18th International Conference on Space Operations, Montreal, Canada, 26 - 30 May 2025.

SpaceOps-2025, ID #415

Development and operations of two mission centers built on top of a CNES generic tool

Maxime Chambon[a]

[a] *NεSS MC and YODA CGCU Technical Manager, CNES DTN/TSA/LMC, France, Maxime.Chambon@cnes.fr*

Abstract

NεSS and YODA are two nanosatellites demonstrators' projects in which CNES, the French space agency, is involved, sharing the same objective: performing experiments in orbit at lower cost than traditional missions.

On the ground segment side, both NεSS MC (Mission Center) and YODA CGCU (Payload Management Center) were built on top of the same layer: SPIRIT. This tool is a generic integration platform developed by the CNES, with the purpose of easing and standardizing the implementation of mission and expertise centers.

This paper will detail how SPIRIT is able to fulfill NεSS MC and YODA CGCU requirements, bringing its own set of prerequisites. Based on these two concrete use cases, the development of a mission center on top of this technology will be described. SPIRIT's main benefits will also be highlighted: its genericity, and its replicability.

The paper will macroscopically present SPIRIT's intern machinery, based on micro-services technologies running into containers, for scalability and resilience purposes. It will then focus more extensively on its capabilities.

In addition, this generic tool can run with shared computation resources, which can help reducing both the overall cost and environmental impact of mission centers. This topic will also be covered, providing metrics on the estimated reduction.

After a year operating NεSS, this paper will detail how SPIRIT assists the MC operators' team in carrying out their duty. Their weekly routine consists in preparing MOFs (Mission Operation Files), in which are sequenced the satellite's daily tasks, twice a week, and processing the downlinked data.
Dedicated algorithms have been developed by CNES experts teams and integrated into SPIRIT as COTS (Commercial Off-The-Shelf) to handle these activities that are not part of its native scope. The main benefits being that these programs could evolved independently from the MC throughout the project lifespan.

As time passes, SPIRIT is getting involved in an increasing number of space programs. Seven at the time of writing, including two in operation. To fulfill projects' various needs, the generic tool's managers work closely with mission centers' teams, in an agile approach. That way, software evolutions can be efficiently prioritized and implemented according to the projects requirements.
The paper will conclude with concrete challenges that SPIRIT will need to face in a close future, and open on advanced evolutions that could be integrated in a further roadmap to improve its functionalities.

Acronyms/Abbreviations

CNES	=	French National Space Agency	CGCU	=	Payload Management Center
MC	=	Mission Center	CCC	=	Command Control Center
COTS	=	Commercial Off-The-Shelf	MOF	=	Mission Operational File
PLTM	=	PayLoad TeleMetry	TC	=	TeleCommand
HKTM	=	HouseKeeping TeleMetry	HMI	=	Human Machine Interface
CPU	=	Central Processing Unit	HPCC	=	High Performance Computing Cluster
RAM	=	Random-Access Memory	VM	=	Virtual Machine
TAI	=	International Atomic Time	OS	=	Operating System
IPA	=	Identity Policy Audit (authentication and authorization policies)	SWOT	=	Satellite launched in 2022 to study: Surface Water and Ocean Topography
SDK	=	SPIRIT Development Kit	SVOM	=	Space Variable Objects Monitor Satellite, launched in 2024 to study gamma-ray bursts
GPFS	=	General Parallel Files System	TRISHNA	=	Satellite to analyze the earth surface in solar and infrared domains

1. Introduction

A traditional CNES space mission embeds three main components, illustrated in Fig. 1:

1. A **space segment**, gathering one or several satellites equipped with payloads onboard. They receive instructions through TCs (TeleCommand) to fulfill their mission, and communicate their achievement afterwards.
2. An **antennas network**, operating as a relay between the space and the ground segment.
3. A **ground segment** split between a CCC (Command Control Center), and a MC (Mission Center).

The main objective of a CCC is to maintain the satellites in nominal state so that they can conduct their mission. Therefore, they analyze the space segment health-state, provided by the HKTM (HouseKeeping TeleMetry), and compute its attitude and orbit. If any anomaly is raised, experts are mandated to investigate on it. A CCC's team also convert inputs received from the mission center into TCs, to communicate it to the satellites.

A MC generates programming plans, detailing instructions for the space segment, based on orbital files provided by the CCC. In addition, it processes PLTM (PayLoad TeleMetry), transmitted by Antennas, with the help of experts' algorithms, to build high-level products. Mission center's team can then distribute these products to external clients.

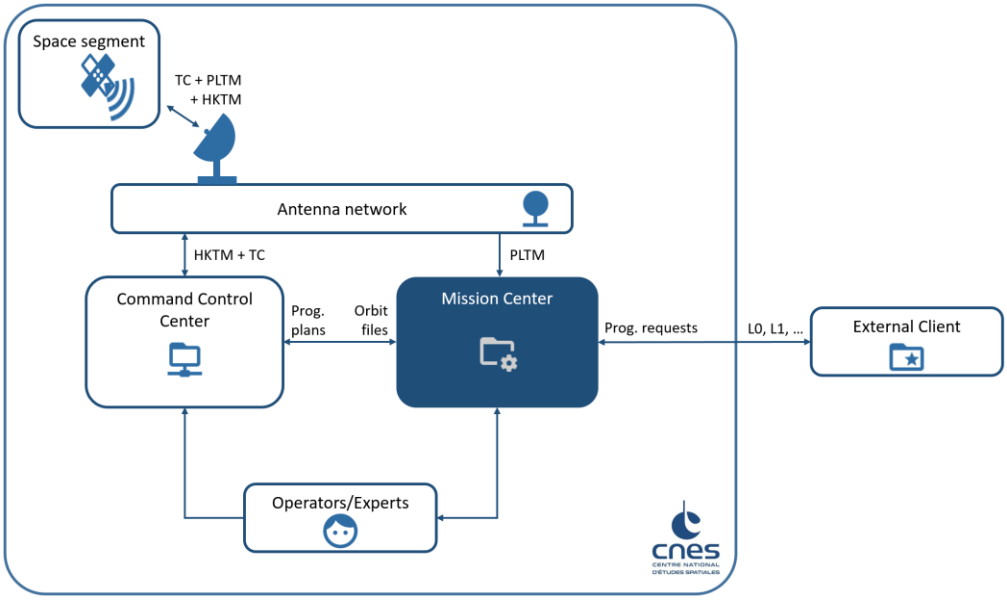

Fig. 1. Satellite ground segment functional architecture

From a ground-segment point of view, space missions can have similar requirements. For instance, a mission center should be capable of acquiring, processing, and distributing data. Moreover, it should be possible to trigger these events automatically, to prevent an operator from monitoring the system 24 hours a day.
Therefore, CNES aims to standardize the implementation of both its CCC and MC components through the development of generic products. On the mission center's side, the SPIRIT project was initiated to fulfill this ambition.

2. SPIRIT : a generic orchestrator

SPIRIT technical architecture is schematized in Fig. 2. This paper won't extensively detail its backend machinery, as the focus has been placed on its capabilities. Nevertheless, to fulfill modularity, flexibility, and simplicity requirements, SPIRIT's architecture is based on micro-services. Each service is treated as an independent entity, ensuring that their environment are isolated from the application's other components. This architecture also offers the application an horizontal scalability. Indeed, each container, hidden behind the services, can be replicated to split the workload and improve the overall resilience. If a container faces an error, its replicas will handle the requests. In addition, all the containers got their own resources being allocated, and are therefore independent. The orchestrator's services interact with a PostgreSQL database, in which are referenced every data accessible through its HMI (Human Machine Interface). SPIRIT was designed as a web application; hence, the communications between services is based on HTTPS protocol.

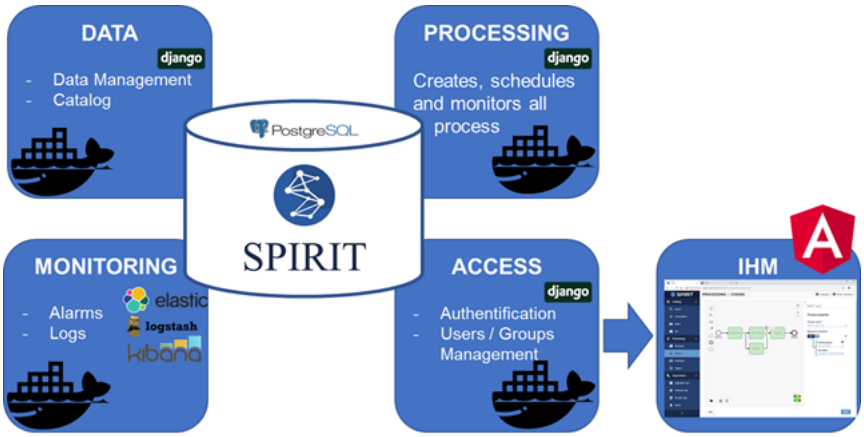

Fig. 2. SPIRIT technical architecture

Five services families can be distinguished within SPIRIT :

- **Data**: covers the data storage, indexing and management. Data models, defined by users are stored in this service, as well as scripts called "importers", and "serializers" describing how to fill these models, from a given input. All these components are written in Python. In addition, any file imported, or exported from the application, is supervised by this service. On the process side, it ensures that their inputs are positioned in the adequate folder.

- **Processing**: handles processes creation, edition, and execution. Though SPIRIT provides a set of generic processes natively available, each mission can develop and integrate its own set of algorithms to fulfill their requirements. A process is defined by two distinct elements: a shell script and a Python file. The former is used by the application to call the latter, where the main part of the logic is stored. Chains can be created by connecting several processes together.
  Users also have the possibility to define "triggers", to automate processes execution. They are started off upon specific events, such as the detection of a new input in the database, or the creation of a new file in a specific directory, for instance. These "triggers" are written as text files, and can be created within SPIRIT HMI.

- **Monitoring**: centralizes information linked to SPIRIT operation. It covers the application native components as well as missions' specific ones. This service is based on a combination of "Elasticsearch", "Logstash", and "Kibana" solutions, granting users an easy access to logs, with powerful search and display engines. In addition, an alarm mechanism, designed to alert users, is available. They can be raised within any process, upon facing a specific event.

- **Access**: manages the application authentication, and grants access to SPIRIT's functionalities, depending on users restrictions. Profiles can be set-up in this service to define precisely which process, and data can be accessed by the registered users, and how they can manipulate with them.

- **HMI** : handles the web application graphical interface, built on top of Angular. A list of sub-menus are available on the left of the page to redirect users into SPIRIT's services functionalities. In addition, a graphical tool, based on "bpmn.io" solution, is available to easily design processing chain, by connecting "blocs" representing unitary processes, as shown in Fig. 3.

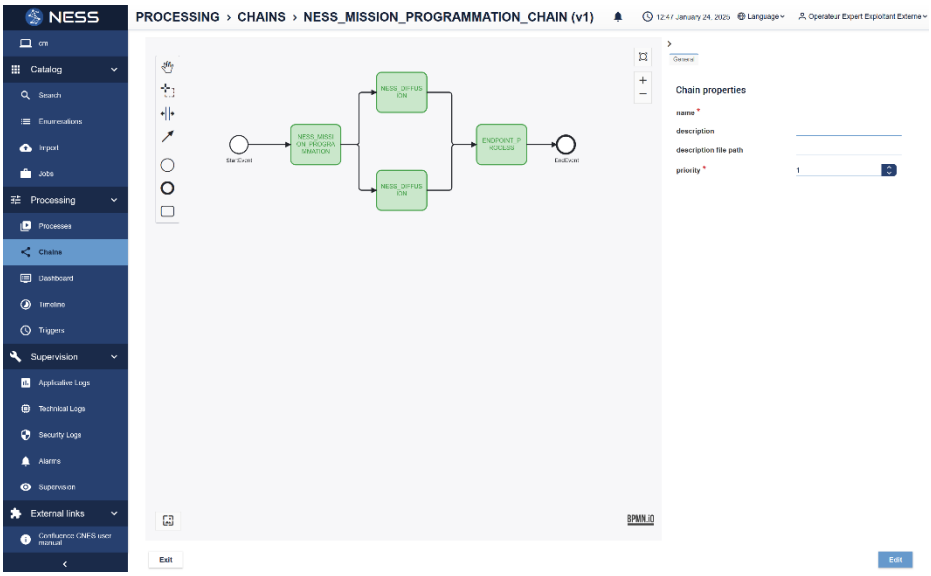

Fig. 3. SPIRIT HMI with chain design tool

3. SPIRIT computing performance optimization

To reduce an overall project cost and environmental impact, SPIRIT can be paired with a HPCC **(**High Performance Computing Cluster). This way, resource-intensive treatments can be run 'remotely', on shared servers, rather than on the orchestrator's ones. This strategy is particularly profitable for missions with high and short consumption peaks, as they can minimize their own infrastructure's dimension, and therefore reduce their cost.

In addition, mutualizing resources has three main benefits. First, it eases the system's maintenance by grouping the knowledge, decisions, and technical interventions within a single team, thus reducing the global human cost. Then, it tends to optimize their computation time by multiplying the demand. Finally, it prevents the proliferation of underused dedicated computing infrastructures. Knowing that about 50% of a server's carbon footprint comes from its manufacturing [1] ; this last point can be viewed as a significant gain from an ecological point of view.

Note that this architecture is not recommended for projects with near-real-time processing requirements, as the shared resources might be occupied by other missions when needed.

4. Development of a SPIRIT based mission center

SPIRIT, as most applications that need to be installed on a project dedicated environment, brings a set of prerequisites to build a fully functional mission center.

First, on the infrastructure's side, as illustrated by Fig. 4, four components are mandatory to deploy an operational SPIRIT instance:

- A pool of minimum **four VMs** (Virtual Machine), or servers to host the application's services.
- A PostgreSQL **database**, to catalog files and metadata.
- A **shared storage space**, mounted on the VMs, to save files received and produced by the MC.
- An **IPA** (Identity Policy Audit) **server**, to handle users' authentication and permissions.

The VMs/servers mentioned beforehand, should meet the minimal specifications detailed in Table 1.

Table 1. SPIRIT VMs requirements

OS	Distribution	CPU	RAM	Disk
Linux	Red Hat 8	4	8 GB	100 GBS

Three additional features are usually added to the base system to fulfill space missions' requirements. They are not mandatory to operate SPIRIT, but can be easily plugged to extend its functionalities:

- A connection to **TAI** (International Atomic Time) **servers**, distributing a common time reference between the ground-segment components.
- An **exchange server**, used as an interface to share files with external entities.
- A connection to a **HPCC**, to compute efficiently high consumption processes, without affecting the application's capabilities.

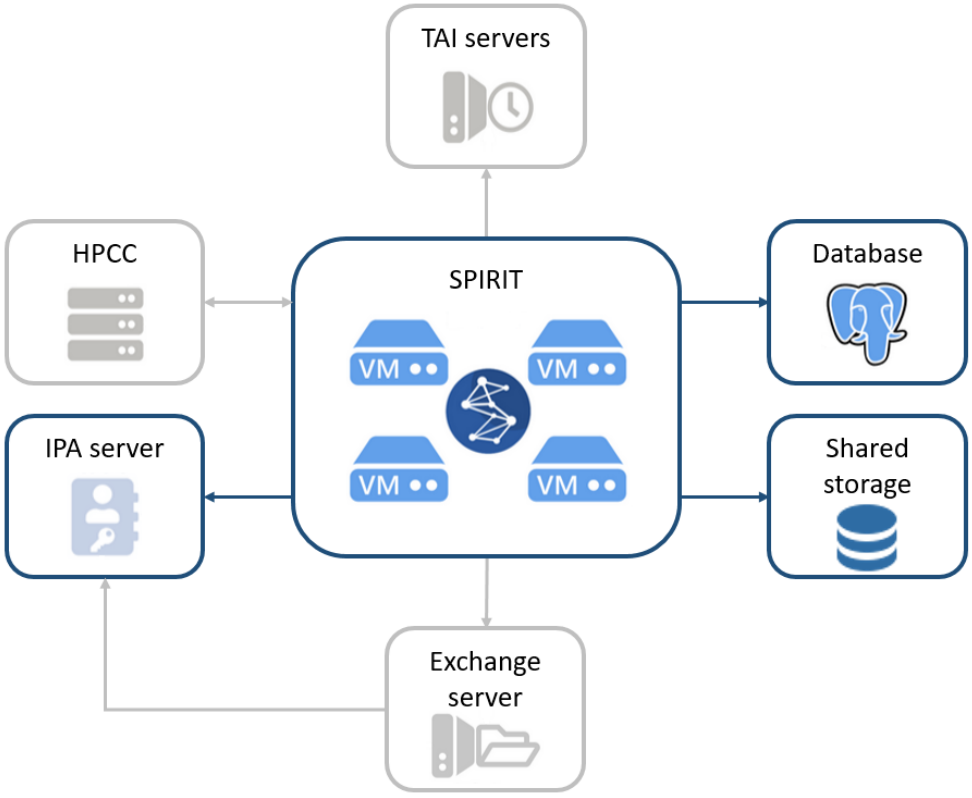

Fig. 4. SPIRIT basic infrastructure

Once the infrastructure is configured, few dependencies must be installed on the servers, to be able to deploy SPIRIT:

- **Docker**, version ≥ 24.0.
- **Docker-compose**, version ≥ 2.12.
- A **docker registry**, to store the application's images.
- The **SDK** (SPIRIT Development Kit) of the targeted version.

The SDK is the cornerstone of a mission center instantiation. It contains:

- A generic part, embedding SPIRIT native programming classes, as well as default processes, which can be taken as examples.
- A part "to missionize", including a set of empty classes, derived from the native ones, to help integrators in their duty. It embeds processes and metadata templates, which need to be customized to fit the project requirements. It also contains a folder in which should be stored the mission's YAML configuration file.
- Pre-built Docker images for each of the orchestrator's modules, except "Data".
- A set of scripts, libraries, and Ansible playbooks, to assist integrators in "missionizing" and deploying SPIRIT.

As mentioned above, "missionizing" an SDK affects two SPIRIT services: "Processing", and "Data".
On the former's side, missions should develop their own Python scripts, compatible with the orchestrator's native functions, especially regarding inputs and outputs management. These algorithms must be integrated into the application as processes, through the creation of a JSON description file, and a Shell executable. Trigger text files can then be added to detail the conditions under which a process should be automatically launched.

On the "Data" side, each element, received or produced by the mission center, which should be consultable on SPIRIT's HMI, must be described in an unitary Python file called a "model". It details all the metadata linked to a given record type, specifying their nature. As mentioned in section §2, each model must be associated with an "importer" depicting how metadata should be extracted from a given input, and a "serializer" that can be used to update their format right before importing them in the database. All these customized files are embedded in the "Data" Docker image when an integrator builds it, through the execution of the adequate SDK's script.

Once a "missionized" SDK has been placed on the target environment, and the Docker images are loaded, SPIRIT's native scripts can be run to deploy, and then start the application.

A SPIRIT instance can easily be duplicated on another configured pool of VMs, by copying a "missionized" SDK on it, and running its deployment and starting scripts. The new replica can be connected to the same database than the original instance, or a distinct one, depending on its configuration file. In addition, records and processes can easily be transferred between the two environments using the orchestrator "import/export" native functionalities.

5. SPIRIT benefits based on NεSS operations analysis and YODA CGCU development

As schematized in Fig. 5, NεSS mission center fulfills four main duties.

First, it generates the satellite programming plans, and diffuse them to the CCC. They are computed thanks to an algorithm, integrated into the MC, which is developed and maintained by an expert. Every acquisition to be recorded by the satellite are detailed in these plans called MOFs as telecommands, and are stored in SPIRIT's database. Listing the acquisitions saved in the orchestrator's catalog allows an operator, as well as the algorithm to retrieve the satellite's programming state at a given date.

Once the measures have been taken, and downlinked through a multi-mission antenna network, they are uploaded on a dedicated SFTP server. The resulting PLTM files are fetched by the mission center and stored in its database.

Then, based on these data measured by the satellite, it elaborates advanced products with the help of several processing layers. The first step, the decommutation, consists in rearranging the information contained in the PLTM files, as it was structured onboard to ease the downlink. Afterwards, the L0 process associates each decommuted telemetry to the acquisition record it originated from, stored in SPIRIT's catalog. Finally, algorithms developed by payload experts and integrated into the MC are executed to generate higher-level products.

Lastly, the results produced are forwarded to our external partner, by uploading them on the previously mentioned SFTP server.

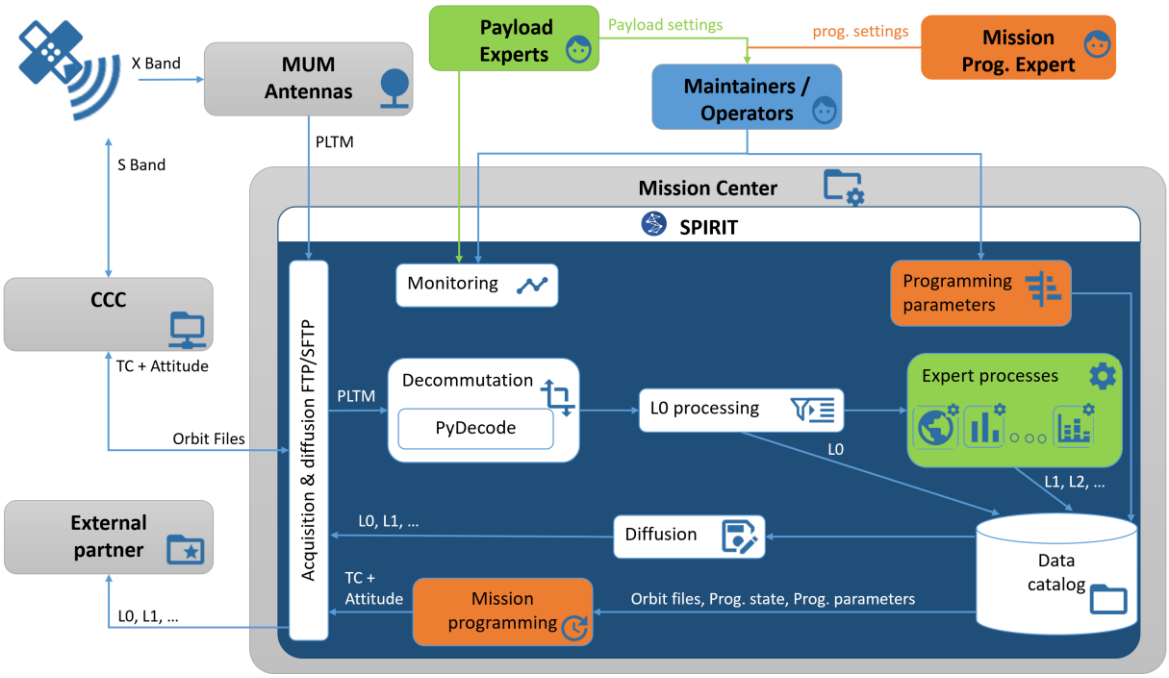

Fig. 5. NεSS MC functional architecture

5.1 SPIRIT monitoring benefits

Most of the processes integrated in the NεSS mission center are triggered automatically by SPIRIT. It includes "mission-programming", which is the only one considered critical, as the mission could be postponed if it fails. For that matter, and to prevent requiring an operator to monitor the system 24 hours a day, it was stated that the MOFs computation would be scheduled twice a week, during working hours. In addition, the orchestrator's alarm mechanism mentioned in section §2 has been plugged to this task, to get both a visual and a mail alert if an error occurs while it's running.

Consequently, supervising the MC on a weekly routine is not restraining. It essentially consists in ensuring that the following automated processes succeed:

- The ingestion of orbital files uploaded by the CCC on the SFTP server, with a recurrence of at least once a day.
- Fetching the experts payload settings, and specific acquisition requests. These files are provided before the programming plan generation, to constrain the algorithm.
- MOFs computation and diffusion to the command control center.
- Reception of the downlinked PLTM, and its decommutation.
- Expert algorithms execution and high-level products creation.
- Distribution of the generated products to external partners.

An operator can easily identify tasks success or failure, thanks to a basic color code in SPIRIT main dashboard view. As illustrated in Fig. 6, once completed, each execution is either displayed as a green line if no error occurred, or as a red one otherwise.

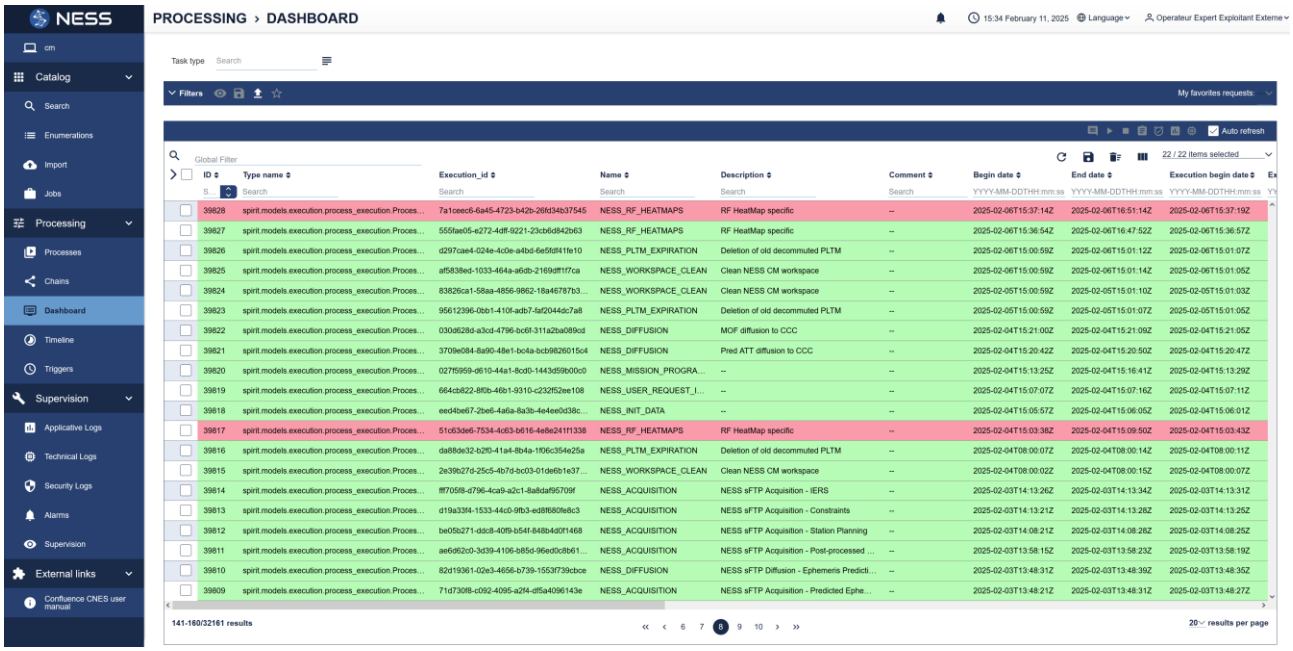

Fig. 6. SPIRIT main dashboard view missionized for NεSS

5.2 SPIRIT processing modularity

In the first months operating NεSS, many issues were faced linked to the execution of the mission-programming algorithm. For the most part, it was a consequence of corner cases that could hardly be tested before the satellite launch. Thankfully, though the CNES mission center's team was small, the members were reactive, and well implicated on the project. Therefore, iterative patches of the algorithm could be developed and integrated in the orchestrator on a day scale, once or twice a week in the early commissioning phase. It was partly made possible thanks to its COTS status regarding SPIRIT. Indeed, during the mission-programming process integration, the Python script called by the orchestrator has been designed to perform the input/output management, and call an executable as a sub-process. This way, a new algorithm patch, with no impact on its interfaces, could easily replace its older compiled version.

The same approach has been applied to payload experts' processes and provided another benefit. As a matter of fact, the data multiplication in the catalog highlighted new requirements, and experts developed additional algorithms, to be integrated in the MC, to fulfill them. Fortunately, the existing orchestrator's Python scripts, linked to experts' high-level processes, could be used as templates, to easily replicate them into new occurrences.

5.3 SPIRIT computing optimization

As mentioned in section §3, SPIRIT can be connected to a high performance computing cluster to optimize its own resources, and reduce a project's infrastructure investment.

On NεSS mission, four consumption peaks per day were identified. Each peak being characterized by the reception of 25 products on average to be treated. A single product requires 2 CPU (Central Processing Unit), 25GB of RAM (Random-Access Memory), and a computation time of approximately 10 min.

Two strategies were studied to manage this workload. The first one consisted in processing the incoming data one after another. In this scenario, each peak required over 4 hours of processing time to be handled. It was judged too slow regarding experts needs to analyze the downlinked acquisitions.

The second one, which got adopted, was based on computing products in parallel. Based on NεSS mission specifications, this scenario required a platform with a minimum of 50 CPUs and 650 GB of RAM, which would have been used at its full capacity for only 3% of its uptime. As no real-time processing was required, it was decided to pair NεSS MC with the CNES HPCC, offering a total of 1024 CPUs and 8 TB of RAM of mutualized resources. Therefore, the mission center's operational infrastructure could be limited to SPIRIT's base prerequisite: 16 CPUs and 32 GB of RAM.

On YODA's side, at the time of writing, high-level processes resources consumption is not well characterized. However, it has already been identified that there will be no real-time processing constraints. In addition, NεSS mission provided a positive feedback on the interconnection between SPIRIT and the CNES HPCC. Consequently, the same architecture has been applied to the CGCU.

5.4 SPIRIT replicability

On NεSS and YODA missions, SPIRIT is installed on three distinct platforms that were deployed to meet respectively their MC and CGCU's needs. These environments are similar in terms of architecture. They embed four VMs following the orchestrator's requirements, as stated in section §4, however, each one of them have a specific purpose:

- The **development** platform is used to missionize SPIRIT SDK, develop new functionalities, and perform basic tests. No crucial data are stored in it, as it is intended to be emptied and restarted on a regular basis.
- The **qualification** environment main purpose is to be a replica of the operational one. Large scale tests are performed on it to investigate on issues identified in production, or to validate new mission center's versions.
- The **production** platform receives and process operational data, to generate high level products.

The same SDK is shared between the environments mentioned above. Therefore, when a new MC or CGCU version is released, the integrator generates an up to date missionized SPIRIT SDK on the development platform. It is then transferred to the qualification environment to be validated. Once this step is completed, the production can be upgraded following the same process. Updating a platform with an already tested SDK only takes 20 to 30 minutes for an experimented maintainer.

Note that on NεSS and YODA missions, a single GPFS (General Parallel File System) file set was mounted on both qualification and production platforms to ease processes and data transfer. This is particularly useful to reproduce issues that were identified on the operational platform, and investigate on the qualification one.

5.5 SPIRIT agile improvement

Throughout the past year operating NεSS, few difficulties have been encountered directly linked to SPIRIT. For instance, one of the "Data" sub-service was generating a bigger amount of logs than expected, which led a Docker container to run out of available space within several days. Remaining on the service's side, a "Processing" container was facing an error every couple of hours, which would eventually block a whole sub-service in the end. These two events required the mission center operator to restart the orchestrator at least once or twice a week.

One last example: an issue was detected while performing tests on a SPIRIT version update. Any process executed on the remote computing cluster ended up in a failed state. It was later discovered that the problem came from a mission dedicated virtual environment that needed to be updated.

These three cases have in common that the difficulties were quickly instructed on the orchestrator team's side. Patches were identified following joint investigation sessions. Thanks to SPIRIT's agile development method, corrections could efficiently be prioritized and affected to future releases, taking in account other missions constraints.

Note that SPIRIT is an efficient and reliable orchestrator for missions with requirements compatible with its capabilities. Indeed, after over a year operating and maintaining NεSS mission center, no major service interruption occurred. Programming plans could be computed for each day the satellite was in nominal mode. A total of 20 000 acquisitions were performed onboard, downlinked, and treated on the ground-segment side. It resulted in the processing of 124 000 high-level products weighing 30TB.

5.6 SPIRIT remaining limitation

SPIRIT is currently known for having a limitation on its capability to handle massive amount and volume of data. Consequently, at the time of writing, some space missions, such as SWOT or TRISHNA, are not compatible with this solution. This is due to the orchestrator core technology, which is planned to be replaced in a future version that should be released in 2025. It emphasizes that this component is in perpetual evolution, and that the development team is mindful of projects' needs.

6. Conclusion

At the time of writing seven space programs managed by the CNES chose to involve SPIRIT in their ground segment. Among them, two satellites have already been launched: NεSS and SVOM, respectively on October 9, 2023, and June 22, 2024. They are currently demonstrating that this orchestrator is operational and reliable.

New missions are getting in touch with this tool on a regular basis, such as YODA, still under development, showing the rising interest of projects' teams for SPIRIT.

The installation and "missionization" procedures are not trivial to master, but once assimilated, thanks to its genericity, it becomes easy to replicate an existing mission center, or build from scratch a new one. In addition, due to its scalability, and its ability to be connected to a computing cluster, this orchestrator can cover a wide range of projects' needs while restraining the investment.

As mentioned in this paper, limitations have been identified on the SPIRIT release used by NεSS mission. However, the development team is working hard to improve the software functionalities and capabilities. The project current challenge is to switch the orchestrator core technology, to improve its scalability. It would allow space programs with massive amount and volume of data to choose this solution for their mission center. On average, a new version is delivered every month showing the team's intent to fit their customers' needs.

Acknowledgements

The author thanks members of CNES projects, such as NεSS, YODA, SPIRIT, HPCC, and many others for providing relevant information that could be integrated in this paper.

References

[1] Life Cycle assessment of Dell PowerEdge R740, June 2019, https://corporate.delltechnologies.com/content/dam/digitalassets/active/en/unauth/data-sheets/products/servers/lca_poweredge_r740.pdf , (accessed 13.02.25).