Open-source Framework for the Concurrent Design of CubeSats

Thibault Gateau
ISAE-SUPAERO
Université de Toulouse
Toulouse, France
thibault.gateau@isae-supaero.fr

Lucien Senaneuch
ISAE-SUPAERO
Université de Toulouse
Toulouse, France
lucien.senaneuch@isae-supaero.fr

Sophia Salas Cordero
ISAE-SUPAERO
Université de Toulouse
Toulouse, France
sophia.salas@isae-supaero.fr

Rob Vingerhoeds
ISAE-SUPAERO
Université de Toulouse
Toulouse, France
rob.vingerhoeds@isae-supaero.fr

***Abstract*—In this paper, the open-source Nanospace framework is presented, dedicated to facilitate concurrent engineering during the preliminary design phase of CubeSats. It allows for direct information exchange between third-party expert software, while allowing transparent data visualization to any team member. It provides a web based GUI, and relies on a database with ACID properties and provides a RESTful API. Nanospace ensures concurrent access to the data, which is relevant when teams are working remotely. Nanospace also provides an intuitive way of visualizing other experts' contributions that can be of high value when looking for internal project understanding and transparency. The paper presents the modular software architecture of Nanospace, as well as design choices and the different options to connect to third-party specialized software and the database.**

***Index Terms*—open-source, Software, CubeSat, Software Architecture, Preliminary Design, Concurrent Design Engineering**

I. Introduction

The preliminary design of CubeSats requires close cooperation between experts from different disciplines for the analysis and modeling of the concerned subsystems. This modeling process of each subsystem may be influenced by design parameters of other subsystems, leading to strong inter-dependencies between subsystems. These inter-dependencies are present throughout the whole life cycle of the CubeSat. This situation explains why often concurrent engineering approaches are used for the initial stage of satellite design; CubeSats are no exception to this. Specific software allowing to structure this concurrent approach may help to support these preliminary design phases.

Within the current practice of CubeSat preliminary design, it seems that no suitable open-source software tools are available to support this design process; many experts use their own software tools and rely on manual information transfer. Functionalities of different software tools are often redundant, or even re-developed each time they are required, rather than re-used. Therefore there is a clear need for a framework to facilitate the management of information, models and data during the concurrent design process of CubeSats. Such framework may be able to increase the consistency throughout the information exchange, important to the preliminary design process of a CubeSat. Afterwards, it could be applied as well all along the life cycle of the project thanks to strong interconnections between domain-specific software.

Ideally, each discipline expert should be able to include necessary up-to-date inputs in their own software tools for running the required simulations or calculations in order to provide the expected outputs to the team. Monitoring this data flow would enable to be aware of real-time changes in a transparent way, which is essential for an effective teamwork; whether it is remotely, on site or a mixture of both.

In this paper, Nanospace (Fig. 1) is presented, a software framework dedicated to facilitate direct information exchange between third-party expert software, while allowing transparent data visualization to any team member. It provides a web based GUI, and relies on a database with ACID properties[1] and provides a RESTful API.

This paper is structured as follows: Section II of this paper presents an overview of the preliminary design of a CubeSat, which includes important data exchange models considerations. Then, in section III, the recommended standards are outlined and current open-source solutions for CubeSats preliminary design reviewed. The modular software architecture of Nanospace (Source code available at https://gitlab.isae-supaero.fr/nanostar/nanospace) is explained in the following section, along with the different options to connect third-party specialized software and the database, as well as the work environment. The achievements and future work recommendations are summarized in the conclusion section.

II. Preliminary Design for CubeSats

The preliminary design of CubeSats requires strong expertise across several fields. In this paper, preliminary design is identified as the 0/A phase according to ESA definition equivalent to Pre-Phase-A/Phase-A for NASA. Moreover, this work provides specific focuses on preliminary design for nanosatellites projects, although its use is wider. The scope of this paper covers specifically CubeSats [1], [2].

During the preliminary design, engineers usually rely on domain specific software for analysis, modeling, simulation, etc. It would be hard to conceive a complex project development

[1] Atomicity, Consistency, Isolation, Durability.

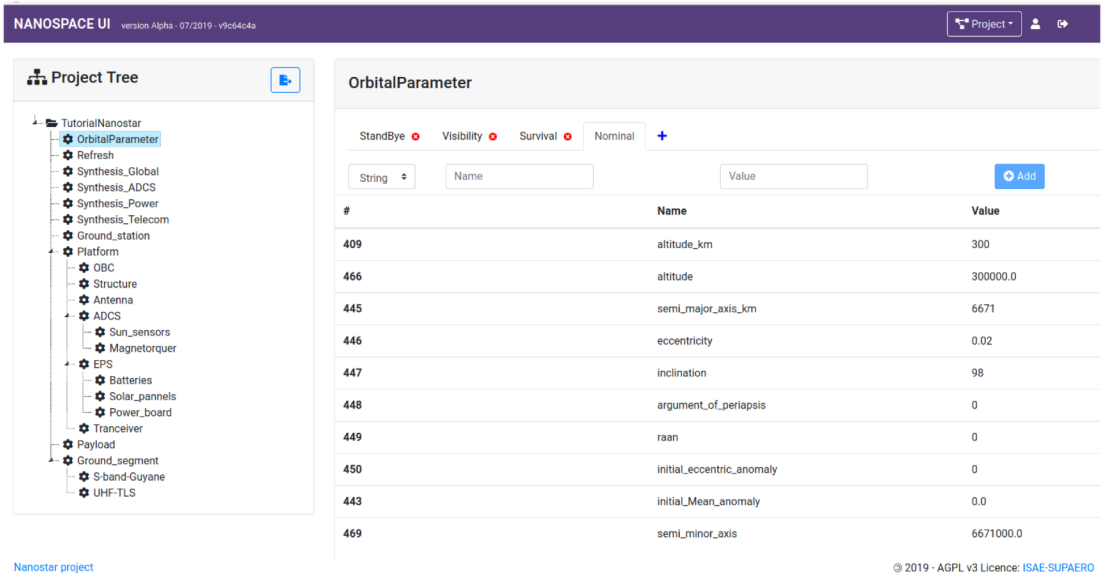

Fig. 1. Web Page visualization of Nanospace-UI: example of a "TutorialNanostar" project, focused on the "OrbitalParameter" component.

without the assistance of a comprehensive set of modeling, simulation or support software [3], [4]. The used software is not necessary identical for each subsystem, much rather each software, although sometimes overlapping, contributes in a complementary manner to the overall approach. Usually though they were not specifically designed to interact with each other. For example, some features may be redundant between software, and input/output data format may differ. Also, engineers and/or communities of practice can be conservative about the software they are already using. The success of a preliminary design requires data sharing and formalism as well as strong communication and cooperation. A software may not replace team management, however it can facilitate communication between team members: a software is able to support the implementation of data standardization and data exchange.

Modifying a single element in a specific subsystem may have significant consequences on the whole design. For example, if after a few iterations, engineers were to observe that the power is insufficient to provide the payload with the power it requires, it might be necessary to increase power supply by adding batteries, solar panels and wires for example. This at the same time might have an impact on the mechanical structure, mass budget, inertial center, thermal architecture, and possibly other elements as well. Many simulations in this case would have to be re-run and data would have to be re-exchanged. And yet again, the increased power consumption might also lead to additional power dissipation, which leads to additional radiators and therefore increased mass budget and so on.

"Experts" throughout this paper are seen as persons specialized in at least one satellite subsystem. The preliminary design process includes an iteration process between experts to converge towards a first valid design. This process requires experts to share and understand common data models. The inputs of some elements could correspond to the outputs of others, and vice-versa. In practice, solving interfaces issues is a complex process. For example, the power management engineer might need eclipse data for the design of the spacecraft in a specific format. The key issue is understanding who should be in charge of (re)computing the eclipse data. The power management engineer can compute it, using his/her own software, based on the data from the mission analysis. The mission analyst may provide the eclipse data, but not necessarily in a compatible format. Similarly, thermal engineers might need the data regarding the solar flux, in a specific format. A system engineer can also guarantee that a data file is consistent by checking that every subsystem team follows the appropriate standards. Nevertheless, having a full time system engineer is not always an option for a resource limited CubeSat projects, for example led by students. There is no one-way solution to sort the data format issue. Both modifying the input or output data format manually and switching to a different software or process to deal with the format issue will have a significant cost.

A. *Observations on CubeSat Preliminary Design*

Based on our past and on-going projects on CubeSats, we found that:

- Subsystem software inputs and outputs are strongly interdependent, during the whole CubeSat life cycle;
- Each expert prefers to use her/his own software. Sometimes, experts even prefer to re-develop an already existing software and have a tendency to prioritize in-house software;

- Many experts share common models, inputs and/or outputs. They are updating them, but not always tracking changes;
- Using existing standards is recommended - to the extent that standards have been defined, are relevant and convenient, and are implemented. Using standards greatly improves teamwork and project management;
- Lack of communication and misunderstanding between experts of different disciples should not be underestimated;
- Human resource limitation such as large turnover should be taken into account from the beginning of the project.

The possibility for effective data exchange between experts is a key requirement in concurrent engineering. The degree of integration of a software into a process depends on the way that the software interacts within the process and how the data is shared. If an expert has to manually re-write data into a software, it is assumed that there is "no automated interaction". Reciprocally, if each affected data is automatically updated when changes are made, it is assumed that there are "fully automated interactions". For example, if the mission analyst modifies the semi-major axis of the orbit, the link budget should have to be updated. It can be done manually, by the telecommunication expert who will re-run a script. However it is also possible to automatically trigger the same script when detecting the semi-major axis change. There are two options to connect specialized software together:

- to re-develop a dedicated specialized software entirely integrated with the concurrent engineering process, with high integration of the software but also high development costs. Software and experts would share an unified data model;
- to use specialized software "as it is" with low development costs and low level of integration of the software, and put the effort on the consistency of input/output data. That options could lead to the conversion of data format.

As often, a mixed solution can answer this problem. Existing specialized software could be modified, and data models could be adapted. A dedicated script can ensure the consistency of the common data.

B. Challenges

Theoretically a legitimate approach would be to rely on standards. If we suppose that all software are following standards such as the one proposed by the ECSS (European Cooperation for Space Standardization) standards [5], [6], the issue of the data sharing consistency should be solved. However, standards and models are still subject to interpretation when dealing with implementation [7], [8].

Even if subsystems experts are not necessarily using software with standardized inputs/outputs, system engineers are used to deal with issues of format consistency. Some reasons for not modifying the data format already in-use in specialized software include in-house culture, operational availability at the moment of implementation, or software version compatibility. Furthermore, different standards may exist for the same type of data.

To summarize, a successful project using concurrent engineering should take into account:

- consistency of shared data format;
- specialized software already in-use;
- transparency of data exchange: data format, data location, data transformation;

C. Approach

Concurrent Design Engineering (CDE) [9] is a methodology which inherently takes into account these success factors above (see section *III-B*). In this paper, concurrent engineering is seen in a spatial context [10], based on "work in parallel" and "co-located work". A bottom-up approach was chosen based on the needs of the subsystems engineers. A system engineer should be in charge of manually ensuring that the process is following the correct path, and that the data is consistent between subsystems and that up-to-date information is correctly shared and understandable between the different experts. The objective is not to suppress the role of the system engineer, but to facilitate the interactions and communications between experts, at a system level. The whole process should be as smooth as possible, to limit any additional work from the subsystem experts. Also, it was encountered that grating access to all the experts is a plus.

Generally, a team conducting a preliminary CubeSat design for a specific space mission should deal with the following elements:

- mass budget
- power budget
- link budget
- data budget
- dissipation budget
- radiation budget
- sometimes a propellant budget
- mechanical architecture
- thermal architecture
- an ADCS sizing
- an activity profile
- launcher restriction choice
- check for compliance with space laws
- check for payloads constraints

Therefore the goal of this paper is to propose a tool that facilitates the process of getting all these deliverable, for a team working remotely.

III. Standards and Software for preliminary design

In this section, a brief overview of classic data standards and existing open-source software that allow partly a CubeSat preliminary design is presented, along with related work on concurrent design engineering tools.

A. Existing Standards for Data Format

The Consultative Committee for Space Data Systems (CCSDS) provides recommendations related to data standardization. Many advantages of respecting such recommendations can be seen, such as interoperability, cross-support, reduction of risk, development time, and project costs. An exhaustive list of used standards can be found in [11]. In practice, the following elements were favored:

- CCSDS Orbit Ephemeris Message (OEM) files;
- CCSDS Attitude Data Messages (AEM) files;
- ECSS-E-TM-10-25A [5];
- STEP file for geometric data, although no real standard used;
- Electronic Data Sheet (EDS) for equipment.

B. Concurrent Design Engineering

Concurrent Design Engineering (CDE) applied to space mission preliminary design [9] is a methodology applied to facilitate the process of subsystem design parameters converging into preliminary models, and architectures. This methodology is usually supported by a Concurrent Design Facility (CDF) [12].

Many CDE implementations are proposed by space agencies – e.g. NASA Team-X [13], ESA OCDT, CNES IDM-CIC [14], and DLR Virtual satellite – by academics – e.g. Cedesk [15] C^{2}ERES DOCKS and FOrPlan [4] – or by private companies – e.g. Rheagroup CDP4, and Valispace. A review and comparison table between these tools has been proposed by Knoll and Golkar [16], also accessible online.

However, for three main reasons make that these software do not meet the requirements imposed to this project: license incompatibility, integration issues with third-party software and/or heavy spreadsheet focus.

In an academic context, there are several reasons to turn to open-source software, also within the framework of CubeSat projects [17]. Many open-source software, methodologies and recommendations can be easily found online, including for example the Libre Space Foundation initiative, full open-source CubeSat projects such as the UPSAT initiative, FloripaSat-I [18], and educational projects [19]. The "open-source Satellite" initiative provides a list of teams and software of the open-source ecosystem.

Few open-source solutions are providing a full set of compatible specialized software that are also open-source. CDF tools do not systematically incorporate discipline-specific software, which implies the need to use internal software that may be proprietary. In addition, it might be required to setup a network dedicated to the deployment of the CDF (e.g. CDP4). In a nutshell, all these CDE implementations on their own are not enough to achieve a full preliminary design. Third-party software are required.

Current usage of CDF tools shows that project managers pre-dominantly use spreadsheets with their pros and cons. Even though spreadsheets are powerful tools, they are not designed to replace databases. Usually spreadsheets neither support competitive data access, nor support data checking, data validation, data searching and data retrieval. Databases improve data integrity and data consistency. "Some things, which are difficult to do or error prone in spreadsheets, can be done easily and reliably in database" [20].

IV. Nanospace Architecture

This section describes the Nanospace software architecture and implementation choices. Nanospace was designed to facilitate academic CubeSats preliminary design and is made up of, mainly, a GUI, a database and an API. The specific requirements established are explained in subsection section *IV-A* . The implementation choices that were taken are presented in section *IV-B* and in section *IV-C* it is shown how third-party external domain specific software can be interfaced to Nanospace.

A. Nanospace high-level requirements

Nanospace shall:

- support Concurrent Design Engineering (see section *III-B*);
- support concurrent access to shared data models;
- allow remote located team-work;
- be modular and flexible to comply with experts needs;
- facilitate third-party software interactions;
- be open-source;
- be adapted to an academic usage (from the final user point of view):
  - easy to deploy or access (on a student laptop);
  - user friendly (intuitive and easy to handle for a student);
  - platform independent.

B. Implementation choices

To comply with the high-level requirements, it was concluded that Nanospace should:

- instantiate a centralized database;
- provide a REST API (REpresentational State Transfer API - Application Programming Interface);
- use Web service oriented software (when provided);
- follow standards (such as ECSS) insofar as possible.

Communications between third-party services and the application also needs to be taken into account to facilitate access to the database for any type of program and language (Python, C, C++, java, Go, Octave, Julia...). For sake of developers access, an interface with a high level of abstraction is necessary. For this purpose, it was chosen to be oriented as web applications, using HTTP requests and respecting REST convention.

Nanospace is therefore composed of three main components (Fig. 5):

- A User Interface - Nanospace-UI - in the frontend (see section *IV-B1*);
- A database - Nanospace-DB - in the backend (see section *IV-B2*);
- A REST API - provided by Nanospace-Service - also in the backend (see section *IV-B3*).

It should be noted that Nanospace requires additional components to be used. For example, a browser is required to interact with Nanospace (see section *IV-B4*).

1) Nanospace-UI:

Nanospace-UI (Fig. 1) is the main visual interface for any project member to interact, at any point in time, manually, with the model. Driven by the need for cross-platform access without any installations from client side (except a "modern" web browser), web technology was chosen. On Nanospace-UI, a user can:

- authenticate her/himself for accessing to the application and the projects he is responsible of;
- add other users as co-responsibles of a project;
- 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;
- make basic operation between referenced values ("like in a spreadsheet");
- create one or several modes for each components;
- access all the requirements;
- export or import the entire model on a system file in JSON (JavaScript Object Notation) format;
- be notified when a modifications is occurring on the data displayed by the UI;
- access an history of the past modifications;
- get notified of a required change whether the data are displayed or not.

The Project Tree of Nanospace is explicitly printed on the left side. A user can access a context menu to modify the model. By clicking on a component, the user can display specific values, and by clicking on a value the user toggle a menu to modify it.

2) Nanospace-DB:

Following usual approaches, data entities can follow a hierarchical tree data structure with variable depth. To increase performance when handling this type of structure, data is stored in graph form. Data must be write-protected in order to allow access competition. Neo4j complies with these requirements. In addition Neo4j has ACID properties. It is a NoSQL database storing graphs. Neo4j has also a larger community than other databases such as OrientDB [21];

Data in the database is stored in a graph form, defining the nodes of the graph as model entities (see Fig. 2). A data model consists of:

- a *project* storing elements that will constitute the model;
- *components* that are composing the *project*;
- a *mode* describing the different functional options of a *component* ;
- a *value* specifying *modes* (and specifying indirectly his *components*);
- a *user* who is responsible for the *project*.

Whereas in the current implementation only five entities have been taken, adaptation to a more complex structure so to fully comply with current standards [12], [22] is straightforward.

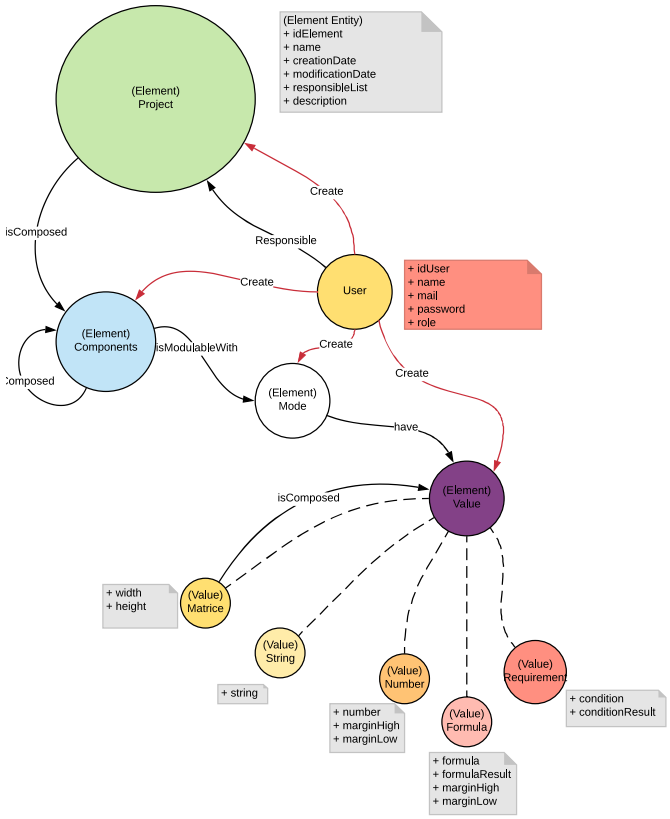

Fig. 2. Graph of the database model prototype. Only five entities are taken into account: *project*, *components*, *mode*, *value*, and *user*.

3) Nanospace-Service:

The Nanospace-Service allows applications (webpage GUI, scripts, etc.) to manipulate the model. Nanospace-Service implements a REST API, allowing specific domain applications and user interfaces to communicate with the database. Third-party programs can also access to the database and:

- can authenticate themselves;
- can create, can modify, or can delete elements of the model[2];
- cannot create requirements;
- cannot create a project.

4) Additional components: Some additional components are deployed:

- An HTTP server to serve the main graphical interface;
- A Broker message to deal with asynchronous events between each application;
- A database to store the *data events*, that correspond to dated events having circulated on the platform.

[2]Each modification of the database should be logged.

C. Three ways to include domain specific modules

Three ways are proposed to interact with *Nanospace-DB* through different levels of complexity, depending on the final user will and skill. The wish is that an engineer with no specific IT skills should be able to use it, whereas a subsystem expert with some skills in SW code development or even to a full stack web developer should be able to use more advanced features. In this way it is for example possible to directly add a parameter for a space mission or also to script an optimization procedure according to a specific database inputs, possibly publishing the results directly on the database. Finally, one can also retrieve database elements from a common HTML requests and taking advantage of the REST API to build an dedicated web application. Therefore, three main ways are available to interact with the project database (see Fig. 5):

1) Manually interacting with Nanospace-UI (see Fig. 1);
2) Embedded code to facilitate Nanospace-Service
   - through a generic Angular Component (see for example Fig. 3, code is available in the npm Registry)
   - through a simple generic Python API (see Fig. 4, code is available in Nanospace deposit).
3) Full compliance with Nanospace REST, API (usually web applications oriented)

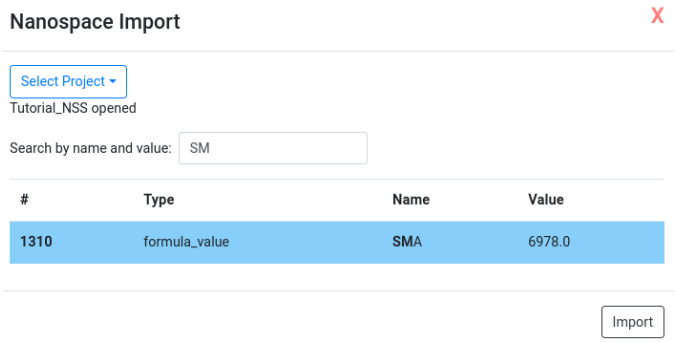

Fig. 3. Visual rendering of the angular nanospace selection menu. Some features are provided, such as researching an element name or value: in this example, the user was looking for the semi major axis (SMA) value in the database and is able to import it into its own web component.

```
from nanospace import Nanospace
user = ('userLogin','userPass')
srvAddr = 'https://SrvUrl/API'
nanospace = Nanospace(srvAddr, user[0], user[1])
id-alt-km = 40
id-SMA-km = 83
alti = nanospace.get-formula-value(id-alt-km)
print("altitude="+str(alti))
nanospace.update-formula-value(\
     id-SMA-km,6371+alti)
```

Fig. 4. Example of code in a Python API call.

V. Work Environment

This section describes the required environment to fully take advantage of the proposed solution.

A. Classic Software Support

It should be noted that concurrent engineering does not replace standard project management approaches and that Nanospace is therefore not intended to provide to the user with a "standard project management tool" with functionalities such as task management, code versioning, or team communication channels. As a consequence, the goal is not to have une full unique framework for preliminary design of CubeSats. Such functionalities need to be covered by other tools, such as the following that are commonly used in our design environment when working on preliminary design (and further):

- a standard project versioning (e.g a Git deposit);
- a chat (e.g RocketChat);
- a video conference service (e.g Jitsi);
- a project scheduler (e.g ProjectLibre);
- a dedicated framework for task management, bugs and issues tracking, often coupled with previous tools (e.g GitLab).

B. Nanostar Software Suite

The Nanostar Software Suite - NSS - is an example of a Nanospace constellation (Fig. 6). This NSS is currently under development. All modules are provided, developed and supported by different institutes of the Nanostar Consortium, and should respect as far as possible the CCSDS standards. Nanospace remains the backbone of the NSS, allowing a smooth interaction between each subsystem expert software.

VI. Discussion

Nanospace does not aim to implement a particular Model-Based Systems Engineering (MBSE) approach nor to replace a system engineer. It aims to facilitate the exchange of data between software. The role of the system engineers is to ensure the project consistency. They should propose an instantiation of Nanosapace according to their team software and requirements. However, the inclusion of the formal MBSE process before the instantiation of a Nanospace software constellation should be explored. open-source system engineering tools have been proposed [23], [24].

The Nanospace version presented in this paper is a prototype. The current tools offered by Nanospace are being used in different CubeSat development projects, and over the course of these projects, new needs and requirements emerge for the evolution of the software, some of which are currently being implemented. Currently designers are able to access and modify directly parameter values in the database. In spite of Nanospace-DB ACID property, consistency issue may appear during third party software computation process. To tackle this issue the implementation of a message broker, to automatically manage event messages, is planned.

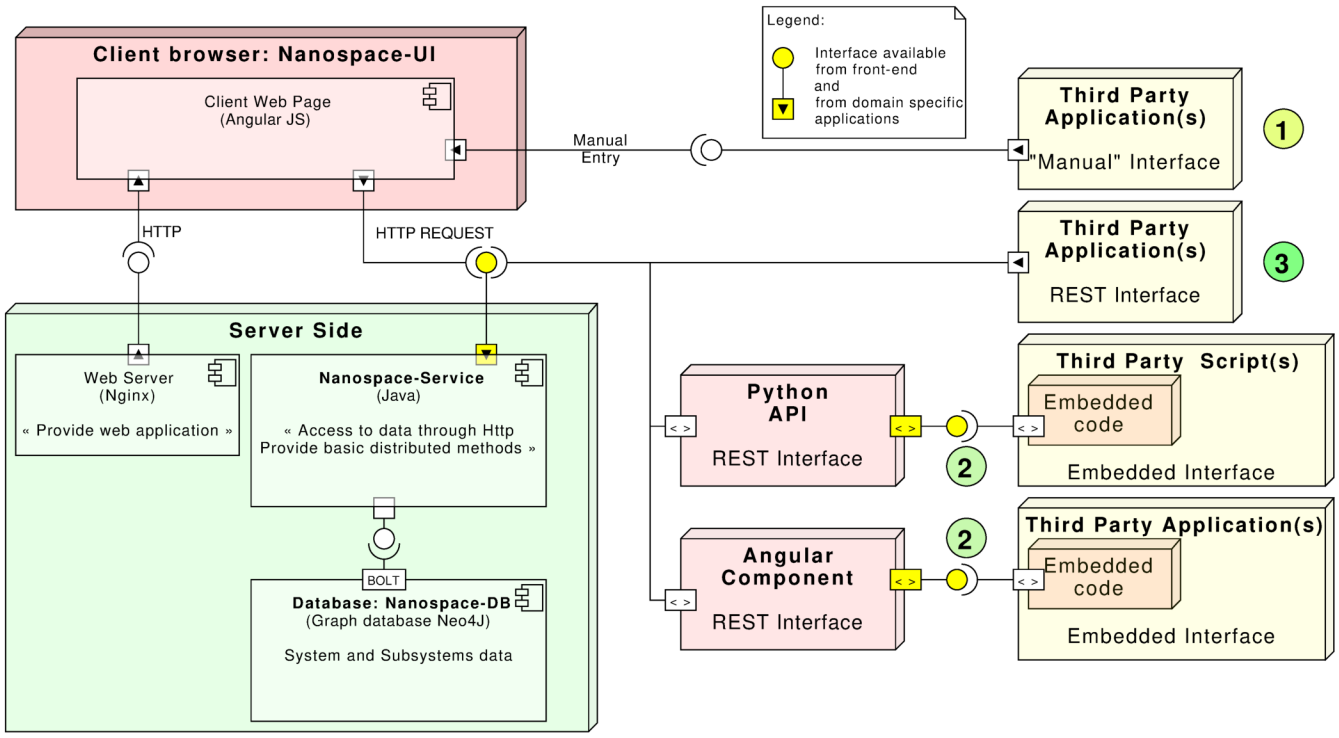

Fig. 5. Simplified Nanospace Architecture. The green circles represent the three ways available to connect third-party applications. (1) Manually interacting with Nanospace-UI. (2) Through a provided generic interface (Python API, or Angular Component). (3) Through the Nanospace REST interface.

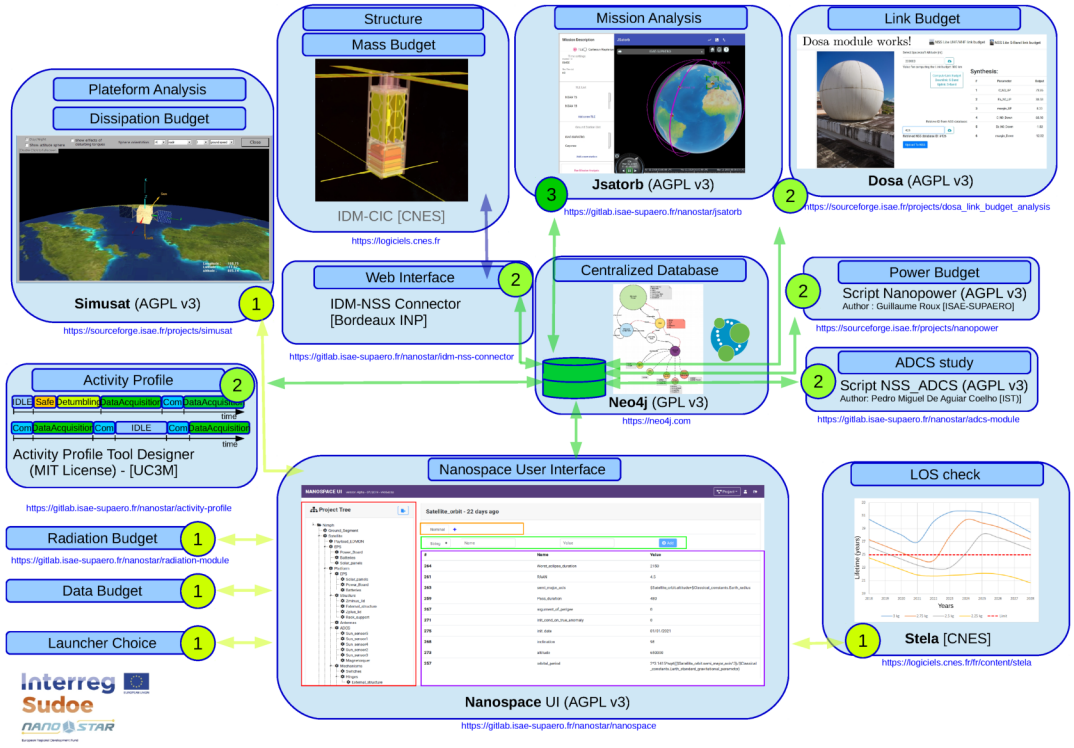

Fig. 6. A Nanospace Constellation for the Nanostar project: the Nanostar Software Suite (NSS). Green circles explicitly illustrate the degree of interconnection, from low connectivity (1) to a fully integrated process (3).

At the moment, the Nanospace-UI allows the user to export/load a project to/from a JSON (JavaScript Object Notation) file. Compatibility with JSON-LD (JSON for Linked Data) would be a plus, to follow standardisation initiatives such as MetaSat [22].

Additional software could be added to a Nanospace constellation. For example, optimization tools may have a direct interest on relying on the database for their different iterations processes: integration of OpenMDAO [25] to a Nanospace instance is currently under study.

VII. Conclusion

Nanospace is an open-source software dedicated to facilitate the preliminary design of CubeSats for remote teamwork. It includes three main components: a web based GUI - Nanospace-UI -, a database with ACID properties - Nanospace-DB -, and a RESTful API through Nanospace-Service.

Nanospace can add value to the preliminary design of a CubeSat project, when integrated into a consistent software package, such as the Nanospace Constellation. In such a constellation, a set of selected third-party software is able to interact with common centralized models provided by the database. Therefore, Nanospace should be easy to integrate to third-party application. Integration can be implemented (or not) at different levels:

1) manual data value filling in Nanospace-UI (totally independent from the application),
2) intermediate interfaces easy to integrate in one's code (Python API or Angular Component examples are provided), and
3) direct HTTP requests based on the provided REST service.

Nanospace ensures concurrent access to the models, which is relevant when teams are working remotely. Nanospace also provides an intuitive way of visualizing other experts' contributions that can be of high value when looking for internal project understanding and transparency. Nanospace is open-source (AGPL v3 license). Source code is available here. Nanospace is also currently running on a test server. An example of a Nanospace constellation has been used in the context of the Nanostar project, the NSS constellation. Which is available here.

References

[1] H. Heidt, J. Puig-Suari, A. Moore, S. Nakasuka, and R. Twiggs, "Cubesat: A new generation of picosatellite for education and industry low-cost space experimentation," *Proceedings of the Small Satellite Conference*, 2000.

[2] C. Cappelletti and D. Robson, "Cubesat missions and applications," in *Cubesat Handbook*. Elsevier, 2021, pp. 53–65.

[3] M. Deshmukh, V. Schaus, P. M. Fischer, D. Quantius, V. Maiwald, and A. Gerndt, "Decision support tool for concurrent engineering in space mission design," in *Concurrent Engineering Approaches for Sustainable Product Development in a Multi-Disciplinary Environment*. Springer, 2013, pp. 497–508.

[4] R. A. Chagas, F. L. de Sousa, A. C. Louro, and W. G. dos Santos, "Modeling and design of a multidisciplinary simulator of the concept of operations for space mission pre-phase a studies," *Concurrent Engineering*, vol. 27, no. 1, pp. 28–39, 2019. [Online]. Available: https://doi.org/10.1177/1063293X18804006

[5] European Cooperation for Space Standardization, "Ecss-etm-10-25a," *https://ecss.nl/*, 2019. [Online]. Available: https://ecss.nl/

[6] M. Jones, E. Gomez, A. Mantineo, and U. Mortensen, "Introducing ecss software-engineering standards within esa," *ESA bulletin*, pp. 132–139, 2002.

[7] R. Arias, F. Kucinskis, and J. D. Alonso, "Lessons learned from an onboard ecss pus object-oriented implementation," in *SpaceOps 2008 Conference*, 2008, p. 3524.

[8] N. Humeau, T. Gateau, G. Crooks, and F. Anne, "A lightweight and efficient control center based on modern technologies," in *2018 SpaceOps Conference*, 2018, p. 2634.

[9] M. Bandecchi, B. Melton, and F. Ongaro, "Concurrent engineering applied to space mission assessment and design," *ESA bulletin*, vol. 99, no. Journal Article, 1999.

[10] D. Knoll, C. Fortin, and A. Golkar, "Review of concurrent engineering design practice in the space sector: state of the art and future perspectives," in *2018 IEEE International Systems Engineering Symposium (ISSE)*. IEEE, 2018, pp. 1–6.

[11] A. Scholz, "Cubesat standards handbook: A survey of international space standards with application for cubesat missions," *https://gitlab.com/artur-scholz/book-cubesat-standards*, 2019.

[12] D. Di Domizio and P. Gaudenzi, "A model for preliminary design procedures of satellite systems," *Concurrent Engineering*, vol. 16, no. 2, pp. 149–159, 2008.

[13] R. E. Oberto, E. Nilsen, R. Cohen, R. Wheeler, P. DeFlono, and C. Borden, "The nasa exploration design team: Blueprint for a new design paradigm," in *Aerospace Conference, 2005 IEEE*. IEEE, 2005, pp. 4398–4405.

[14] J.-L. Le Gal and P. R. Lopes, "Idm-cic," *https://www.clever-age.com/fr/case-studies/cnes-une-application-de-modelisation-3d/*, 01 2016. [Online]. Available: http://igosat.in2p3.fr/wp-content/uploads/2016/06/13_2016_06-Les%20outils%20CIC%20du%20CNES-JLLeGal%20-%20PLopes.pdf

[15] Skoltech, "CEDESK," https://cedesk.github.io/, 2019, [Online] Accessed: 2021-02-25.

[16] D. Knoll and A. Golkar, "A coordination method for concurrent design and a collaboration tool for parametric system models," *Concurrent Engineering*, vol. 26, no. 1, pp. 5–21, 2018.

[17] A. Scholz and J.-N. Juang, "Toward open source cubesat design," *Acta astronautica*, vol. 115, pp. 384–392, 2015.

[18] M. G. Mariano, F. E. Morsch, M. S. Vega, S. L. Oriel, S. L. Kessler, B. E. Augusto *et al.*, "Qualification and validation test methodology of the open-source cubesat floripasat-i," *Journal of Systems Engineering and Electronics*, vol. 31, no. 6, pp. 1230–1244, 2020.

[19] D. Geeroms, S. Bertho, M. De Roeve, R. Lempens, M. Ordies, and J. Prooth, "Ardusat, an arduino-based cubesat providing students with the opportunity to create their own satellite experiment and collect real-world space data," in *22nd ESA Symposium on European Rocket and Balloon Programmes and Related Research*, vol. 730. Citeseer, 2015, p. 643.

[20] K. J. Gordon, "Spreadsheet or database: Which makes more sense?" *Journal of Computing in Higher Education*, vol. 10, no. 2, pp. 111–116, 1999.

[21] N. Vergnes, "Bases de donnees graphes : comparaison de NEO4J et OrientDB," *Conservatoire National des Arts et Metiers*, 2015. [Online]. Available: https://www.irit.fr/~Thierry.Millan/MemoiresENG221/Nicolas_vergnes.pdf

[22] Bouquin, Daina and Papadeas, Pierros and Chivvis, Daniel and Williams, Allie and Tsiligiannis, Vasilis and Damkalis, Fredy and Frey, Katie, "Describing smallsat missions with metasat," *Proceedings of the Small Satellite Conference*, 2020.

[23] N. JPL, "Open model based engineering environment," http://www.openmbee.org/., 2019, [Online] Accessed: 2021-02-25.

[24] T. Kulkarni, K. DeBruin, A. Nelessen, K. A. Reilley, R. Peak, S. J. Edwards, and D. N. Mavris, "A model based systems engineering approach towards developing a rapid analysis and trades environment," *AIAA SPACE 2016*, p. 5472, 2016.

[25] J. S. Gray, J. T. Hwang, J. R. R. A. Martins, K. T. Moore, and B. A. Naylor, "OpenMDAO: An open-source framework for multidisciplinary design, analysis, and optimization," *Structural and Multidisciplinary Optimization*, vol. 59, no. 4, pp. 1075–1104, April 2019.