
- MPM file (Mode definition, time independant - CIC format) (s)

==========
= Power Budget = Nanopower + interface  (in dev)
==========
IN:
- MEM file (Activity profile, time dependant - CIC format) (s)
- MPM file (Mode definition, time independant - CIC format) (s)
- EOM file
- AEM files (spacecraft + solar arrays)

OUT:
- Power Budget (^)
- Consumption profile for each equippment (*)
- Global consumption profile (*)
- Battery Level (*)

==========
= Disipation Budget = UM module + interface (in dev)
==========
IN:
- Node model (*)
- AEM file  (s)
- OEM file (s)
- Eclipses (*)
- EDS files - (hw and sw specification) (*)

USED INTERNAL DATA:
- Solar flux received (*)

OUT:
- Temperature profile of each node/equipment (*)
- Temp min/max (*)

==========
= Radiation Budget = UPM module + interface  (dev)
==========
IN:
- Orbital parameters (*)

```
OUT:
- Radiation received (*)


======================
= Visual tools / COM = - IDM view - VTS  (available) - interface (?)
======================
IN:
- AEM file (s)
- OEM file (s)
- 3D structural model (*)
OUT:
- Pretty views (^)


============
= Launcher = optional
============
IN:
- Mass Budget (^)
- Ppod type (^)
- Orbital parameters (*)
- Date (year) (^)

OUT:
- selected launcher (^)
- launcher constraints (*)
```

# References


[Braukhane and Bieler, 2014] Braukhane, A. and Bieler, T. (2014). The dark side of concurrent design: A story of improvisations, workarounds, nonsense and success. In *6th International Conference on Systems and Concurrent Engineering for Space Applications, Stuttgart, Germany*, pages 8–10.

[Di Domizio and Gaudenzi, 2008] Di Domizio, D. and Gaudenzi, P. (2008). A model for preliminary design procedures of satellite systems. *Concurrent Engineering*, 16(2):149–159.

[DLR, 2019] DLR (2019). Virtual satellite. *https://www.dlr.de/sc/en/desktopdefault.aspx/tabid-5135/8645_read-8374/*.

[ESA, 2019] ESA (2019). Ocdt. *https://ocdt.esa.int*.

[European Cooperation for Space Standardization, 2019] European Cooperation for Space Standardization (2019). Ecss-etm-10-25a. *https://ecss.nl/*.

[Le Gal and Lopes, 2016] Le Gal, J.-L. and Lopes, P. R. (2016). Idm-cic. *https://www.clever-age.com/fr/case-studies/cnes-une-application-de-modelisation-3d/*.

[Ningaraju, 2018] Ningaraju, P. (2018). Rf simulation and link analysis tool for csut. Technical report, ISAE-SUPAERO.

[Oberto et al., 2005] Oberto, R. E., Nilsen, E., Cohen, R., Wheeler, R., DeFlono, P., and Borden, C. (2005). The nasa exploration design team: Blueprint for a new design paradigm. In *Aerospace Conference, 2005 IEEE*, pages 4398–4405. IEEE.

[Perea, 2019] Perea, C. A. (2019). Generic cubesat power simulator. Technical report, ISAE-SUPAERO.

[Rheagroup, 2019] Rheagroup (2019). Cdp4. *https://www.rheagroup.com/fr/news/cdp4-open-source-community-edition*.

[Schreiber and Carley, 2005] Schreiber, C. and Carley, K. (2005). Ineffective organizational practices at nasa: A dynamic network analysis. *Available at SSRN 2726789*.

[Valispace, 2019] Valispace (2019). Valispace. *https://www.valispace.com/*.

[Vergnes, 2015] Vergnes, N. (2015). Bases de données graphes : comparaison de NEO4J et OrientDB. *Conservatoire National des Arts et Métiers https://www.irit.fr/ Thierry.Millan/MemoiresENG221/Nicolas_vergnes.pdf*.