https://doi.org/10.5028/jatm.v11.1081 ORIGINAL PAPER

A Comparative Survey on Flight Software Frameworks for 'New Space' Nanosatellite Missions

Danilo José Franzim Miranda[1,2,*], Maurício Ferreira[3], Fabricio Kucinskis[1], David McComas[4]

Miranda DJF https://orcid.org/0000-0002-9186-1740
Ferreira M https://orcid.org/0000-0002-6229-9453
Kucinskis F https://orcid.org/0000-0001-6171-761X
McComas D https://orcid.org/0000-0002-2545-5015

How to cite

Miranda DJF; Ferreira M; Kucinskis F; McComas D (2019) A Comparative Survey on Flight Software Frameworks for 'New Space' Nanosatellite Missions. J Aerosp Technol Manag, 11: e4619. https://doi.org/10.5028/jatm.v11.1081

ABSTRACT: Nanosatellite missions are becoming increasingly popular nowadays, especially because of their reduced cost. Therefore, many organizations are entering the space sector due to the paradigm shift caused by nanosatellites. Despite the reduced size of these spacecrafts, their Flight Software (FSW) complexity is not proportional to the satellite volume, thus creating a great barrier for the entrance of new players on the nanosatellite market. On the other side, there are some available frameworks that can provide mature FSW design approaches, implying in considerable reduction in software project timeframe and cost. This paper presents a comparative survey between six relevant flight software frameworks, compared according to commonly required 'New Space' criteria, and finally points out the most suitable one to the VCUB1 reference nanosatellite mission.

KEYWORDS: Flight Software, On-Board Software, NASA cFS, New Space.

INTRODUCTION

Flight Software (FSW) is software that runs on a processor embedded in a spacecraft's avionics. It is responsible for managing on-board activities, data processing and spacecraft health and safety.

The name "flight software" reflects the location where it executes, i.e. in the spacecraft, to differentiate from "ground software", which runs in the ground segment. It is considered a high-risk system because it interacts directly with spacecraft hardware, controlling virtually all of the onboard systems in real time at various levels of automation (Dvorak 2009). In such a domain, the level of failure protection and recovery is generally higher than in usual computer systems as a result of the lack of accessibility to the hardware after launch, as well as the space environment intrinsic characteristics.

In a space mission, the responsibility for the tasks realization is shared between the ground and the space segment. The first space missions allocated almost all tasks to the ground segment. In such missions, the space segment was a mere executor of the commands sent from the ground. By adding computers to the space segment, the operators were allowed to start delegating tasks to on-board execution (Kucinskis and Ferreira 2013). With the evolution of spacecraft on-board computer and electronics, as well as the gain of confidence on autonomous spacecraft operations, more responsibility was given to the flight software. Nowadays

1.Instituto Nacional de Pesquisas Espaciais – Engenharia e Tecnologia Espacial – São José dos Campos/SP – Brazil.
2.Visiona Space Technology – São José dos Campos/SP – Brazil.
3.Instituto Nacional de Pesquisas Espaciais – Centro de Controle de Satélites – São José dos Campos/SP – Brazil.
4.National Aeronautics and Space Administration – Goddard Space Flight Center – Flight Software Branch – Greenbelt/MD – United States.
Correspondence author: danilojfmiranda@gmail.com
Received: Nov. 25, 2018 | **Accepted:** Feb. 26, 2019
Section Editor: Luiz Martins-Filho

missions are increasingly demanding more powerful, autonomous, robust and flexible flight software to meet complex mission requirements.

Despite that fact, Eickhoff (2012) states that the on-board computer, flight software and the operations design are normally a challenge for new entrants in the space field, especially due to the lack of specialized literature.

On the other side, there are some open-source frameworks devoted to space projects. They were created by space agencies, universities, or space companies to serve as reference designs and promote code reuse in their future projects. Being a relatively new option for FSW development, these frameworks have not so far achieved a significant adoption outside their host organizations.

The start of a FSW project from scratch, especially for new organizations, can be a very expensive endeavor. Usual software systems development normally profits from existing Software Development Kits (SDK), frameworks, design patterns and different tools to support the developer's work. In spacecraft projects, code reuse and framework usage are not wide spread concepts.

Schmidt *et al.* (2004) state that the use of a well-established software framework leads to a shorter development program, with reduced costs and improved quality. These consequences are beneficial for space organizations, which is what motivated the current survey.

Six relevant FSW frameworks were analyzed according to the nanosatellite reference mission software selection criteria. This project being typical of what is called 'New Space'.

The reference mission, named VCUB1, is an in-orbit demonstration 6U CubeSat under development at the Brazilian space company Visiona Space Technology S.A. This CubeSat project is a typical New Space one: small team, short schedule, reduced budget, piggyback launch, best-effort reliability paradigm (almost no equipment redundancy on-board, no rad-hard protection in most equipment), simplified verification and validation (V&V) cycle on a single spacecraft protoflight unit.

Regarding the FSW, starting the design from scratch was not a feasible option due to the project tight schedule and limited resources. It was decided to benefit from an existing flight-proven FSW framework to serve as design paradigm.

RELATED WORK

There are few comparisons of frameworks for spacecraft use. One of them was presented by Rexroat (2014). He compares several candidate middleware, aiming at his target distributed embedded systems, which were Unmanned Air Vehicles (UAV) and CubeSats. Nevertheless, the author considered not only service-level middleware, but also lower level middleware at Network (e.g. SpaceWire) and Distribution Layer (e.g. MIL-STD-1553) levels.

Some of Rexroat's selection criteria were:

- Network communication – node-oriented versus message-oriented systems
- Coordination – synchronous versus asynchronous behavior
- Reliability – in terms of message delivering: at-most-once, at-least-once, exactly-once
- Scalability – abstraction in the following aspects: access, location, migration and replication
- Heterogeneity – portability and interoperability with different hardware, network or software

MAVLink, which was originally developed for the UAV world, was his middleware solution of choice for UAVs and CubeSats, NASA core Flight System (cFS) being second in the rank.

The present work focuses, differently from Rexroat, on software design aspects for a nanosatellite mission rather than in network aspects.

'NEW SPACE' AND THE VCUB1 MISSION

The changes caused by the greater involvement of the private sector in the global space activity in the past several years is called 'New Space' (Paikowsky 2017). This term is used as the opposite for 'Old Space', i.e. the existing ecosystem since the beginning of space activities, characterized by being mainly controlled by national activity and mostly a State-only business.

In New Space context, the technological miniaturization of satellites enabled a decrease in the costs of developing and launching satellites. The fact that clients and investors are private actors causes a shift in the financial models from cost plus to fixed price. This change requires different methods of management and demands shorter durations of time devoted to Research and Development (R&D). Satellites, systems, and components can now be purchased off the shelf and satellites spend relatively less time in orbit. As a result, project management is more inclined to take risks, to perform technological demonstrations while in service and generally does not aim for 100% success in orbit, as was the case for satellite development under the 'Old Space' ecosystem (Paikowsky 2017).

Among of the most successful results of New Space are the CubeSats. The CubeSat concept was publicly proposed in 2000, with the first CubeSats launched in 2003 (Swartwout 2013). CubeSats are small cuboid-shaped satellites developed around multiples of the basic volume unit of 10 cm × 10cm × 10cm, defined as 1U.

CubeSats opened quicker and cheaper mission opportunities due to the standardization effort done specially with the CubeSat dispenser or deployer, which was gradually being accepted as secondary payload by multiple launch agencies (Chin *et al.* 2008).

VCUB1 reference mission is a 6U demonstration CubeSat under development at the Brazilian space company Visiona Space Technology S.A. The target launch date is foreseen for 2020.

The spacecraft contains two payloads. One is a multispectral optical camera, devoted to remote sensing. The second is a data collection radio based on FPGA, allowing two-way narrow-band data communication. The satellite will be placed in a LEO Sun-Synchronous orbit with an average of 3 daily passes over Brazil (Conto *et al.* 2018).

This mission was targeted for being low-cost, fixed price, using mainly Commercial Off-The-Shelf (COTS) parts. It was conceived as an R&D project to validate in-orbit some company's software technologies. Therefore the project management philosophy is willing to accept some risk. The satellite will be launched as secondary payload, susceptible to the CubeSat launch opportunities available. Consequently, it is a typical New Space mission following the CubeSat standard.

With respect to the mission FSW, the low-cost and short-schedule premises and the best-effort reliability paradigm led the team to look for a well-established open-source FSW framework compatible with CubeSat COTS On-Board Computers (OBC).

REFERENCE MISSION SOFTWARE SELECTION CRITERIA

The criteria for the flight software framework selection are summarized in Table 1, being:

1. Software code and documentation,
2. Flight heritage,
3. Small footprint
4. Quality attributes
5. Long-term support
6. User-community collaboration and
7. The consultative committee for space data systems (CCSDS) standardization.

Criteria 1 to 3 came from the reference mission high-level requirements. Nevertheless, these are typical requirements from space missions, especially under the New Space paradigm.

Quality attributes (criteria 4) were inspired from Wilmot *et al.* (2016). Not all fourteen quality attributes identified by NASA's Software Architecture Review Board were taken, only those more closely related to software implementation. To those quality attributes, it was added Formal Specification and Well-Defined Semantics, inspired in some framework's documentation.

Criteria 5 to 7 purpose is to assess the frameworks capability to evolve and improve based on a variety of missions (User-community collaboration), and possibility of standardization (CCSDS), all that performed by their sponsoring/maintaining institutions (Long-term support).

Table 1. FSW selection criteria.

#	Criteria	Description
1	Software code and documentation	
1.1	Open-source	Software source-code and respective documentation freely available on an internet open code repository, preferably with versioning control
1.2	Technical documentation	Availability of software user-guide and comprehensive technical documentation
1.3	Demo/training	Existence of demonstration applications that provide training and insights on how to use the framework. The provision of a SDK is considered an asset
2	Flight heritage	Flight software framework that has flown successfully on previous spacecraft or instrument missions
3	Small footprint	Minimal memory and CPU usage loads
4	Quality attributes	
4.1	Reliability	Reliability of a software code using by criteria the existence of unit testing with significant code coverage
4.2	Formal specification	Use of architectural rules or requirements to formally specify the software implementation
4.3	Well-defined semantics	Existence of a clear and unambiguous defined behavior
4.4	Requirements traceability	Requirements individually traced to their implementation and to verification evidence
4.5	Modularity (component-based design)	Flight software architecture that emphasizes the separation of concerns by means of loosely coupled independent software components or applications. It favors software code reusability
4.6	Portability (to several RTOS and processors)	A design and implementation property of the architecture and applications supporting their use on systems other than the initial target system. It generally involves software isolation and abstraction techniques
5	Long-term support	Framework software support guaranteed for long-term according to the sponsoring/maintaining institution strategic plan
6	User-community collaboration	Users return of experience expressed in a centralized internet open code repository. Capacity of universally opening ticket issues, forking and uploading software code, all that controlled and moderated by the sponsoring institution
7	CCSDS standardization	The framework has been recognized as expressive by a competent authority (in this case, CCSDS) and is being standardized as a reference flight software architecture

FLIGHT SOFTWARE FRAMEWORKS

There are some software frameworks that have the goal of standardizing, modularizing and decoupling the application layer from the lower layers in space projects. A review on the literature was performed to find a representative set of these architectures and their characteristics. Six state-of-the-art frameworks will be analyzed to choose which one to adopt in the nanosatellite reference mission.

GERICOS

The GEneRIC Onboard Software (GERICOS) framework was developed and qualified by LESIA (*Laboratoire d'Études Spatiales et d'Instrumentation en Astrophysique*), a French laboratory for space research, for the rapid development of payload flight software.

The GERICOS studies started in 2007. At the time there was no off-the-shelf software solution available. At the end of 2015 the Radio Plasma Wave instrument flight software was delivered, built and qualified using GERICOS framework (Plasson *et al.* 2016). This instrument will fly on ESA's Solar Orbiter mission, expected to be launched at 2020.

GERICOS framework offers a set of reusable and customizable flight software components, written in C++. It is composed of 3 layers, as shown in Fig. 1:

- GERICOS::CORE: Lightweight, optimized implementation of the active object paradigm on top of a real-time kernel. It includes the concepts related to real-time and embedded systems, such as interrupts, synchronized objects, shared resources and circular buffers. This layer allows a developer to quickly build a real-time application using the object-oriented approach while being independent from a specific real-time operating system and from a specific hardware target. Each active object has its own message queue and computational thread.
- GERICOS::BLOCKS: offers a set of reusable software components for building flight software based on generic solutions to recurrent functionalities: telecommand management, telemetry management, some of the European Coordination for Space Standardization (ECSS) Packet Utilization Standard (PUS) services, mode management, time management, CCSDS protocol management, etc.
- GERICOS::DRIVERS: implements software drivers corresponding to COTS IP cores used in the chip.

LESIA is working to make possible the use of GERICOS framework by other organizations. One evolution already done for GERICOS::CORE is its port to ARM processors, that allowed GERICOS use in the context of CubeSats, as made on the PicSat mission launched in January 2018 (Lapeyrere *et al.* 2017). The code nevertheless is not yet open-sourced, preventing a more detailed analysis of the framework architecture.

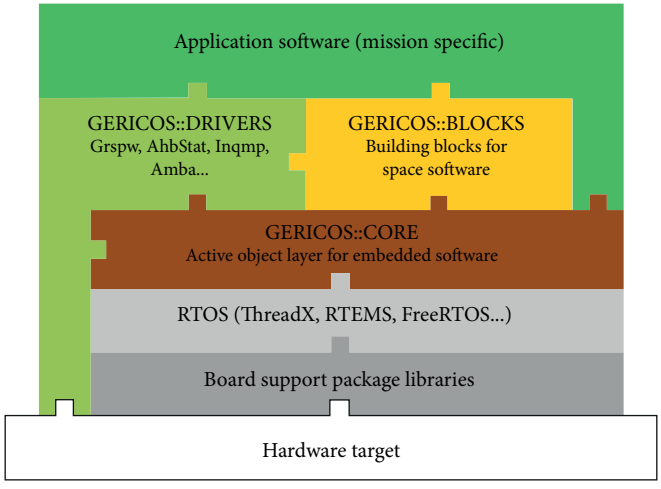

Figure 1. GERICOS software layers (Plasson *et al.* 2016).

NASA CFS

The core Flight System (cFS) platform is a 3-layered flight software architecture developed by NASA Goddard Space Flight Center (GSFC). It provides an application runtime environment independent from both operating system and hardware. cFS was originally developed as NASA Class B software for GSFC science missions, based on the heritage of successful missions. It is composed of three layers, as shown in Fig. 2, from bottom to top:

- Abstraction library layer, composed of:
  - Operating System Abstraction Layer (OSAL): small software library that isolates the flight software from the real time operating system
  - Platform Support Package (PSP): software that is needed to adapt the cFE core to a particular processor architecture
- cFE core layer, composed of:
  - Executive Services (ES): manages cFE core and cFS applications
  - Event Services (EVS): provides an interface for sending asynchronous debug, informational, or error message telemetry
  - Software Bus (SB): provides a portable inter-application message service
  - Table Services (TBL): manages all cFS table images
  - Time Services (TIME): manages spacecraft internal time and distributes tone signal to querying applications

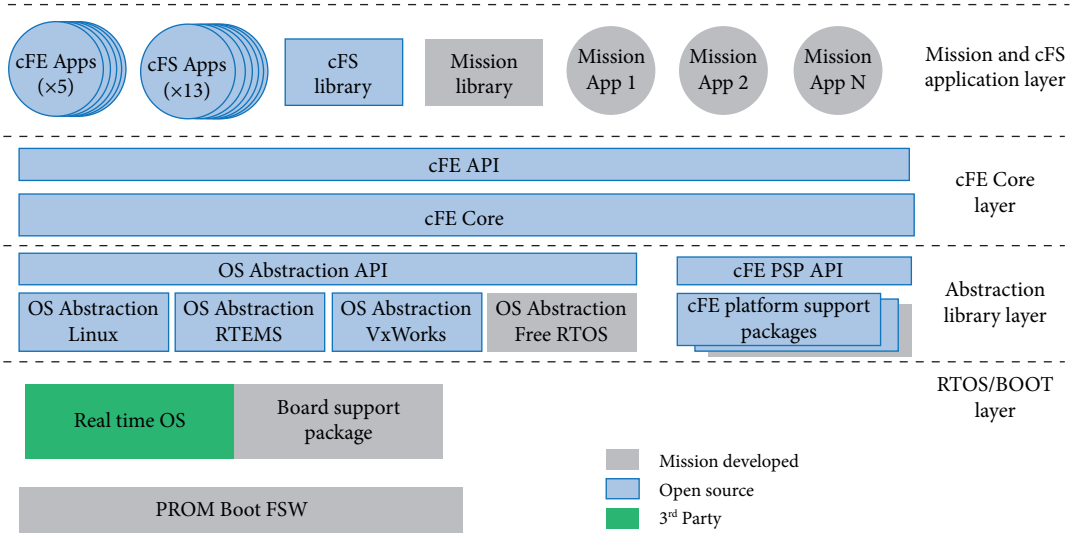

Figure 2. NASA cFS Software Layers and Components. Source: NASA, 2014.

- Mission and cFS application layers: composed of reusable and mission-specific applications. NASA has currently 13 cFS applications distributed as open-source (NASA 2017) that implement common spacecraft functionality. Additionally, a new mission has to define and implement its mission-specific functionalities by means of cFS applications.

The cFS components were incrementally developed and publicly released. The core Flight Executive (cFE) and the platform abstraction layer were first used on the Lunar Reconnaissance Orbiter (LRO), launched in 2009. The initial suite of cFS applications was first used on the Global Precipitation Measurement (GPM) spacecraft launched in 2014. The entire cFS software suite was released as open source in the beginning of 2015 (Core Flight System 2017).

The CCSDS application support services working group is currently in charge of conducting a project called "NASA cFS as a CCSDS Onboard Reference Architecture". This initiative highlights the importance of cFS. The project code and documentation is available at NASA's GitHub page (https://github.com/nasa/cFE) and there is also a community page (http://coreflightsystem.org/) for getting help about cFS and questions and answers.

The European Space Agency (ESA) launched a space standardization initiative called Space AVionics Open Interface aRchitecture, a.k.a. SAVOIR, which formally started in November 2008, as an outcome of ESA's yearly workshop called ADCSS (Avionics Data, Control and Software Systems) (SAVOIR 2018).

One of SAVOIR goals is to identify the main avionics functions and to standardize the interfaces between them, in a way that allows building blocks to be developed and reused across projects (Terraillon 2012). Among the several working groups of ESA's SAVOIR initiative, there was the SAVOIR-FAIRE group, which is in charge of the on-board software reference architecture.

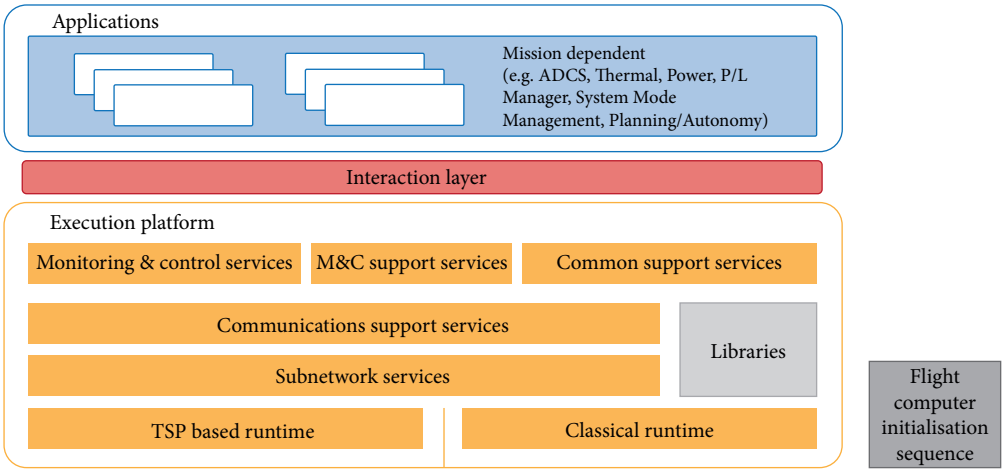

Figure 3. SAVOIR-COrDeT software reference architecture (SAVOIR 2018).

As part of this effort, an architecture was proposed (named Component Oriented Development Techniques – COrDeT) based on the segregation of the application software and the execution platform (Terraillon 2012). Figure 3 shows a static representation of the COrDeT reference architecture.

COrDeT output from SAVOIR group was a reference architecture with no particular implementation and API rules. Nevertheless, P&P software, one of the companies of the consortium funded by ESA to specify COrDeT, has provided a formal specification of the framework and a C-language implementation of COrDeT called C2, which is publicly available at the company's webpage (P&P software 2018).

C2 implementation of COrDeT is a service-oriented application with strong correspondence to ECSS PUS defined services. C2 implementation is being used in ESA's CHEOPS satellite payload software, which will be flying in 2019 (P&P software 2018).

The CCSDS application support services working group is in charge of conducting a project called "SAVOIR as a CCSDS Onboard Reference Architecture", expected to finish by July 2020.

KUBOS

Kubos is a space software start-up company located in Texas, USA, founded in 2014. The company developed the KubOS flight software framework, which has BSP for 3 major CubeSat OBC's. Its full package is open-source. Users can pay for additional features, like support services and software updates (kubos 2018).

KubOS stack is shown in Fig. 4. Its layers are:

- Kubos Linux: a custom Linux distribution designed with embedded devices in mind. It focuses on including only drivers that are useful for space applications (eg. I2C and SPI, rather than display drivers), multi-layer system validation and recovery logic. Kubos Linux projects are built into binaries, which will run as Linux user space applications
- Kubos services: a Kubos service is defined as any persistent process used to interact with the satellite. Services rarely make decisions but will allow the user to accomplish typical flight software tasks such as telemetry storage, file management, shell access, and hardware interaction

- Mission applications: mission applications compose the upper layer and are virtually anything that governs the satellite's behavior. For example, they govern state management, accomplish scripted tasks and monitor onboard behavior. Each

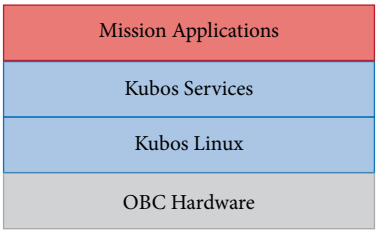

Figure 4. KubOS stack (Kubos 2018).

application is typically dedicated to a certain mode or isolated task the satellite is supposed to accomplish to keep them lightweight and portable. They can be simple, such as a telemetry beacon application, or complex, such as a payload operations application (Kubos 2018).

KubOS is open-sourced in a GitHub repository. First time users can create its own KubOS project using a SDK provided by the company. It was noticed that KubOS is light-weighted and provides the basis for writing mission applications using Rust or Python languages. C and Lua are also claimed to be supported (Kubos 2018).

CAST SOFTWARE REFERENCE ARCHITECTURE

China Academy of Space Technology (CAST) is the main spacecraft development and production facility in China. Xiongwen *et al.* (2015) pointed out that the problem of spacecraft avionics software reuse in space projects is an important issue to be solved in Chinese programs.

A spacecraft avionics software architecture has been designed and implemented by CAST technicians. It is based on Spacecraft Onboard Interface Services (SOIS) recommended by CCSDS and Packet Utilization Standard (PUS) recommended by European Cooperation for Space Standardization (ECSS) standards.

According to Xiongwen *et al.* (2015), the results of the test and validation demonstrate that the software architecture has great positive effects on software reuse. Under Chinese National Space Agency (CNSA) lead, this effort is gaining CCSDS standardization status through a dedicated working group, which also happened with NASA cFS and ESA SAVOIR.

Figure 5 shows CAST software reference architecture. This reference architecture is composed of:

- Operating system layer: the interface of operating system is encapsulated and a uniform Application Program Interface (API) is provided by the operating system layer. Any operating system that supports this interface can be used in the avionics system
- Middleware: common service platform between the operating system layer and application layer. It has standard program interface and protocols, and can realize the data exchange and cross support among different hardware and operating system
- Application layer: contains most of the common functions of avionics system. The implementation of this layer may be different for different projects.

Nonetheless, CAST source-code architecture is not available as open-source. No mention was found regarding any flight heritage.

NANOSAT MO FRAMEWORK (NMF)

The NanoSat Mission Operations service Framework (NMF) is a flight software project that will be flying on an experimental ESA sponsored nanosatellite called OPS-SAT (target launch date in mid-2019). This mission is intended to demonstrate the improvements in mission control capabilities that will arise when satellites can fly with more powerful on-board computers.

NMF provides a standard on-board software framework for nanosatellites based on the CCSDS MO framework (CCSDS 520.0-G-3). It facilitates not only the monitoring and control of the nanosatellite software applications, but also the interaction

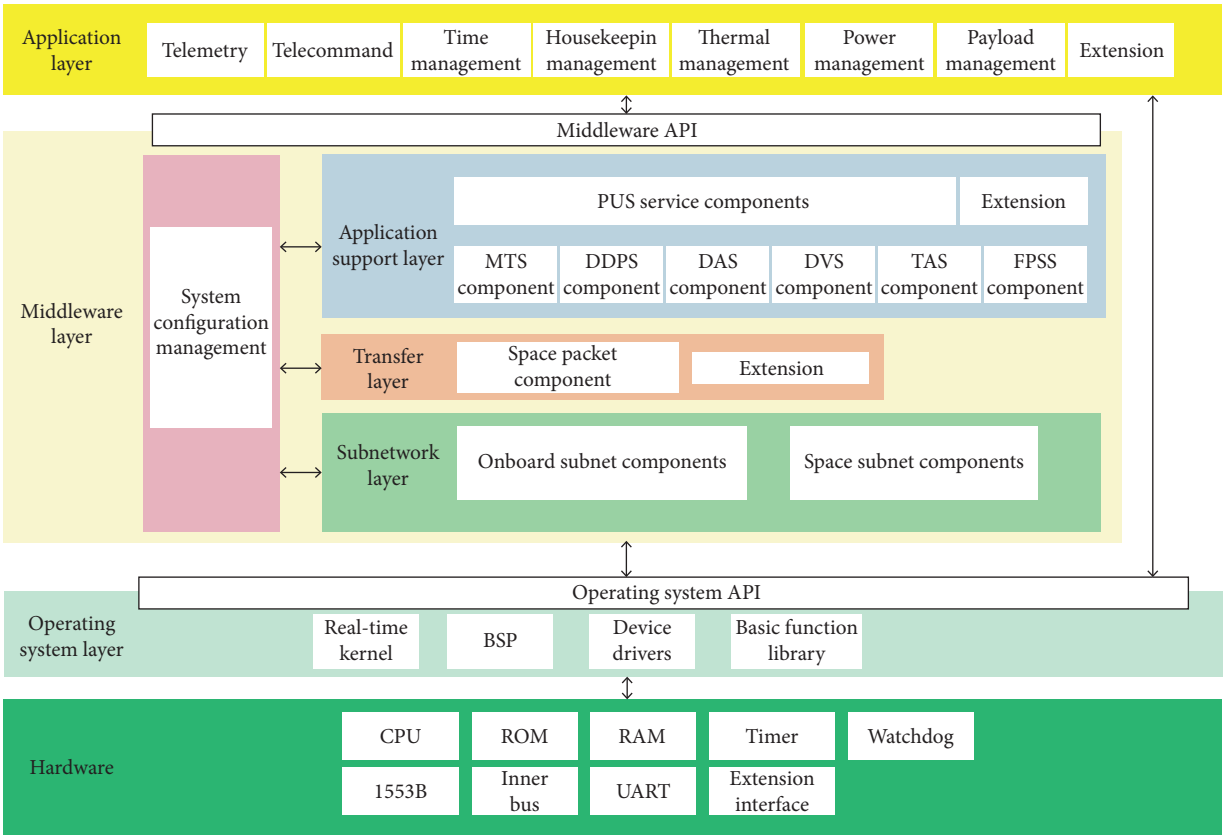

Figure 5. CAST software reference architecture (CCSDS 2016).

with the platform peripherals. This is achieved by using the CCSDS mission operations monitor and control services included in the MO service suite and by defining a set of new platform services (Coelho *et al.* 2016).

OPS-SAT spacecraft provides an experimental platform composed of a MityARM device with 1 GB of RAM, an ARM processor with 925 MHz and a lightweight version of Linux (Coelho 2016). The increase of computing power, a trend for CubeSat projects, allowed the NMF framework to shift from usual embedded C or C++ lightweight software to Java multi-platform agnostic applications.

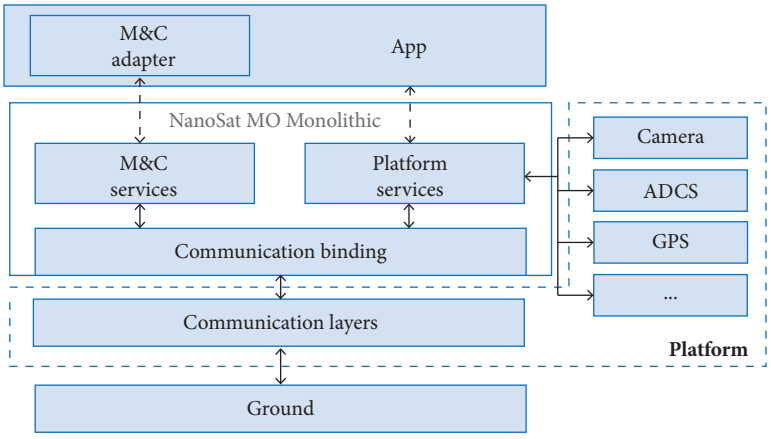

Figure 6. NanoSat MO framework architecture (Coelho 2016).

Figure 6 presents NMF architecture with one application only. For deploying multiple applications that can be concurrent in the use of peripherals, NMF proposed the use of two new components. One of them is the NanoSat MO Supervisor, responsible for managing the applications and providing platform services that can be used by different applications. The other is the NanoSat MO connector, responsible for connecting to the platform services, exposing them to the logic of the applications and interfaces to monitor and control the application from ground or from other applications.

Coelho *et al.* (2016) mentioned that the main difference between NMF and classical FSW frameworks is that NMF is developed for systems that are not scarce in resources but can run complete operating systems. NMF framework is open-sourced in a GitHub public repository and comes along with an SDK, OPS-SAT satellite simulator, and technical documentation for running the software.

RESULTS

Table 2 presents the frameworks comparison study. The conformance to each selection criteria presented in previous section was individually assessed. Some of the frameworks (GERICOS and CAST) are not open-source yet, being available only to their developing organizations, making difficult further technical analysis over them.

Among the open-source frameworks (NASA cFS, COrDeT C2, NanoSat MO and KubOS), the amount of documentation each one provides is different. COrDeT, for example, traces individually each software requirement to the code and to verification evidence, while NanoSat MO relies mostly on the CCSDS 520.0-G-3 documentation that inspired the framework creation.

Regarding semantics, GERICOS and COrDeT are the only ones that created their own semantics, fully specifying their architecture behavior. The others have their syntax spread over the documentation and sometimes one has to look into the code to get the correct comprehension about the software architectures.

NASA cFS and GERICOS are the frameworks that currently have successfully flown on previous missions. The others will acquire flight heritage in the next few years.

A comparison on each software SDK reveals that cFS OpenSatKit is the most "turn-key" solution. It provides not only a running instance of cFS bundle but also an open-source ground control software tailored to interface with cFS and a CubeSat simulator example, providing great insights on the framework functionality.

The CCSDS application support services working group is currently conducting some projects called "CCSDS Onboard Reference Architecture" for three particular frameworks: SAVOIR, CAST and NASA cFS. This evidence shows that these architectures have gained particular recognition by CCSDS due to their technical quality and considerable utilization. This standardization effort will also contribute to their long-term perpetuation and to spread their concepts to other international organizations.

A comprehensive analysis on Table 2 leads to three frameworks that are compliant with most of the selection requirements: SAVOIR/COrDeT, KubOS and cFS.

SAVOIR/COrDeT is the most complete one with respect to software documentation and syntax definition. But it has no flight heritage and depends on a non-defined message-passing middleware to run. Therefore, it could not be immediately used.

KubOS is an architecture conceived specifically for the CubeSat world, as its name suggests. In terms of code footprint and portability, it is a very suitable architecture with a turn-key software package. However, besides its absence of flight heritage, the reference mission is also looking for a framework with assured long-term support. Some ways to ensure that are by public-access standards (CCSDS is an accredited agency) and the existence of an established software control board committee.

Regarding software quality and long-term technical support, cFS is managed by a dedicated NASA configuration control board that controls and approves future changes to cFS core and main applications. Software core maintenance and evolution is then done by NASA, which is also the main user of cFS framework. The long-term support criterion is key for space organizations that want to rely on a third-party framework.

According to the results presented in the frameworks comparison (Table 2) and the criteria defined in previous section, NASA cFS was the most compliant framework to the nanosatellite reference mission needs and constraints. It was therefore adopted by VCUB1 mission as design baseline.

Table 2. FSW framework comparison table.

Criteria		cFS	COrDeT/SAVOIR (through C2)	CAST	GERICOS	KubOS	NanoSat MO
1.1	Open-source	Yes (GitHub)	Yes (C2 implementation, in GitHub)	No	No	Yes (GitHub)	Yes (GitHub)
1.2	Technical documentation	Yes	Yes	No	No	Yes	Yes
1.3	Demo/training	Yes (OpenSatKit)	Yes (there is a demo in C2)	No	No	Yes (KubOS SDK)	Yes (NMF SDK)
2	Flight heritage	Yes	No (ESA's CHEOPS to be launched in 2019)	No	Yes (PicSat launched in Jan 2018)	No (TBC)	No (ESA's Ops-Sat to be launched in 2019)
3	Small footprint	Yes	Yes	Yes	Yes	Yes	No (Java implementation requires a powerful OBC)
4.1	Reliability	Yes	Yes	Not found	Not found	Yes	Not found
4.2	Formal specification	Partially compliant (no centralized cFS architectural rules document. Rules are spread through documentation)	Yes (The CORDET framework C2 implementation - user requirements)	Partially compliant (framework claims to be fully compliant with CCSDS SOIS and ECSS PUS)	Partially compliant (framework claims to be compliant with some ECSS PUS services)	Partially compliant (no KubOS architectural rules document. Rules are found through API's documentation)	Partially compliant (no centralized NMF architectural rules document. But, framework claims to be fully compliant with CCSDS 520.0-G-3)
4.3	Well-defined semantics	Partially compliant (there is no semantics model or document. There is a cFS sample app which contains the basic API calls and a developers user's guide)	Yes (CORDET C1 framework profile)	Not found	Yes (GERICOS UML Profile)	Partially compliant (there is no semantics model or document. There are some template Mission applications which contains the basic KubOS API calls)	Partially compliant (there is no semantics model or document. There is a NMF app development guide)
4.4	Requirements traceability	Partially compliant (cFE and cFS suite requirements exist, but are not traced to the code)	Yes (documented in the CORDET framework C2 implementation - user requirements)	Not found	Not found	Not found	Partially compliant (a very short list of NMF software requirements exist, but are not traced to the code)
4.5	Modularity (component-based design)	Yes	Yes	Not found	Yes	Yes	Yes
4.6	Portability (to several RTOS and processor cards)	Yes (cFS OSAL allows that)	N/A (C2 implementation goes on top of a non-defined message-passing middleware)	Yes (according to the CCSDS CAST architecture documentation says so)	Yes (GERICOS::CORE allows that)	Yes (portability to FreeRTOS, Linux and to some CubeSat OBC's)	Yes (due to Java capabilities)
5	Long-term support	Yes (NASA support)	Yes (ESA support)	Yes (CAST support)	Yes (LESIA support)	Yes (KubOS company support)	Not found (no explicit mention in ESA's documentation)
6	User-community collaboration	Yes (through GitHub tickets)	Yes (because CORDET C2 implementation is in GitHub)	No	No	Yes (through GitHub tickets)	Yes (through GitHub tickets)
7	CCSDS standardization	Yes	Yes	Yes	No	No	Partially (Not standardized as a reference architecture by specific working group, but it is a spin-off of CCSDS 520.0-G-3 standard)

CONCLUSIONS AND FUTURE WORK

Six relevant FSW frameworks were analyzed with respect the nanosatellite reference mission selection criteria. The authors' CubeSat mission can be considered a typical New Space project due to low-cost, short-schedule, management and reliability philosophy and use of COTS parts characteristics. The solution adopted after careful compliance analysis was NASA cFS.

Many software architectures elaborated by the academia were found by the authors during the literature research. These solutions were often developed around a single nano or small satellite project in their respective universities. These architectures, for example Switerzland EPFL' BIP framework (Mavridou *et al.* 2016) and US Vermont' CubedOS (Vermont Technical College 2017), despite interesting, were not analyzed due to the decision to emphasize frameworks with potential reusability across several projects. These solutions can though be explored in a future work.

Commercial solutions, such as Terma's Oboss (Terma 2004) and Scisys's flight software (Scisys 2018), despite flight proven, were not available for further analysis.

It is worth acknowledging that two frameworks were recommended to the authors consideration too late to be cogitated for the paper: RODOS and F Prime. The German "Realtime Object-Oriented Distributed Operating System" (RODOS) (Dannemann and Montenegro 2013) contains an object-oriented middleware that is an interesting open-source framework for space applications. NASA JPL "F Prime" framework was recently released. It states some differences compared to cFS. For example, F Prime architecture uses static topologies with typed connections, whereas cFS uses publish-subscribe over a software bus (Bocchino Jr. *et al.* 2018). RODOS and F Prime will be explored in a future work.

AUTHOR'S CONTRIBUTION

Conceptualization, Miranda DJF, Ferreira M and Kucinskis F; Methodology, Miranda DJF, Ferreira M, Kucinskis F and McComas D; Investigation, Miranda DJF and McComas D; Writing – Original Draft, Miranda DJF; Writing – Review and Editing, Miranda DJF, Ferreira M, Kucinskis F and McComas D; Supervision, Ferreira M and Kucinskis F.

FUNDING

There are no funders to report for this submission.

REFERENCES

Bocchino Jr. RL, Canham TK, Watney GJ, Reder LJ, Levison JW (2018) F Prime: an open-source framework for small-scale flight software systems. Presented at: 32nd Annual AIAA/USU Conference on Small Satellites; Logan, USA.

CCSDS (2016) CAST flight software as a CCSDS OnBoard reference architecture. Washington: CCSDS.

Chin A, Coelho R, Brooks L, Nugent R, Puig-Suari J (2008) Standardization promotes flexibility: a review of CubeSats' success. Presented at: 6th Responsive Space Conference; Los Angeles USA.

Coelho C, Koudelka O, Merri M (2016). NanoSat MO framework: achieving on-board software portability. Presented at: SpaceOps 2016 Conference; Daejeon, Korea. https://doi.org/10.2514/6.2016-2624

Conto A, Mattei AP, Saquis-Sannes P, Carvalho H, Miranda D, Balbino F (2018) Use of SysML and model-based system engineering in the development of the Brazilian satellite VCUB1. Presented at: 10th European Cubesat Symposium; Toulouse, France.

Core Flight System (2017) About the technology and why cFS. Core Flight System; [accessed 2018 May 18]. http://coreflightsystem.org/why-cfs/

Dannemann F, Montenegro S (2013) Embedded logging framework for spacecrafts. Presented at: DASIA 2013 DAta Systems In Aerospace; Porto, Portugal.

Dvorak DL (2009) NASA study on flight software complexity. Presented at: AIAA Infotech@Aerospace Conference; Seatle, USA. https://doi.org/10.2514/6.2009-1882

Eickhoff J (2012) Onboard computers, onboard software and satellite operations: an introduction. Berlin: Springer.

Kubos (2018) KubOS; [accessed 2018 July 10]. https://docs.kubos.com/1.5.0/index.html

Kucinskis NF, Ferreira VGM (2013) On-board satellite software architecture for the goal-based Brazilian mission operations. IEEE Aerospace and Electronic Systems Magazine 28(8):32-45. https://doi.org/10.1109/MAES.2013.6575409

Lapeyrere V, Lacour S, David L, Nowak M, Crouzier A, Schworer G, Perrot P, Rayane S (2017) PicSat: a Cubesat mission for exoplanetary transit detection in 2017. Presented at: 31st Annual AIAA/USU Conference on Small Satellites; Logan, USA.

Mavridou A, Stachtiari E, Bliudze S, Ivanov A, Katsaros P, Sifakis J (2016) Architecture-based design: a satellite on-board software case study. In: Kouchnarenko O, Khosravi R, editors. Formal Aspects of Component Software. FACS 2016. Cham: Springer. https://doi.org/10.1007/978-3-319-57666-4_16

NASA (2014) core Flight System (cFS) Background and Overview. NASA; [accessed 2017 February 5]. https://cfs.gsfc.nasa.gov/cFS-OviewBGSlideDeck-ExportControl-Final.pdf

NASA (2017) NASA software catalog 2017-2018. NASA; [accessed 2017 November 25]. https://software.nasa.gov/ NASA_Software_Catalog_2017-18.pdf

Paikowsky, D (2017) What is new space? The changing ecosystem of global space activity. New Space 5(2):84-88. https://doi.org/10.1089/space.2016.0027

Plasson P, Cuomo C, Gabriel G, Gauthier N; Gueguen L, Malac-Allain L (2016) GERICOS: A Generic Framework for the development of on-board software. Presented at: DASIA 2016. Data Systems in Aerospace; Tallin, Estonia.

P&P Software (2018) CORDET C2 Implementation Download. P&P Software; [accessed 2018 May 1]. https://www.pnp-software.com/cordetfw/download.html

Rexroat JT (2014) Proposed Middleware Solution for Resource-Constrained Distributed Embedded Networks (Master's Dissertation). Lexington: University of Kentucky.

SAVOIR (2018) SAVOIR Outputs. SAVOIR; [accessed 2018 May 1]. http://savoir.estec.esa.int/SAVOIROutput.htm

Schmidt DC, Gokhale A, Natarajan B (2004) Leveraging application frameworks: why frameworks are important and how to apply them effectively. ACM Queue 2(5):66. https://doi.org/10.1145/1016998.1017005

Scisys (2018) SCISYS flight software brochure. Scisys; [accessed 2018 May 1]. https://www.scisys.co.uk/fileadmin/user_upload/Downloads/Operational_Divisions/Space/SCISYS-Space-Flight-Software-2018.pdf

Swartwout M (2013) The first one hundred CubeSats: a statistical look. Journal of Small Satellites 2(2):213-233.

Terma (2004) Onboard Operations Support Software homepage. Terma; [accessed 2018 May 1]. http://spd-web.terma.com/Projects/OBOSS/Home_Page/

Terraillon JL (2012) SAVOIR: Reusing specifications to improve the way we deliver avionics. Presented at: Embedded Real Time Software and Systems; Toulouse, France.

Vermont Technical College (2017) CubedOS project overview. Vermont Technical College CubeSat Laboratory; [accessed 2018 May 1]. http://cubesatlab.org/CubedOS.jsp

Wilmot J, Fesq L, Dvorak D (2016) Quality attributes for mission flight software: a reference for architects. Presented at: IEEE Aerospace Conference; Big Sky, USA. https://doi.org/10.1109/AERO.2016.7500850

Xiongwen H, Bowen C, Dong Y, Jianbing Z, Ming G (2015) Design and implementation of spacecraft avionics software architecture based on spacecraft onboard interface services and packet utilization standard. Presented at: International Astronautical Congress; Jerusalem, Israel.