DOI: 10.13009/EUCASS2023-342
Aerospace Europe Conference 2023 – 10TH EUCASS – 9TH CEAS

A finite state machine approach to nano-satellite SW design: the HERMES case study

Margherita Piccinin⋆*, Andrea Brandonisio*⋆*, Andrea Colagrossi*⋆†*, Giovanni Zanotti*⋆*, Stefano Silvestrini*⋆
and Michèle Lavagna⋆
⋆*Politecnico di Milano - Via La Masa 34, Milano, Italy*
margherita.piccinin@polimi.it · andrea.brandonisio@polimi.it · andrea.colagrossi@polimi.it ·
giovanni.zanotti@polimi.it · stefano.silvestrini@polimi.it · michelle.lavagna@polimi.it
†Corresponding author

Abstract

In recent years, the number of launched small satellites, has exponentially grown, thanks to their low cost and modularity. Nevertheless, the software of such platforms still requires remarkable mission-specific tailoring efforts. In this paper, a linear, flexible and modular Finite State Machine (FSM) approach for nano-satellites software design is proposed, capable of being easily tailored to the specific mission and tuned according to variations of requirements and functionalities. The proposed FSM structure is composed of three major macro-modes, offering a time-driven mode and a scheduled mode for nominal operations, with a cyclic and basic safe mode, potentially applicable to every generic nano-satellite mission. The presented method has been used for the SW design of the HERMES mission, a 6 CubeSats constellation financed by Italian Ministry of University and Research (MUR), the Italian Space Agency (ASI) and the European Community (EC). HERMES FSM is presented in the paper, as case study for the method application.

1. Introduction

The Space community interest in small-sized spacecraft increased considerably in recent years, being the number of launched nano-satellites and CubeSats growing with an exponential trend in the last decade. According to current figures, until 1st January 2023 the total number of launched nano-satellites is 2138, with a forecast of about other 2000 nano-satellites to be launched in the next 6 years.[1] Small platforms are born with the idea of building a low cost solution and of allowing a quick development time. However, with the consolidation of small spacecraft technologies, they are being used to achieve more and more complex mission objectives. Indeed, their applications span from interplanetary CubeSats to Earth satellites for science, technological demonstrations and remote sensing. In particular, Low Earth Orbit (LEO) constellations offer the possibility to exploit a distributed instrument architecture, otherwise impossible to reach with just one satellite. With small-sized satellites, mega constellations have also become possible. Due to the increasing complexity of such missions, and to achieve high value mission goals it is not sufficient to simply assemble Commercial Off-The-Shelf (COTS) components, but customised and tailored solutions are often necessary.

A fundamental aspect for the success of such missions consists into the design of reliable on-board software, which is a vital element of the platform and in charge of performing Fault Detection Isolation and Recovery (FDIR) functions.[2] The definition of the software logic and routine workflows is a task with large complexity, as it involves major design choices and has a large impact on the whole system functioning.

The problem of nano-satellites software design is not new and, in literature, different approaches can be found, often relying on a Finite State Machine (FSM) to execute the mission functionalities in a deterministic manner.[3–5] However, there is still the lack of a consolidated and broadly recognised framework for designing the logic underlying the on-board software. Currently, it can be still observed a tight bond among the mission needs and the specific solutions that are consequently adopted in the software architecture. On the other hand, the nano-satellite market would benefit from a linear, modular and easily adaptable software framework.

With the purpose of making some steps forward in this direction, a multi-disciplinary and systematic approach to break down the software design process into effective steps is proposed and described in the paper. The proposed solutions and the adopted concepts tackle the typical challenges that can be found in many CubeSats LEO missions, such as the need of performing an automatic LEOP (Launch and Early Operation Phase) in a short pre-defined time, operating the platform via on-ground scheduled commands, and the necessity of a highly reliable software managing the low-cost components of the satellite. The described approach is then applied to the High Energy Rapid Modular

Copyright © 2023 by First Author. Posted on line by the EUCASS association with permission.

DOI: 10.13009/EUCASS2023-342

Ensemble of Satellites (HERMES) mission, funded by Italian Ministry of Research (MUR), the Italian Space Agency, (ASI) and the European Commission (EC), as a case study.[6,7]

HERMES space segment is composed of a LEO constellation of six nano-satellites, with the objective of detection, localisation and communication to ground of energetic astrophysical transients. To achieve such complex objectives, the on-board software is executed by two different OBCs on the spacecraft (i.e., OBC-MAIN and OBC-ADCS), while interfacing with other spacecraft's components, as the payload, the power module, the radios, the sensors and the actuators. Each OBC is driven by a dedicated software (i.e., SW-MAIN and SW-ADCS), which is structured according to its own FSM. The two pieces of software are interfaced by means of structured data commands and monitoring parameters, with a "master-slave" logical architecture.

The paper has therefore three main contributions:

- A general and effective approach for nano-satellite's software design is proposed, presenting design guidelines and a clear strategy to include FDIR functions.
- The proposed FSM-based method for the design and implementation of the on-board software can be easily standardised and applied to different scenarios and space missions, improving the flexibility and the modularity of classical FSMs.
- The method is applied to the HERMES mission, highlighting the main specific design challenges and presenting the results up to the formal verification of the software FSM in the framework of a validated environment.

The paper is structured as follows. After presenting a survey of nano-satellite software design approaches in Section 2, a structured and general method for small-spacecraft mission software design is proposed in Section 3. Then Section 4 introduces the HERMES case study, presenting the specific challenges and the adopted solutions of this application case while applying the proposed method for software design. The achievements are shown in Section 5, including the FSM verification and some multi-disciplinary analyses in support to the design choices. Finally, conclusions are drawn in Section 6.

2. Survey of nano-satellite software design approaches

Many nano-satellite software design approaches are founded on the principle of separating the on-board software in different layers to encapsulate different abstraction levels. This decision is frequently linked to the exploitation of one operating system (OS) in the fundamental levels. Commonly, third-party available OS are used, like FreeRTOS or GNU/Linux. The latter requires powerful on-board processors, and it is selected whenever high-level programming language or features available only in GNU/Linux are necessary.[8–10] However, if the design approach allows for low-level programming language, or limited performance OBCs are available, or saving processing resources is a mission requirement, an OS for embedded systems is preferred (e.g., FreeRTOS).[11]

In general, nano-satellites' flight software is required to be modular, extensible, flexible, robust and capable to handle on-board failures. These qualities are commonly implemented in the upper abstraction layer, which contains the main application software. This can be formalised and designed following a FSM approach or a component-based and service-oriented architecture. FSM provides a straightforward and neat implementation once all the mission's functional requirements are available, but it is typically less prone to include changes and follow the development process, indulging on modifications that affect the evolution of the states and transitions. On the contrary, centralised and decentralised architectures based on components and services, if well organised, are more flexible supporting incremental development and updated requirements.

2.1 FSM-based approaches

Finite state machines are defined as abstract computation models that at one time can be in exactly one of a finite number of states. The FSM changes from one state to another are driven by specified inputs, and they are denoted as transitions. To define a FSM, the full list of its states, the initial state and the inputs triggering each transition shall be declared. Then, a FSM allows to follow the operations of a spacecraft, transitioning among the different mission modes with different functionalities. Very simple spacecraft are entirely based on a hierarchical FSM, driving the entire on-board software,[12] while more complex architectures contain the FSM in the upper application layer.[8,13] In any case, a very structured hierarchical FSM, from one side makes the software design very plain and linear, customised for the specific mission modes, on the other end it requires deep and extensive modifications if any alteration is needed. In order to overcome such problems, hierarchical decomposition methods are being study in the field of automata theory.[14]

DOI: 10.13009/EUCASS2023-342

A FSM-based approach allows the support of formal validation and verification of the main software architecture. Indeed, the field of FSM formal specification and verification is very rich and consolidated.[15,16] Furthermore, standard and codified FSM description languages exists and are very well documented, like SDL (Specification and Description Language)[17] and, moreover, dedicated validation tools exist.[18] In the context of satellite on-board software formal verification, the ESA tool OpenGEODE, which is included in the TASTE environment,[19] provide a valuable example and useful instrument to validate the developed FSM.

2.2 FDIR methods

Spacecraft FDIR (Fault Detection, Isolation and Recovery) methods are at the base of on-board health monitoring and management systems. These techniques are fundamental for the reliability, availability and safety of a space mission. They are addressed and studied since the very beginning of the space era, because of the intrinsic problem of maintenance in space. Indeed, to minimise the risk of major failures, which may be caused by small initial issues associated to apparently negligible symptoms, it is paramount to quickly detect any anomaly occurring on-board, stopping its propagation to other system's components or subsystems.[2]

The rapid progress in on-board computational power supported the transfer of FDIR functions from the ground to the flight segment and the enhancement of the spacecraft on-board autonomy.[20] This is very common also in nano-satellite missions, not because of outstanding computing performance is available, but because an increased spacecraft autonomy is closely linked to a reduction in operating costs. Thus, there is a strong requirement to implement FDIR functions on-board modern nano-satellites, and this is commonly achieved by inserting these functionalities into dedicated software modules.

The nano-satellites' FDIR functions are typically very simple and their main requirements are focused on the reliability and the low computational burden. Often, they are based on sequential monitoring of primary parameters that are compared to thresholds, or on cross-checking of error and status flags. The detection of any anomaly in these monitoring steps is frequently directly associated to a transition to safe mode.[21] Then, nano-satellite missions are often equipped only with fault detection capabilities. The possibility to include an intermediate layer of autonomous failure isolation and recovery, before the transition to safe mode requiring ground intervention, is remarkably useful and may lead to a great improvement in the system performance and objectives.[22] The design of such complex software functionalities may benefit from a model-based design approach, achieving a monolithic component dedicated to the specific system configuration.[23] In alternative, it may be founded on a modular structure embedded in the FSM architecture. The separate software modules implement the distinct detection, isolation and recovery functions, and are strongly interconnected with the primary FSM. Moreover, these can be formally verified with the same approach adopted for the whole FSM, and their testing is directly performed at overall software level.[24,25] This last approach is the one also used in this research work.

2.3 Lessons learnt

The proposed approach to develop and implement a nano-satellite software with a linear and modular FSM structure tries to summarise the lesson learnt from previous nano-satellite missions. The method allows a formal verification of the entire on-board software in terms of system functionalities and components, including the possible transitions between the different software elements. Moreover, the distinct software states and modules can directly contain the fault detection, isolation and recovery blocks for a synergic integration of nominal and non-nominal phases. These characteristics have been proven to be effective in facilitating a positive outcome of the space mission.[26–28] Particularly, the modularity of such architecture, facilitating software updates and patches, and the close integration of FDIR routines with the primary software level, enabler of a thorough ground verification and testing of nominal and non-nominal transitions, are considered as best-practices leading to successful space missions. Furthermore, the limitation of the hierarchy levels in the FSM structure, in favour of a more linear architecture, reduces the possibilities of unforeseen anomalous transitions and facilitates the complete testing of all the software elements. These benefits are also suggested from real on-orbit experiences and lesson learned from previous missions.

3. Proposed finite state machine-based method

3.1 Method overview

The proposed approach for FSM design, is schematised in Fig. 1. First, starting from missions' typical high-level functionalities, the fundamental FSM macro-modes, such as LEOP, nominal routine (NOM) and hard safe mode (HSAFE),

DOI: 10.13009/EUCASS2023-342

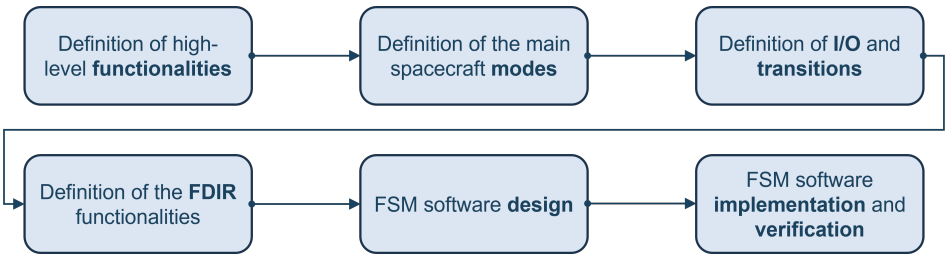


Figure 1: *Step-based* strategy for the design of general-purposed CubeSat mission definition.

are identified and the connecting transitions among them are defined. Then, the logic underlying each mode is described accounting for either the relevant requirements to be satisfied, the mission's operations and the actual functioning of the spacecraft's components. Secondly, in order to check the compatibility with the spacecraft's design and operations, the logic's design is supported and consolidated with analyses involving the spacecraft subsystems, such as the attitude determination and control (ADCS), the power generation and management, and the on-board data handling. During such process, particular attention is paid to the design of the nano-satellite's FDIR and to its translation into the FSM logic, which shall be achieved by means of monitoring functions that perform the Fault Detection (FD) and the soft safe (SSAFE) routines, which are devoted to the Fault Identification (FI) and, possibly,fault correction. Afterwards, the derived FSM logic is represented in detail with the Specification and Description Language (SDL), which allows the formal verification of the FSM and bridges the high-level definition to the final on-board software implementation.

3.2 Identification of general software functionalities

Despite of mission-specific objectives, a large number of functionalities need to be always granted by a general nano-satellite's on-board software. These include not only common low-level tasks, such as the boot and start-up of the system, the communication with peripherals and the monitoring of on-board components, but most importantly high-level functionalities. The latter range from on-board tasks as communication with ground, power generation, distribution and storage, and (for the more complex nano-satellites' missions) the attitude determination and control of the spacecraft; up to operational functionalities, related to the the ground planning and operation of the satellites, which are tightly related to the mission phases.

3.3 Proposed high-level software structure

Derivation of a generic backbone software structure In this section, a generally applicable backbone structure for a nano-satellite on-board software is proposed, adopting a FSM-based approach. In the framework of FSM approaches, the software backbone structure can be defined as the software modes themselves.

The key insight that leads to the definition of the FSM modes is the identification of the very different underlying working principles that are required to accomplish the various high-level software functionalities. In other words, the FSM modes can be found by clustering together functionalities that can be satisfied by implementing a software piece based on the same drivers. It is important to highlight that such identification process focuses on high-level software functions only, because it is noticed that the low-level functionalities should be always present throughout the whole software functioning.

For a small-sized spacecraft software, three high-level functionalities groups based on similar working principles are identified:

1. Tasks that shall be accomplished only once during the mission, i.e. at the beginning of the operations in space, for which a complete autonomy of the satellite is mandatory, because the system is not ready yet to communicate with ground.

2. Tasks that need to be repeatedly performed, for which the ground control and planning is desirable.

3. Tasks that can possibly be performed repeatedly during the mission, but that depend on contingencies and thus can not be planned or that cannot be handled immediately by the ground mission control.

FSM backbone states description From the above high-level clustering, it is found that a nano-satellite software can be structured upon three fundamental software macro-modes, which are here called LEOP, NOM and HSAFE.

DOI: 10.13009/EUCASS2023-342

1. The LEOP mode includes the operations that start with the spacecraft first switch-on in space that must be executed only once (such as system boot, appendages deployment and initial detumbling). The driving principle here is the one-time successful execution of a sequence of tasks. Moreover, due to the fact that in this phase batteries are the main power source, such tasks shall be completed within a reduced time-framework. It is thus proposed to base such mode on time-tagged commands that shall be repeated until the success assessment of any single task or expiration of the allocated time-out. Such software mode should be employed during the first part of the LEOP mission phase.

2. The NOM mode implies a regular communication with ground, hence it should be typically used during the spacecraft commissioning and during all nominal mission phases. It is here proposed to base the NOM mode upon scheduled commands, uploaded by ground during communication windows. Each command is paired with a scheduled time at which the on-board software triggers its execution. The sequence should be scheduled by ground keeping in mind the most proper sequence of operations for the platform (e.g. required attitude pointing and control mode, ground stations visibility windows, payload operational mode). These scheduled command can be either low-level, commanding basic functionalities and actions, or high-level, commanding automatic sequences of basic commands that achieve a more complex system action.

3. The HSAFE mode, instead, consists of automatic routines to follow when an error has been detected and identified without being able to recover the failure or, more generally, when a ground contact is immediately desirable and the system shall survive at the best, while trying to achieve the such communication. The HSAFE is driven by the maximisation of the spacecraft survivability, thus increasing the probability to communicate with ground and of maintaining a positive power balance.

Transitions among FSM states The entry and exit transitions of these three major software states are now described.

- The LEOP mode is entered if the time is below a threshold from the the first switch-on of the on-board computer in space. Such threshold should be defined and consolidated by means of analyses and tests. The LEOP mode is exited when the time is above this time threshold, with a transition to the HSAFE mode. This is due to the fact that the HSAFE mode guarantees a positive power balance, while maintaining the basic system functionalities and maximising the chances to get the first ground contact.

- The entrance to the NOM mode can be only commanded by ground, thanks to a communication contact happening in the HSAFE state and requiring the upload of a schedule. On the contrary, exiting from NOM mode is dictated by contingencies, i.e. failures and non-nominal situations detected and identified on-board, triggering the transition to the HSAFE mode.

- As already said, the HSAFE mode can be entered from both LEOP and NOM via on-board events and can be exited with a transition to NOM via ground intervention only.

The major advantage in exploiting such macro-modes is the possibility to develop a modular and linear FSM, that, contrary to hierarchical structures, is portable to different missions and more easily adaptable to the specific requirements.

3.4 Entailment of FDIR functionalities into the FSM

The big challenge of nano-satellite missions consists in achieving the mission objectives with a low cost design, which implies taking more risks. Indeed, the probability and severity of failures are often higher than in larger scale missions, due to different reasons, such as lack of redundancies, low design margins on power and telemetry budgets or reduced test campaigns that typically characterise small size spacecraft.

For this reason, particular attention shall be paid to the design of the nano-satellite's FDIR and to its translation and entailment into the FSM. For small spacecraft, the FDIR strategy shall be designed to grant the safety and the reliability of the system minimising on-board recovery actions and untracked autonomous transitions, while putting the system into a hard safe mode that maximises both the power budget margin the attempts to communicate to ground for intervention.

It is here proposed to include each FDIR function in the software FSM by means of monitors and SSAFE routines, as follows.

- **FD**. Some monitors shall continuously check the system variables that are necessary for the correct functioning of the platform. Such monitors shall be differentiated according to the FSM mode.

DOI: 10.13009/EUCASS2023-342

- **FI**. In case some failure is detected by the monitors, some dedicated routines, here called SSAFE, shall be in charge for the Fault Identification. The extreme exit condition of the SSAFE procedure is to request the entire system to transition into HSAFE, where the nominal functioning of the platform is stopped to fast and safely communicate with ground.

- **FR**. The FR functions are part of the SSAFE routines. Once an anomaly is isolated, the software can autonomously try to solve the possible problems arisen, usually and when possible, trying to reboot the error-related component. Only few other autonomous FR actions should be kept on-board, i.e. only when an immediate intervention is necessary to avoid a catastrophic loss of the system. In all the other cases, the FR shall be performed with the ground in the loop. For this purpose, the system stays in the HSAFE mode, while waiting for ground commands.

The FDIR monitors and routines run during all system modes, with the only exception of the LEOP. Indeed such phase is inherently designed to perform tasks accomplishment checks and it continuously tries to re-issue such tasks until success. In parallel to the schedule, during the NOM mode, parameters monitors and SSAFE routines are also included. It is important to underline that in HSAFE mode, the monitors and SSAFE routines are still active. The proposed SSAFE approach has the advantage of granting attempts of failure isolation and recovery autonomously, in certain conditions, before interrupting the current tasks and waiting for ground intervention. In this way, on one hand a prompt recovery can be attempted immediately at the failure time and on the other hand some simple failure cases can be overcome, having a positive impact on the mission operations.

3.5 Approach for detailed software design

After having clarified the general software structure and the entailment of FDIR functionalities, this section tackles the approach to be followed for the detailed design of the software. In this case, mission-specific factors heavily affect the design output, but still the approach to adopt during the detailed design can be standardised, by exposing the major parameters of interest in the SW. Some general guidelines for the detailed design are here provided.

The detailed design shall start from the knowledge of the functioning of the nano-satellite COTS components, in order to understand which parameters can be monitored and which commands can be provided. All the missing or custom desired software pieces shall be addressed to in very early design stages.

Besides the design, implementation and testing of the low-level software components, the FSM routines shall be verified following a multi-disciplinary approach, in order to keep an holistic view over the mission software, but at the same time accounting for subsystem-specific needs. The detailed design shall be based on system engineering and shall be performed with multiple iterations, in order to account for mission-specific components, tasks and operations.

It is also suggested to perform multi-disciplinary analyses in the domain of different subsystems, to be related to the expected software behaviour. In particular, attitude determination and control simulations, power budgets and housekeeping telemetry budgets. Some major common FSM-related trade-offs involving multiple subsystems are here identified as:

- definition of priorities and timing of the LEOP mode tasks sequence;

- selection of a restricted set of parameters to monitor and related SSAFE routines;

- definition of events to log and related severity.

The output of the detailed FSM design, shall be the clear schematics of the different modes, the monitored parameter lists and the schematics of the SSAFE routines.

3.6 Software implementation and verification

Once the conceptual definition of the FSM and FDIR have been detailed, the state machine can be implemented in the Opengeode environment, an open-source graphical editor developed by the European Space Agency (ESA), contained inside the TASTE package. In such a way, it is possible to directly identify possible faults and formal errors in the prototype logics, thus relying on a more accurate and with wider coverage FSM verification.

This tool exploits the SDL language needed for the formal description of finite state machines, in a safe and robust manner. Up to now, OpenGEODE is a technology demonstrator tool, therefore can only be used to describe and preliminary verify the implementation of the FSM. Being based on the SDL language, that has a defined syntax and semantics, the tool can perform several checks on the formal design of the state transitions, input and output parameters, and procedures' calling and definition.

DOI: 10.13009/EUCASS2023-342

During the code development and deployment, it is important to foresee testing techniques to inject failures, both hardware and software, into the system.

4. The HERMES case study

In the following section the HERMES mission case study is presented, starting from a brief explanation of the mission itself. Afterwards, the FSM approach developed and defined in the previous section is applied to the HERMES requirements, and the following outcome in terms of FSM design is described. In particular, it will be visible how all the steps defined in the proposed method can be easily shaped for the specific request of a single CubeSat mission, without losing the generality aspects that characterised the method, making it usable for various kinds of missions. The section is divided into four subsections, describing the HERMES mission and its main requirements, the software high-level structure, the FDIR functionality, and finally the detailed design.

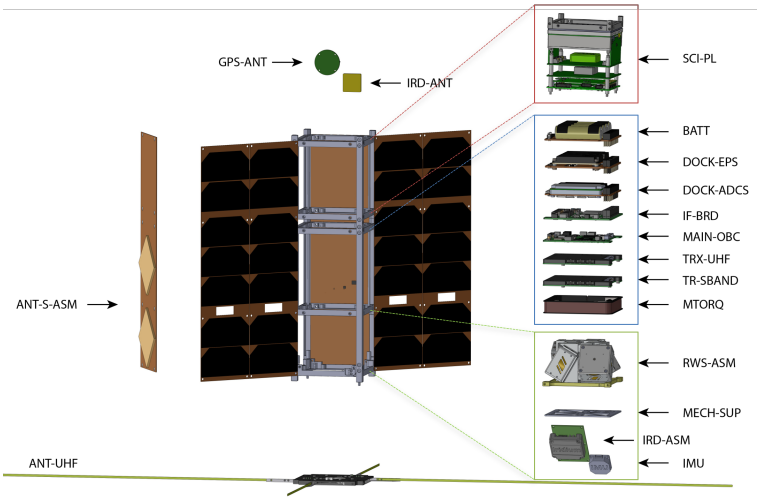


Figure 2: CAD description of the HERMES CubeSat's platform.

4.1 The HERMES mission

The High Energy Rapid Modular Ensemble of Satellites (HERMES) mission, is a CubeSat mission aimed to contributes to the so called Multi-Messenger Astrophysics. The HERMES' space segment is composed of a Low Earth Orbit (LEO) constellation of six nano-satellites. Each of them is a 3U CubeSat carrying a novel miniaturised detector sensitive to X and gamma-rays. Indeed, the mission aims at detecting and localising energetic astrophysical transients, such as short gamma-ray bursts (GRB), which are the electromagnetic counterparts of gravitational wave events. Once the detector detects the GRB, the system shall promptly communicate the localised event to ground. The constellation development is both funded by Italian Ministry of University and Research (MUR), the Italian Space Agency (ASI) and the European Community (EC).[6] Starting from these very high level requirements, needed by the spacecraft scientific payload, the main objective of the development of the HERMES platform can be summarised in the test and validation of the design of a robust and reliable CubeSat platform, equipped with a flexible and configurable attitude determination and control (ADCS) system, and a potential continuous and active link with ground. The HERMES' platform is shown in Fig. 2.

4.2 HERMES software high-level design with proposed method

Following the steps of the method proposed in Sec. 3, here the high level functionalities and design are described. As proposed in Sec. 3.3, the HERMES' software high level structure is divided into three main modes, LEOP, NOM, and HSAFE, grouping the three main functionality principles proposed for CubeSat's software. The LEOP mode is the one aimed to boot the system, the ADCS system, and the main communication system, in order to rapidly obtain the first contact with ground. Moreover, it is dedicated to the deployment of the folded appendages (e.g., communication antennas and solar arrays) and to detumble the satellite. NOM is the mode in which the platform performs attitude manoeuvres, commands the scientific payload, connects to ground with all the communication systems, and monitors the satellite critical parameters. On the other hand, the HSAFE mode is dedicated to avoid any waste of power, to fast

DOI: 10.13009/EUCASS2023-342

communicate with ground, to control the ADCS to maximise the power generation or to handle non nominal situation, such as tumbling or unexpected wheel saturation. To achieve the very complex mission's objectives, the on-board software for the HERMES nano-satellites is executed by two different OBCs on the spacecraft, namely OBC-MAIN and OBC-ADCS. The OBC-MAIN is the interface between all the spacecraft's components, as the scientific payload, the power and telecommunication modules, and the OBC-ADCS itself. Differently, the OBC-ADCS interfaces with all the attitude determination sensors and control actuators.

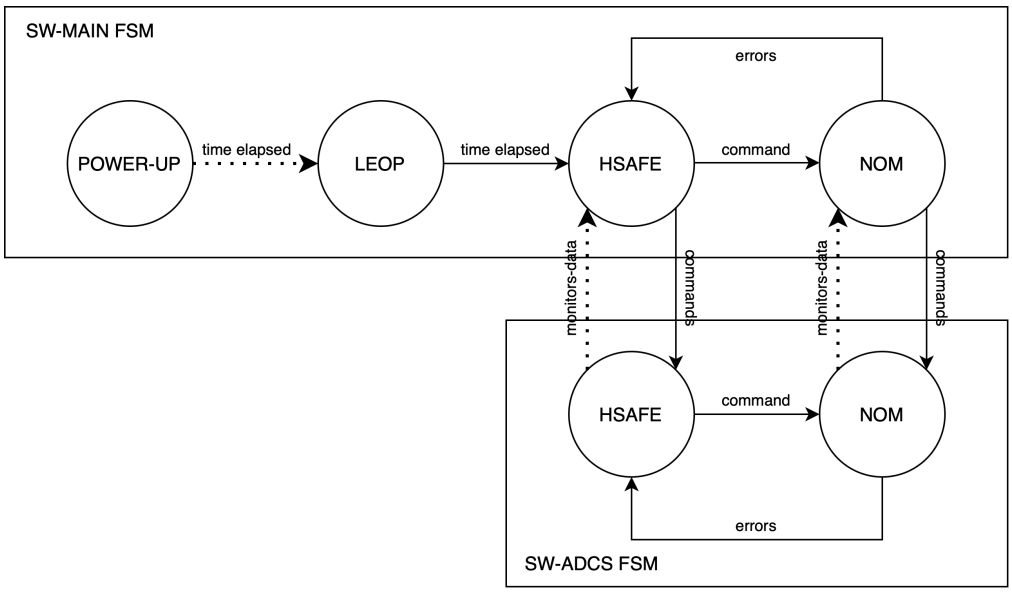


Figure 3: HERMES' software high-level structure.

The software running in each of the two OBCs, named SW-MAIN and SW-ADCS respectively, is structured according to its own FSM, that still follows the functionalities presented before. The two are interfaced by means of structured data commands and monitoring parameters, with a *master-slave* logical architecture. Specifically, the SW-ADCS is subordinated to the requests and commands defined by the SW-MAIN. This high-level structure can be depicted in Fig. 3. It can be noticed that both software's FSM reflect the same architecture defined by the three main modes: LEOP, NOM and HSAFE. The only exception is the lack the LEOP mode for SW-ADCS, useless because subordinated to the SW-MAIN. The *master-slave* relation is justified by the fact that SW-ADCS exchanges with SW-MAIN counterpart monitors and data, while only receiving commands.

In Fig. 3, the transaction between modes with their relative entry and exit conditions are defined exactly as those presented in the general-purpose methodology of Sec. 3.3: HSAFE is used as the LEOP exit condition, since it is designed to be power positive and to look for a ground contact; while the first schedule upload allows transition to NOM mode.

4.3 HERMES software detailed design with proposed method

In order to be compliant with the FDIR functionalities described in Sec. 3.4, also the HERMES FDIR strategy is based on the concept of the SSAFE routines. To recap, the SSAFE object is a continuous monitor, that runs during both the NOM and the HSAFE phase of the mission, and is in charge of continuously checking HERMES vitals parameters in order to detect possible failures or errors in the system. Afterwards, it tries to solve the problem, with simple FR procedures, otherwise, request the entire system in HSAFE.

In the next paragraphs, each of the software's modes will be briefly analysed more in detail following the approach defined in Sec. 3.5, in order to generate as output the clear schematics of the modes' architecture , the overall monitor parameters list, and consequently the schematics of the SSAFE routines. The description will be divided between the two software's design: SW-MAIN and SW-ADCS.

4.3.1 HERMES SW-MAIN finite state machine

The FSM of the HERMES SW-MAIN is divided into three main modes: LEOP, NOM and HSAFE. The design of each mode is hereby briefly described.

DOI: 10.13009/EUCASS2023-342

LEOP mode. As proposed in Sec 3.3 every operation in the SW-MAIN LEOP mode is time tagged. Nominally, the correct operational order is the one schematised in Fig. 4, and described in the following list. It is important to understand that, since the process is time-tagged, all the operations can happen only in a defined time slot, within which the SW attempts cyclically to execute each task, as visible again in Fig. 4.

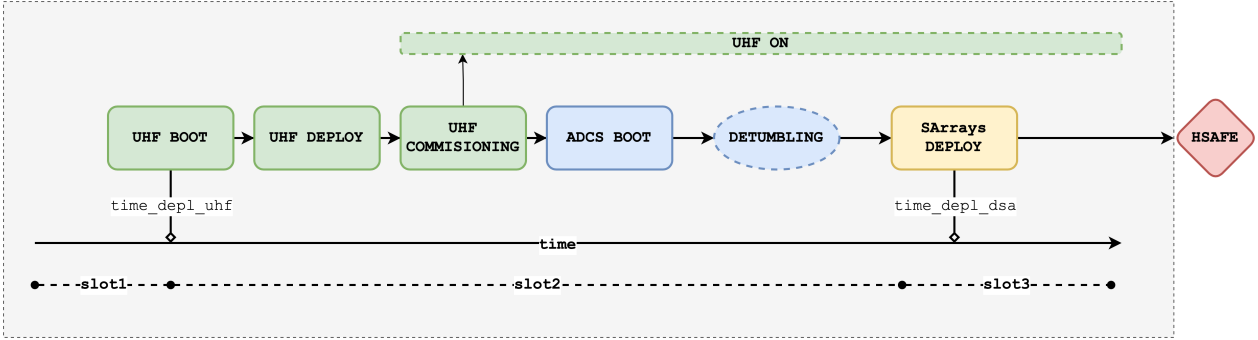


Figure 4: SW-MAIN LEOP mode operational schematic.

- At first, the OBC-MAIN board switches on and performs its automatic power-up and boot procedure.
- Once the OBC is operative, it waits until the end of the short *slot 1*, it boots the UHF board (one of the three HERMES' telecommunication systems) and then it deploys the two antennas. If the boot and deployment complete successfully, the UHF is commissioned and the board starts sending a beacon message to ground in order to acquire the first contact.
- After the UHF operation, the LEOP mode boots the OBC-ADCS board.
- Then, if the boot is successful the software commands the ADCS to activate the detumbling mode, in order to detumble the spacecraft before trying the deployment of the solar arrays, that shall not sustain high angular velocities.
- When the time *slot 2* is concluded, even if the ADCS has not managed to completely detumble the satellite, the SW commands the deployment of the spacecraft's solar arrays. This is because the risk of opening the solar panels at high tumbling rate is minor of the one associated to running a long detumbling mode on battery with folded panels.

Once this process is concluded, the OBC system automatically enters the HSAFE mode. The LEOP mode is constructed to fast and safely execute the solar panels deployment: indeed, without them, the survival of the platform is practically impossible and therefore, the mission may be completely lost. This underlines the importance of having a LEOP phase that is time tagged and not event tagged in order to ensure the panels deployment at the defined time, even if all the previous operations have failed.

NOM mode. The SW-MAIN nominal mode is based on the considerations done in Sec. 3.3, i.e. the software operates following the commands present in a defined and delivered schedule, which is periodically uploaded from ground during the satellites' operations. The scheduled commands do not correspond to a single action, but they will be implemented as Scripted Commands. Such scripts can be classified into two groups: Routine Scripted Commands, that will be used mainly during the nominaloperations, and the Commissioning Scripted Commands, that will be employed only during the commissioning phase. The main concepts of the Routine Scripted Commands operation are listed in the following:

- Manoeuvre commands: used to command a manoeuvre to the ADCS. The inputs to this script include the guiding polynomial and the preferred actuator.
- Start, make, and stop observation commands: used to enter and exit the scientific observation mode exploiting the scientific payload and the Iridium system as a low latency communication channel.
- Data transfer commands: used to transfer data from the scientific payload to the OBC-MAIN internal memory.

DOI: 10.13009/EUCASS2023-342

- Start, make, and stop communication window commands: used to handle ground communication windows. Input arguments include the communication channels to be used and the ground stations to communicate with. Nominally with SBAND telecommunication system.

Differently, the NOM mode Commissioning Scripted Commands activities can be summarised as:

- Full commissioning of ADCS subsystem.
- Full commissioning of Telemetry Tracking and Command (TT&C) subsystem.
- Full commissioning of Electric Power Subsystem (EPS).
- Full commissioning of the Payload.

These or additional scripted commands can be uploaded and modified from ground, thus guaranteeing flexible operations. Moreover, the schedule can also contain basic low-leve actions and functionalities to be executed. As mentioned before, meanwhile the Scripted Commands are executed in the NOM state, the SSAFE monitor and routines are included and periodically run to grant the FDIR functionalities of the SW-MAIN and the entire HERMES system.

HSAFE mode. As introduced before, some nominal monitors can lead the system to enter the HSAFE mode, due to components' errors or failures. The logical architecture of the HSAFE mode has a tree structure, where system checks determine branches, which finally can lead to possible actions. Two main parameters manage the functioning of the HSAFE process: the battery voltage and the ADCS status. The battery voltage is necessary to understand the level of power that is still available in the platform, while the ADCS status can have different output in terms of direct SW-ADCS requests or availability status, determining the operating mode of the satellite. In Fig. 5, the logical scheme of the HSAFE mode is described. When the system enters HSAFE, the on-board computer powers off all HERMES' telecommunication systems, puts the ADCS in standby and the payload in power-save mode. In case of errors coming from the ADCS or the payload, their shut-down is enforced by calling the respective procedures. After such operations, which constitute the initialisation phase of HSAFE, the process enters the main central loop, which is based on the continuous check of battery voltage and ADCS status, and that can be exited only by a ground command.

Depending on the battery and ADCS status, the process may lead to five different branches (as in Fig. 5) depending on the status of its sensors and actuators: fatal error, detumbling, desaturation, safe or nominal ADCS branches. The battery status can directly lead to the fatal error branch if its status is below a certain voltage threshold. If the battery is in critical condition, no operations are performed because the entire system will be automatically shut-down. In all other cases, the main operation of the FSM is selecting the best available communication instrument in order to connect with ground as fast as possible.

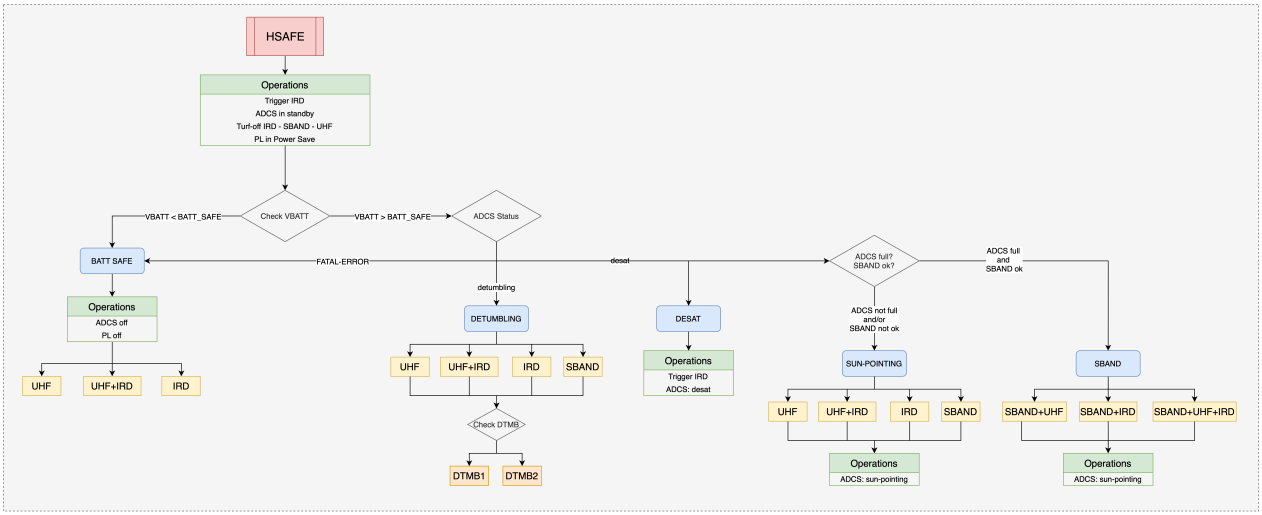


Figure 5: SW-MAIN HSAFE mode operational schematic.

4.3.2 HERMES SW-ADCS finite state machine

The FSM of the HERMES SW-ADCS is divided into two main modes: NOM and HSAFE. The design of each mode is hereby briefly described.

DOI: 10.13009/EUCASS2023-342

NOM mode. The SW-ADCS NOM mode provides the nominal interaction between the ADCS algorithms and the monitors and SSAFE routines of the ADCS COTS components. The software is designed to have two main parts, the ADCS algorithm, and the ADCS SSAFE monitors. The first one is dedicated to the spacecraft GNC, and its operative mode is directly commanded by the SW-MAIN schedule, while the latter reads the algorithm generated flag to safely monitor ADCS software, sensors, and actuators. If an error or failure occurs and the relative SSAFE procedure does not manage to solve it, it directly exposes the error to the SW-MAIN, that will switch mode to HSAFE. The architecture reflects the *master-slave* interaction defined in Sec. 4.2. The logical architecture scheme is depicted in the right side of Fig. 6.

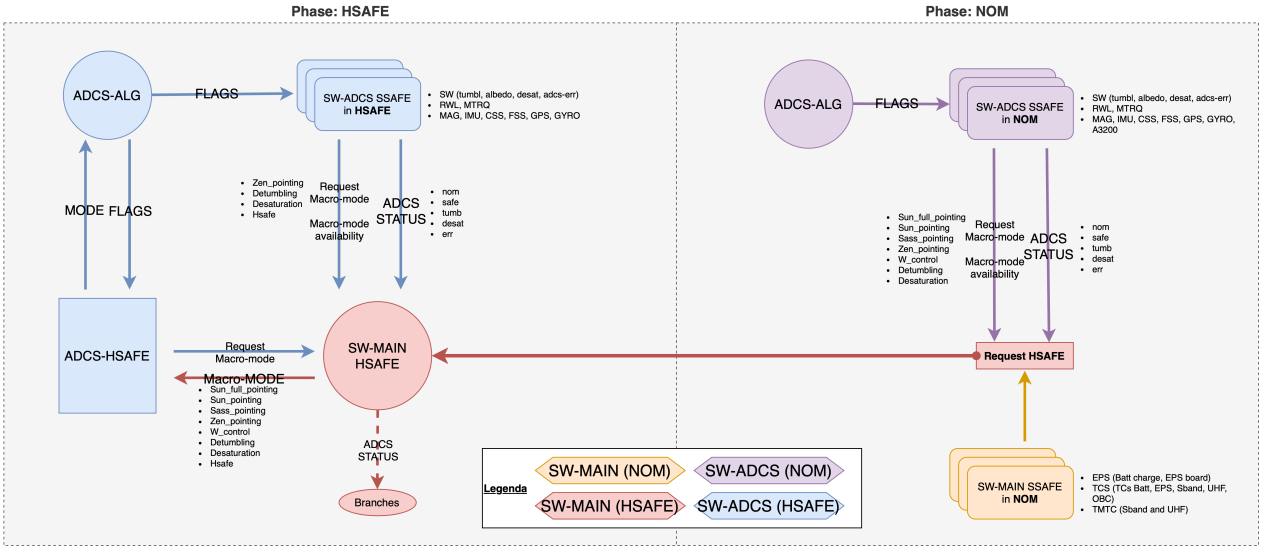


Figure 6: SW-ADCS NOM and HSAFE modes operational schemes.

HSAFE mode. The SW-ADCS HSAFE mode still provides the coupling between ADCS algorithms, monitors and SSAFE routines, with an additional element, active only in this mode, aimed to select the correct ADCS operative mode, following the requests and instruction of the SW-MAIN HSAFE tree logic. Once the SW-MAIN system switches to HSAFE mode, even if the transition is not caused by an ADCS failure, also the ADCS-SW switches to its HSAFE mode. Here the software is composed of three actors: the usual ADCS algorithm, the ADCS SSAFE monitors and the ADCS-HSAFE logic, as described in the left side of Fig. 6. The first does not change with respect to NOM phase, still requesting the mode command and exposing flags. The monitors, in HSAFE mode, slightly change, but still outcome the ADCS status and the modes availability and request a different set of *macro-modes* (set of modes characterised by the same subset of sensors and actuators). The new element, instead, is the ADCS-HSAFE logic, that becomes the bridge between the SW-MAIN HSAFE mode and the ADCS algorithm. Indeed, the HSAFE procedure reads the monitor macro-mode requests, if present, and tells the macro-mode to the ADCS-HSAFE; here, depending on the macro-modes availability the first compatible mode is then commanded to the ADCS algorithm.

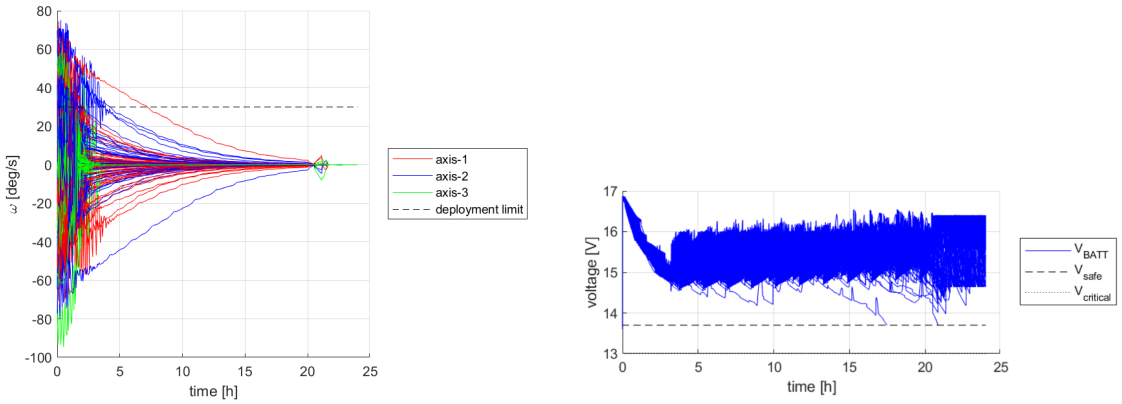
5. Results

In this section, some results of different analyses are reported, in order to highlight how the presented and implemented methods are exploited to drive the design of the on-board software.

5.1 Inter-disciplinary analyses

In support to the FSM design, multi-disciplinary analyses were carried-out, most importantly regarding the ADCS and EPS subsystems. Due to the limited available on-board resources, a crucial challenge for the HERMES mission was to select and correctly scheduling the different operating modes in relation to the SW architecture. The two subsystems models were run in open-loop, accounting for the FSM structure as the high level SW commands of the subsystems modes.

DOI: 10.13009/EUCASS2023-342



(a) Angular velocity.

(b) Battery voltage.

Figure 7: Montecarlo simulation during LEOP (50 runs).

Example of simulation impacting on the FSM design. An example of analysis that affected the FSM design is here reported, where multi-disciplinary Montecarlo simulations of the LEOP phase have been carried-out. In Fig. 7a the angular rate of the satellite during the LEOP is depicted, while in Fig. 7b the corresponding batteries voltage is shown.

The general success rate of the simulations is 96% and the statistical values of the final angular velocity are reported in 1. The detumbling is always successfully completed. The batteries final state of charge is reported in Table 1 as well, showing that the batteries are effectively recharged during the sun pointing phase. As shown in Figure 7b,

Table 1: LEOP final angular velocity and battery voltage.

Variable	**mean**	**std (1σ)**
ω_x [$^\circ/s$]	3.00e-04	9.00e-04
ω_y [$^\circ/s$]	-1.00e-04	7.00e-04
ω_z [$^\circ/s$]	1.60e-03	9.30e-03
`V-BATT` [V]	15.595	0.495

the satellite relies on batteries at the beginning of the LEOP, while in almost all the cases the batteries are recharged as soon as the solar arrays are deployed.

The major outcome from the analyses was the consolidation and verification of operation sequence and the need to of adopting strategies to eliminate the failure cases in which the batteries voltage drops down below the critical value, like:

- earlier deployment of the solar arrays;
- reduced communications for power save before the LEOP end.

5.2 FSM implementation and verification in *OpenGEODE*

Following the steps of the proposed method for the FSM design, the last passage consists in the development and implementation of the FSM and FDIR routines in SDL within the OpenGEODE environment, which allows the formal verification of the logical architecture, as briefly introduced in Sec. 3.6. Using SDL language for these kind of tasks can be very useful since the language is formally completed, and therefore, it can be directly used to generate the simulation or target code. The SDL language relies on four main aspects: structure, communication, behaviour, and data.

- The structure of the system is made by different function blocks that can be further decomposed in other blocks. The lowest level of block is instead defined with a series of one or more processes, describing the actual operations of the FSM.
- Blocks and sub-blocks are connected through communication channels that carry internal or external signals, depending if the communication must occur between blocks of the same or different levels respectively.

DOI: 10.13009/EUCASS2023-342

- The process inside each low level block describes all the action that the system shall perform. It could consists in reading external input, sending messages, taking actions (cycle, assignment), making decision or call procedures.

- All the variables defined in the process must follow a syntax, which supports pre-defined data types (e.g. integer, float, boolean, string, ...)

This simple and well-defined formalism makes the SDL language reliable for the sizing of robust software. Indeed, being HERMES a 3U CubeSat mission, the on-board software needs to be simple and reliable. Therefore, the choice of OpenGEODE as editing tool is useful to reduce the complexity of the state chain, streamlining the checking parameters selection and the procedures definition. This is why all the HERMES FSM modes and SSAFE routines, defined in Sec. 4, have a verified OpenGEODE formulation: each of these are based on inner procedures which are again implemented and verified with OpenGEODE.

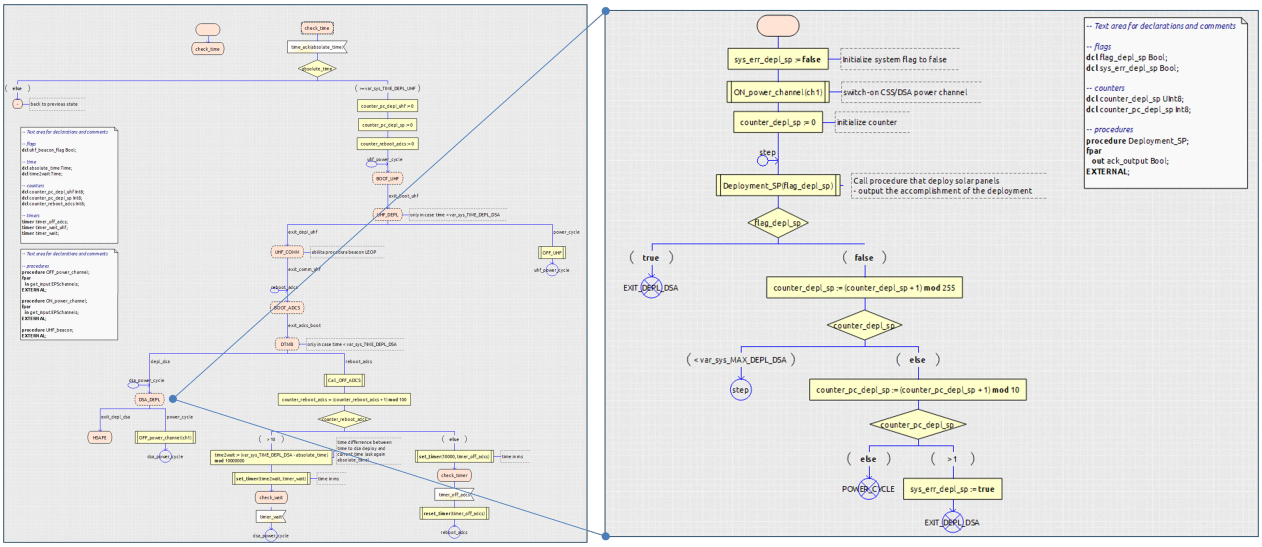


Figure 8: OpenGEODE implementation of the SW-MAIN LEOP mode, with a inside look at the Solar Array deployment state.

An example of the formalisation of HERMES' finite state machine in OpenGEODE is shown in Fig. 8, that presents the verified SW-MAIN LEOP mode. As explained in Sec. 3.6, the overall FSM implementation in OpenGEODE is used as formal verification of the logical architecture designed for the different modes of the two satellite's software, that is, therefore, formally verified. The next step, instead, may be related to the direct translation of the OpenGEODE's SDL code in the final FSM coded in the real spacecraft's OBC systems.

6. Conclusion

In conclusion, three main objectives have been satisfied by the work presented in the paper:

- A specific *step-based* strategy for the design of general-purposed nano-satellite mission software functionalities has been defined.

- An easy way to structure the FSM for this kind of missions has been proposed.

- A simple and yet effective way to formally verify the design of the FSM has been analysed and proved.

At last, the paper proposes an effective architecture for the HERMES' mission software, based on the presented FSM design approach, also including the satellite's FDIR strategy. A multi-disciplinary simulator is used to help verifying the LEOP sequence and tuning some parameters to ensure the CubeSat survivability. Finally, each FSM part is defined in SDL, with the OpenGEODE tool, and formally verified.

DOI: 10.13009/EUCASS2023-342

7. Acknowledgments

The authors would like to acknowledge the entire HERMES project consortium, composed of the Italian Space Agency (ASI), Italian Institute of Astrophysics (INAF), Politecnico di Milano, Cagliari University, National Institute of Higher Mathematics (INdAM), Skylabs Technology, Deimos Space, Nova Gorica University, Tübingen University, Loránd Eötvös University, Aalta Lab, C3S Electronics.

The authors want to acknowledge the Italian Ministry of University and Research (MUR), the Italian Space Agency (ASI) and the European Commission for the funding of the HERMES project.

References

[1] E. Kulu, "Nanosatellite launch forecasts-track record and latest prediction," in *36th Annual Small Satellite Conference 2022*, 2022.

[2] M. Tipaldi and B. Bruenjes, "Spacecraft health monitoring and management systems," in *2014 IEEE Metrology for Aerospace (MetroAeroSpace)*, pp. 68–72, IEEE, 2014.

[3] K. V. de Souza, Y. Bouslimani, and M. Ghribi, "Flight software development for a cubesat application," *IEEE Journal on Miniaturization for Air and Space Systems*, vol. 3, no. 4, pp. 184–196, 2022.

[4] I. Latachi, T. Rachidi, M. Karim, and A. Hanafi, "Reusable and reliable flight-control software for a fail-safe and cost-efficient cubesat mission: Design and implementation," *Aerospace*, vol. 7, no. 10, p. 146, 2020.

[5] S. Buckner, C. Carrasquillo, M. Elosegui, and R. Bevilacqua, "A novel approach to cubesat flight software development using robot operating system (ros)," in *34th Annual AIAA/USU Small Satellite Conference*, 2020.

[6] F. Fiore, L. Burderi, M. Lavagna, R. Bertacin, Y. Evangelista, R. Campana, F. Fuschino, P. Lunghi, A. Monge, B. Negri, *et al.*, "The hermes-technologic and scientific pathfinder," in *Space Telescopes and Instrumentation 2020: Ultraviolet to Gamma Ray*, vol. 11444, pp. 214–228, SPIE, 2020.

[7] A. Colagrossi and M. Lavagna, "Fault tolerant attitude and orbit determination system for small satellite platforms," *Aerospace*, vol. 9, no. 2, p. 46, 2022.

[8] S. Johl, E. Glenn Lightsey, S. M. Horton, and G. R. Anandayuvaraj, "A reusable command and data handling system for university cubesat missions," in *2014 IEEE Aerospace Conference*, pp. 1–13, 2014.

[9] M. Schmidt and K. Schilling, "An extensible on-board data handling software platform for pico satellites," *Acta Astronautica*, vol. 63, no. 11, pp. 1299–1304, 2008.

[10] C. Mitchell, J. Rexroat, S. A. Rawashdeh, and J. Lumpp, "Development of a modular command and data handling architecture for the kysat-2 cubesat," in *2014 IEEE Aerospace Conference*, pp. 1–11, 2014.

[11] C. E. Gonzalez, C. J. Rojas, A. Bergel, and M. A. Diaz, "An architecture-tracking approach to evaluate a modular and extensible flight software for cubesat nanosatellites," *IEEE Access*, vol. 7, pp. 126409–126429, 2019.

[12] P. Fiala and A. VobornÃk, "Embedded microcontroller system for pilsencube picosatellite," in *2013 IEEE 16th International Symposium on Design and Diagnostics of Electronic Circuits and Systems (DDECS)*, pp. 131–134, 2013.

[13] J. Guo, J. Bouwmeester, and E. Gill, "In-orbit results of delfi-n3xt: Lessons learned and move forward," *Acta Astronautica*, vol. 121, pp. 39–50, 2016.

[14] O. Biggar, M. Zamani, and I. Shames, "Modular decomposition of hierarchical finite state machines," *arXiv preprint arXiv:2111.04902*, 2021.

[15] A. R. Flora-Holmquist and M. G. Staskauskas, "Formal validation of virtual finite state machines," in *Proceedings of 1995 IEEE Workshop on Industrial-Strength Formal Specification Techniques*, pp. 122–129, IEEE, 1995.

[16] D. Lee and M. Yannakakis, "Testing finite-state machines: State identification and verification," *IEEE Transactions on computers*, vol. 43, no. 3, pp. 306–320, 1994.

[17] P. Poizat, "Sdl: a language based on extended finite state machines with abstract data types," *Software Specification Methods: An Overview Using a Case Study*, pp. 147–164, 2001.

DOI: 10.13009/EUCASS2023-342

[18] S. C. P. F. Fabbri, J. C. Maldonado, and M. Delamaro, "Proteum/fsm: a tool to support finite state machine validation based on mutation testing," in *Proceedings. SCCC'99 XIX International Conference of the Chilean Computer Science Society*, pp. 96–104, IEEE, 1999.

[19] I. Dragomir, M. Bozga, I. Ober, D. Silveira, T. Jorge, E. Alana, and M. Perrotin, "Formal verification of space systems designed with taste," *arXiv preprint arXiv:2111.10132*, 2021.

[20] M. Tipaldi and L. Glielmo, "A survey on model-based mission planning and execution for autonomous spacecraft," *IEEE Systems Journal*, vol. 12, no. 4, pp. 3893–3905, 2017.

[21] L. Franchi, L. Feruglio, R. Mozzillo, and S. Corpino, "Model predictive and reallocation problem for cubesat fault recovery and attitude control," *Mechanical Systems and Signal Processing*, vol. 98, pp. 1034–1055, 2018.

[22] M. Tipaldi, S. Silvestrini, V. Pesce, and A. Colagrossi, "Chapter eleven - FDIR development approaches in space systems," in *Modern Spacecraft Guidance, Navigation, and Control* (V. Pesce, A. Colagrossi, and S. Silvestrini, eds.), pp. 631–646, Elsevier, 2023.

[23] J. S. Lobo, P. Ghiglino, S. L. Escobedo, M. S. Rivo, and K. Robotics, "Design of a model-based failure detection isolation and recovery system for cubesats," 2019.

[24] M. C. Vitelli, M. Tipaldi, and L. Troiano, "A domain specific language oriented to fault detection, isolation and recovery," in *2013 International Conference on Soft Computing and Pattern Recognition (SoCPaR)*, pp. 343–348, IEEE, 2013.

[25] O. Durou, V. Godet, L. Mangane, D. Pérarnaud, and R. Roques, "Hierarchical fault detection, isolation and recovery applied to cof and atv avionics," *Acta Astronautica*, vol. 50, no. 9, pp. 547–556, 2002.

[26] C. Gonzalez, C. Rojas, A. Becerra, J. Rojas, T. Opazo, and M. Diaz, "Lessons learned from building the first chilean nano-satellite: The suchai project," 2018.

[27] J. Scharnagl, R. Haber, V. Dombrovski, and K. Schilling, "Netsat - challenges and lessons learned of a formation of 4 nano-satellites," *Acta Astronautica*, vol. 201, pp. 580–591, 2022.

[28] A. Slavinskis, M. Pajusalu, H. Kuuste, E. Ilbis, T. Eenmäe, I. Sünter, K. Laizans, H. Ehrpais, P. Liias, E. Kulu, *et al.*, "Estcube-1 in-orbit experience and lessons learned," *IEEE aerospace and electronic systems magazine*, vol. 30, no. 8, pp. 12–22, 2015.