MINISTÉRIO DA CIÊNCIA, TECNOLOGIA, INOVAÇÕES E COMUNICAÇÕES
INSTITUTO NACIONAL DE PESQUISAS ESPACIAIS

sid.inpe.br/mtc-m21c/2019/12.16.01.24-TDI

A "NEW SPACE" APPROACH ON SPACECRAFT FLIGHT SOFTWARE DEVELOPMENT USING NASA CFS FRAMEWORK

Danilo José Franzim Miranda

Master's Dissertation of the Graduate Course in Engineering and Space Technology/Space Systems Engineering and Management, guided by Drs. Maurício Gonçalves Vieira Ferreira, and Fabrício de Novaes Kucinskis, approved in December 13, 2019.

URL of the original document:
<http://urlib.net/8JMKD3MGP3W34R/3UJDK9L>

INPE
São José dos Campos
2019

PUBLISHED BY:

Instituto Nacional de Pesquisas Espaciais - INPE
Gabinete do Diretor (GBDIR)
Serviço de Informação e Documentação (SESID)
CEP 12.227-010
São José dos Campos - SP - Brasil
Tel.:(012) 3208-6923/7348
E-mail: pubtc@inpe.br

MINISTÉRIO DA CIÊNCIA, TECNOLOGIA, INOVAÇÕES E COMUNICAÇÕES
INSTITUTO NACIONAL DE PESQUISAS ESPACIAIS

sid.inpe.br/mtc-m21c/2019/12.16.01.24-TDI

A "NEW SPACE" APPROACH ON SPACECRAFT FLIGHT SOFTWARE DEVELOPMENT USING NASA CFS FRAMEWORK

Danilo José Franzim Miranda

Master's Dissertation of the Graduate Course in Engineering and Space Technology/Space Systems Engineering and Management, guided by Drs. Maurício Gonçalves Vieira Ferreira, and Fabrício de Novaes Kucinskis, approved in December 13, 2019.

URL of the original document:
<http://urlib.net/8JMKD3MGP3W34R/3UJDK9L>

INPE
São José dos Campos
2019

Cataloging in Publication Data

Miranda, Danilo José Franzim.

M672n A “new space” approach on spacecraft flight software development using NASA cFS framework / Danilo José Franzim Miranda. – São José dos Campos : INPE, 2019.

xxviii + 150 p. ; (sid.inpe.br/mtc-m21c/2019/12.16.01.24-TDI)

Dissertation (Master in Engineering and Space Technology/Space Systems Engineering and Management) – Instituto Nacional de Pesquisas Espaciais, São José dos Campos, 2019.

Guiding : Drs. Maurício Gonçalves Vieira Ferreira, and Fabrício de Novaes Kucinskis.

1. Flight Software. 2. New Space. 3. NASA cFS. 4. Software Frameworks. 5. Software Development Approach. I.Title.

CDU 629.78:004.41

Aluno (a): *Danilo José Franzim Miranda*

Título: "A "NEW SPACE" APPROACH ON SPACECRAFT FLIGHT SOFTWARE DEVELOPMENT USING NASA cFS FRAMEWORK"

Aprovado (a) pela Banca Examinadora em cumprimento ao requisito exigido para obtenção do Título de *Mestre* em

Engenharia e Tecnologia Espaciais/Eng. Gerenc. de Sistemas Espaciais

Dr. Walter Abrahão dos Santos

Presidente / INPE / São José dos Campos - SP

( ) Participação por Video - Conferência

(x) Aprovado *( ) Reprovado*

Dr. Maurício Gonçalves Vieira Ferreira

Orientador(a) / INPE / SJCampos - SP

( ) Participação por Video - Conferência

(x) Aprovado *( ) Reprovado*

Dr. Fabrício de Novaes Kucinskis

Orientador(a) / INPE / São José dos Campos - SP

( ) Participação por Video - Conferência

(x) Aprovado *( ) Reprovado*

Dr. Ronaldo Arias

Membro da Banca / INPE / São José dos Campos - SP

( ) Participação por Video - Conferência

(x) Aprovado *( ) Reprovado*

Dr. Rodrigo Rocha Silva

Convidado(a) / FATEC / Mogi das Cruzes - SP

( ) Participação por Video - Conferência

(x) Aprovado *( ) Reprovado*

Este trabalho foi aprovado por:

( ) maioria simples

(x) unanimidade

São José dos Campos, 13 de dezembro de 2019

"We are like a pencil with which God writes the texts He wants spoken in the hearts of men"

St. Dulce of the Poor

Às minhas amadas esposa Beatriz e filha Manuela

ACKNOWLEDGEMENTS

Portuguese / Português*:*

Gostaria de agradecer inicialmente a Deus, nosso Criador e Salvador, pelo dom da vida. Muitas vezes não fui fiel e merecedor, mas as inúmeras bênçãos dadas por gratuidade, amor e misericórdia me concederam chegar até aqui, essa importante etapa de formação e capacitação profissional em minha vida. Muito obrigado, Mestre! Nossa Senhora, amável e admirável intercessora junto a Cristo, obrigado por não me desamparar e deixar esmorecer.

A seguir, gostaria de agradecer à minha esposa, Beatriz, companheira de todas as batalhas da vida. Ela sempre me lembrou da importância de estudar, me ajudou a vencer a preguiça e o desânimo do dia-a-dia e me motivou quando eu hesitava em continuar adiante. Me poupou diversas vezes dos cuidados com a casa e com a neném para que eu pudesse ter mais tempo livre para estudar. Suas orações, amor e carinho certamente me ajudaram a vencer e seu exemplo de dedicação profissional silenciosamente me guiou e inspirou nessa jornada. Te amo muito, meu amor! O fruto do nosso amor, nossa princesa Manuela, veio a este mundo recentemente (durante o final do Mestrado!) para fazer nossa família mais completa e feliz. Espero que ela um dia venha a ter orgulho do Papai, que encontrou no seu olhar inocente e amoroso uma injeção nova de motivação para continuar perseverando profissionalmente.

Aos meus pais e irmão, que estão longe em Brasília, mas pertinho em oração e coração. Frequentemente nos telefonemas e nos encontros presenciais se interessavam em saber sobre o meu Mestrado e enfatizavam a importância de sempre estar estudando e se atualizando para ser um bom profissional e vencer na vida. Obrigado também pelas orações constantes. Deus os abençoe! Amo muito vocês!

Aos meus demais familiares e amigos, tanto em Brasília quanto em São José dos Campos, obrigado pelo carinho, orações e apoio! Em especial meus sogros, cunhados, sobrinhos/afilhados. Também aos meus avós, tios e primos. Vocês são muito queridos.

Aos meus orientadores, Maurício Gonçalves Vieira Ferreira e Fabrício de Novaes Kucinskis, reconhecidos especialistas em tecnologia de software do setor espacial brasileiro, e grandes mestres e seres humanos. Perdi a conta de quantas reuniões ocorreram na sala do prof Maurício, intercaladas por diversas

mensagens de whatsapp e e-mail, além dos muitos comentários e revisões cuidadosamente feitos nas 62 versões que esse trabalho possuiu. Enfim, a intensa troca de informações e conhecimento, e particularmente a paciência, dedicação e atenção dispensadas a mim foram primordiais para que a caminhada valesse a pena e esse Mestrado pudesse ser concluído com êxito. Muito obrigado, professores!

À pós-graduação do INPE, pelo acolhimento e serviço silencioso, porém eficaz, de seu quadro de funcionários, muito obrigado. Agradeço em especial à banca de revisão, composta pelos profs Walter Abrahão (presidente), Ronaldo Arias e Rodrigo Rocha Silva, pelas valiosas contribuições dadas à dissertação durante o exame de proposta de qualificação e também na defesa final.

À Visiona, meu ambiente profissional nos últimos 7 anos, em especial à direção da engenharia e da empresa, nomeadamente Janio Kono, Carlos Santana, Himilcon Carvalho e João Paulo Campos, que me incentivaram e autorizaram a cursar o mestrado, me liberando para comparecer às aulas e reuniões em horário de expediente, e pela confiança que vem depositado em mim. Aos meus colegas da Visiona e das organizações parceiras com quem lado a lado me dediquei nos projetos dos satélites SGDC-1 e VCUB1, obrigado! O enorme desafio técnico por trás do nanossatélite nesses últimos anos, em especial, foi uma das minhas maiores motivações para o engajamento no Mestrado. É uma honra poder trabalhar com vocês!

ABSTRACT

This dissertation was motivated by a real-life problem of developing a Flight Software (FSW) for a commercial nanosatellite mission, heavily constrained in cost and schedule. FSW is a complex subject, demanding a software development team with competencies in embedded systems, real-time systems, spacecraft engineering and spacecraft operations in order to conceive a project. This set of skills is rarely found together, consisting of a great barrier for new entrants. The “New Space” FSW development approach proposed herein consists in four steps: 1) selection and adoption of a FSW framework; 2) compliance assessment of the framework with respect to applicable space software standards; 3) software design rules proposition to better adhere to framework and improve quality; 4) creation of a tool that facilitates the implementation of the aforementioned rules in the software development. NASA cFS was the chosen framework, being the central piece of this work. Despite its considerable heritage and success in several NASA scientific missions and being open source since 2015, cFS is still not widely adopted outside the American space agency. This work also helps filling the lack of academic literature with respect to frameworks employment and their systematic use in “New Space” missions.

Keywords: Flight Software. FSW. OBDH. OBSW. Flight Management Systems. Onboard Data Processing. New Space. NASA cFS. Software Frameworks. Software Development Approach.

UMA ABORDAGEM "NEW SPACE" PARA O DESENVOLVIMENTO DE SOFTWARE DE VOO PARA SATÉLITES UTILIZANDO O FRAMEWORK NASA cFS

RESUMO

A presente dissertação foi motivada por um problema real em desenvolver um software de voo para uma missão nanossatélite comercial, fortemente restringida em custo e cronograma. Software de voo é um tema complexo, demandando um time de desenvolvimento de software com competências nas áreas de sistemas embarcados, sistemas real-time, engenharia de satélites e operações para poder conceber e realizar um projeto nessa área. Esse conhecimento é raramente encontrado reunido, consistindo assim em uma grande barreira para novos entrantes. A abordagem de desenvolvimento de software de voo "New Space" proposta nesse trabalho consiste em quatro etapas: 1) seleção e adoção de um *framework* para software de voo; 2) análise de conformidade do *framework* escolhido com respeito a normas aplicáveis de software para espaço; 3) proposição de regras de projeto de software para aprimorar a aderência ao *framework* e melhorar a qualidade; 4) criação de uma ferramenta que facilite a implementação de tais regras no desenvolvimento de software. O *framework* escolhido foi o NASA cFS, peça chave de estudo do presente trabalho. Apesar de possuir considerável herança de voo e sucesso em diversas missões científicas da NASA e seu código ser liberado para uso público desde 2015, o cFS ainda é pouco adotado fora da agência espacial estado-unidense. Esse trabalho também contribui para preencher a lacuna de literatura acadêmica com respeito ao emprego de *frameworks* e seu uso sistemático em missões "New Space".

Palavras-chave: Software Embarcado. Software de Voo. OBDH. OBSW. FSW. Sistemas de Gestão de Voo. Processamento de Dados a Bordo. New Space. NASA cFS. Frameworks de Software. Abordagem de Desenvolvimento de Software.

FIGURES LIST

Page

TABLES LIST

Page

ABBREVIATIONS AND ACCRONYMS LIST

ADCSS	ESA Workshop on Avionics, Data, Control and Software Systems
AOCS	Attitude and Orbit Control System
AIT	Assembly, Integration and Tests
API	Application Programming Interface
APL	Applied Physics Laboratory
APP	Application (software application)
ASM	Attached Sync Marker (CCSDS context)
ATS	Absolute Time Sequence (cFS context)
AUTOSAR	Automotive Open System Architecture
BCH	Bose–Chaudhuri–Hocquenghem (coding technique)
BSP	Board Support Package
CADU	Channel Access Data Unit (CCSDS context)
CASC	China Aerospace Science and technology Corporation
CAST	China Academy of Space Technology
CBERS	China-Brazil Earth Resources Satellite
CBSE	Component-Based Software Engineering
CCSDS	Consultative Committee of Space Data Systems
CDS	Critical Data Storage (cFE Service)
CERES	Compact Radiation belt Explorer
CF	CFDP App (cFS App)
CFDP	CCSDS File Delivery Protocol
cFE	core Flight Executive
cFS	core Flight System
CI	Command Ingestor (cFS App)
CLCW	Command Link Control Word
CLTU	Communications Link Transmission Unit
CNSA	China National Space Administration
COP-1	Command Operation Procedure-1
COrDeT	Component Oriented Development Techniques

COTS	Commercial Off-The-Shelf
CPU	Central Processing Unit
CRC	Cyclic Redundancy Check
CS	CheckSum (cFS App)
CSP	CHREC Space Processor
CUC	CCSDS Unsegmented time Code
DAS	Device Access Service (CCSDS SOIS)
DDPS	Device Data Pooling Services (CCSDS SOIS)
DDS	Device Discovery Service (CCSDS SOIS)
DES	Device Enumeration Service (CCSDS SOIS)
DR	Design Rule
DS	Data Storage (cFS App)
DTW	Dan Templates Wrapper
DVS	Device Virtualization Service (CCSDS SOIS)
ECSS	European Committee for Space Standardization
EDS	Electronic Data Sheet
EEPROM	Electrically Erasable Programmable Read-Only Memory
EGSE	Electrical Ground Support Equipment
ES	Executive Services (cFE App)
ESA	European Space Agency
EVS	Event Services (cFE App)
FM	File Manager (cFS App)
FPGA	Field Programmable Gate Array
FS	File Services (cFE Context)
FSW	Flight Software *(synonym to OBSW in this work)*
GERICOS	Generic Onboard Software
GFSC	Goddard Space Flight Center (NASA Center)
GMT	Greenwich Meridian Time
GNC	Guidance, Navigation and Control
GPM	Global Precipitation Measurement
HK	HouseKeeping (cFS App)

HS	Health and Safety (H&S)
ID	Identification
IO	Input Output library (cFS Context)
INMS	Ion-Neutral Mass Spectrometer
INPE	*Instituto Nacional de Pesquisas Espaciais*
IP	Internet Protocol
ISO	International Organization for Standardization
ISS	International Space Station
JATM	Journal of Aerospace Technology and Management
KARI	Korean Aerospace Research Institute
LADEE	Lunar Atmosphere and Dust Environment Explorer
LC	Limit Checker (cFS App)
LDCP	Low-Density Parity Check
LEO	Low Earth Orbit
LESIA	*Laboratoire d'Études Spatiales et d'Instrumentation en Astrophysique*
LM	Lockheed Martin
LRO	Lunar Reconnaissance Orbiter
MAL	Message Abstraction Layer
MAP	Multiplexer Access Point (CCSDS context)
MB	Mega Bytes
MD	Memory Dwell (cFS App)
MET	Mission Elapsed Time
MID	Message ID
MM	Memory Manager (cFS App)
MMS	Magnetospheric Multiscale Mission
MO	Mission Operations
N/A	Not Applicable
NASA	National Aeronautics and Space Administration
NMF	NanoSat MO Framework
NOOP	No-Operations command (cFS context)
NOS3	NASA Operational Simulation for Small Satellites
NTNU	Norwegian University of Science and Technology

NUTS	NTNU Test Satellite
OBC	On-Board Computer
OBCP	On-Board Control Procedure
OBSW	OnBoard Software *(synonym to FSW in this work)*
OCF	Operational Control Field
OS	Operating System
OSAL	Operating System Abstraction Layer
OSI	Open Systems Interconnection model
PDU	Protocol Data Unit
PMM	*Plataforma Multi-Missão*
POSIX	Portable Operating System Interface
PSP	Platform Support Package
PUS	Packet Utilization Standard
RAM	Random Access Memory
RBPS	Radiation Belt Storm Probes
RTEMS	Real-Time Executive for Multiprocessor Systems
RTOS	Real-Time Operating System
RTS	Relative Time Sequence (cFS context)
SACI	*Satélite de Aplicações Científicas*
SAVOIR	Space Avionics Open Interface Architecture
SB	Software Bus (cFE App)
SBN	Software Bus Network (cFS App)
SC	Subcommittee (ISO context)
SC	Stored Command (cFS App)
SCD	*Satélite de Coleta de Dados*
SCH	Scheduler (cFS App)
SDK	Software Development Kit
SDLP	Space Data Link Protocol
SDLS	Space Data Link Security Protocol
SDO	Solar Dynamics Observatory
SOA	Service Oriented Architecture
SOIS	Spacecraft Onboard Interface Services

SPARC	Scalable Processor Architecture
SPF	Single Point Failure
SPI	Serial Peripheral Interface
SPL	Software Product Line
SPP	Solar Probe Plus
SRAM	Static RAM
STCF	Spacecraft Time Correlation Factor
STF-1	Simulation-To-Flight 1
TAI	*Temps Atomique International*
TBL	Table Services (cFE App)
TC	Technical Committee (ISO context)
TC	Telecommand (FSW context)
TDM	Time Division Multiplexed
TIME	Time Services (cFE App)
TM	Telemetry
TMTC	Telemetry and Telecommand
TO	Telemetry Output (cFS App)
TRL	Technology Readiness Level
UART	Universal Asynchronous Receiver-Transmitter
UAV	Unmanned Air Vehicle
UDP	User Datagram Protocol
UML	Unified Modeling Language
USA	United States of America
UT	Universal Time
UTC	Coordinated Universal Time
VC	Virtual Channel
VCP	Virtual Channel Package
VCUB1	Visiona Cubesat 1

CONTENTS

Page

1 INTRODUCTION

1.1. **Motivation**

On space applications, the On-Board Software is a software that is embedded in a spacecraft computer, being responsible for managing on-board activities and data processing. It is usually referred to as “Flight Software” (FSW). In such a domain, the level of failure protection and recovery is generally higher than that of usual computer systems due to space environment intrinsic characteristics, and the reduced accessibility after the launch of the space vehicle.

The FSW is, on most cases, responsible for the operation of the space segment, but it hasn’t always been like this.

In a space mission, the responsibility for performing tasks 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, but by adding computers to the space segment, the operators could start delegating tasks to on-board execution (KUCINSKIS, 2013).

With the evolution of spacecraft on-board computer and electronics and with the gain of confidence on autonomous spacecraft operations, more responsibility was given to the FSW. Nowadays missions are increasingly demanding more powerful, autonomous, robust and flexible flight software to meet complex mission requirements.

Due to the flexible nature of software in general, the FSW is usually perceived in a space project as a highly customizable item, which is easier or less costly to change compared to hardware-intensive spacecraft subsystems. This leads to a misconception that FSW can be continuously modified to achieve mission’s objectives and represents no challenge or critical path to a space systems development.

Dvorak (2009) pointed that flight software complexity is growing rapidly, with the number of code lines experimenting exponential growth rate of a factor of 10

approximately every 10 years in NASA missions. To deal with such growth in size and complexity, a software development process is even more necessary now than in previous space missions. However, the cost of a complete software development cycle for every mission is difficult to afford, and a simplification is required to save time and money.

FSW is a complex subject that demands a software development team with competencies in embedded systems, real-time systems, spacecraft engineering and spacecraft operations in order to conceive a flight software project. This set of skills is rarely found together in the literature and in the academia, consisting of a great barrier to start a flight software project from scratch.

It is reasonable to say that this knowledge is mostly present in established space companies, agencies and space research institutes who have already invested considerable time and human resources on this endeavor. New entrants to this field often have to find their own way and dig deeply into the subject to conceive a software product.

In Brazil for example, the Brazilian National Institute for Space Research (INPE), founded in 1961, has great competence in FSW development, having fully developed or participated on the development of FSW for the *Satélite de Coleta de Dados* (SCD), China-Brazil Earth Resources Satellite (CBERS), *Satélite de Aplicações Científicas* (SACI) and the *Plataforma Multimissão* (PMM) satellites series, with the first being Amazonia-1, currently on its final Assembly, Intagration and Testing (AIT) phases.

Nevertheless, all mentioned missions had FSW specifically designed for a given hardware and a given operating system. This is also true for most space missions to date.

Having said that, there are few multi-mission software product line approaches in the space industry, which could be used in order to reduce the overall development effort.

According to Schmidt et al. (2004), 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 engineering

organizations, and that's what motivated this work engagement on space frameworks research, evaluation and application.

After FSW framework selection, despite its support in software design, new users often spend a considerable amount of time on its learning, understanding and finally adoption in a space mission. Schmidt et al. (2004) states that in order to use a framework effectively, one must evaluate whether the time spent learning it outweighs the time saved by software reuse.

This issue motivated this research engagement on evaluating FSW frameworks suitability for a "New Space" mission, and finally simplifying the use of the chosen framework by means of a systematic development of software applications.

1.2. **Research objectives**

The main objective of this dissertation is to provide an approach for FSW development using a suitable framework, aiming to address the demands of "New Space" missions.

This main objective is achieved through the accomplishment of the following steps:

1. Selection and adoption of a FSW framework;
2. Compliance assessment of the framework with respect to applicable space software standards;
3. Software design rules proposition to better adhere to framework and improve quality;
4. Creation of a tool that facilitates the implementation of the aforementioned rules in the software development.

1.3. **Methodology**

Six relevant FSW frameworks were analyzed according to criteria typically required in what is being called *"New Space"* missions.

"New Space" in this dissertation is understood as a paradigm shift in space engineering, especially involving nanosatellites, in which reliability, conformance to standards, quality assurance and some industrial aspects are simplified aiming at costs saving.

After careful consideration and according to the selected criteria, NASA cFS was chosen as the flight software framework in this research reference mission.

According to Core Flight System (2017), "cFS initial conception started with a team of NASA senior engineers that performed a structured heritage analysis of missions covering more than a decade. The diversity of the heritage missions (single vs. redundant components, varying orbits, different operational communication scenarios, etc.) provided valuable insights into what drove FSW commonality and variability across the missions. The team took the entire FSW life cycle into consideration, including in-orbit FSW sustaining engineering, as they performed their analysis. They identified system and application level variation points to address the range and scope of the flight systems domain. A primary goal was to enable portability across embedded computing platforms and to implement different end-user functional needs without the need to modify the source code".

The present work performed a careful analysis on the FSW design of a Brazilian "New Space" mission, that finally adopted cFS as the design framework through all the spacecraft design.

This use case happened during the time frame between 2016 and 2019, in Visiona Space Technology, a Brazilian space systems company located in São José dos Campos city, state of São Paulo. The reference mission is an In-Orbit demonstration CubeSat with two payloads, an optical camera and a data collection radio. It will be placed in a LEO Sun-Synchronous orbit, with an average of 3 daily passes over Brazil.

The reference spacecraft is using a CubeSat COTS OBC and FSW based on NASA cFS v6.5.0 open-source version. This use case will be referenced along

the project and constitute the main experimentation that justifies most part of the conclusions of this work.

Software standards like CCSDS SOIS (Spacecraft Onboard Interface Services) and ECSS PUS (Packet Utilization Standard) establish a minimum number of services that might be implemented on the flight software to allow successful spacecraft operation.

These norms constitute an aggregate of space agencies and institutes experience and bring valuable heritage and lessons learned to new missions. Because of that, the author has carefully studied these standards before analyzing the reference mission FSW project.

It is important to emphasize that there is no contradiction in partially or entirely adopting space standards and being a “New Space” mission. Quality is still an important subject and using standards may not represent prohibitive costs, especially if the software framework already implements some of the standards’ proposed functionalities.

cFS is a framework with considerable flight heritage at NASA Goddard and other institutions. It is not a coincidence that it implements several of the services specified in CCSDS SOIS and ECSS PUS on its core functions. The rigorous use of cFS rules on mission-created applications is therefore a good indication of compliance with space standards.

Nevertheless, one of the immediate results from the reference mission analysis is that even though cFS, on its open-source version, is a consistent and validated architecture and software product line, its adoption was not straight-forward at all.

Among other tasks, the recurrent “handcrafted” confection of high-level cFS-compliant applications led to the occurrence of basic errors such as typos, absence of cFE mandatory API’s declarations and wrong use of cFE/cFS functions. Correcting these mistakes and associated troubleshooting, in order to guarantee cFS rules compliance, was a time-consuming activity in the reference mission and could certainly be optimized.

Some root causes for these issues are: different developers might code the same cFS-application differently; the developer might not know cFS deeply

enough; different applications need slightly different API's and function calls; and the absence of an universal customizable cFS application template.

To deal with such problems, this dissertation proposes a systematic approach for cFS applications development, consisting of design rules creation, adoption and the use of a corresponding cFS template generator.

This template generator can create customized application files according to a few pre-defined application types, all of them implementing the proposed rules accordingly. The design rules and the respective tool are validated through end-to-end testing with runtime FSW and a ground control system in a simulated environment.

1.4. **Expected contributions**

The expected contribution of this work is to establish a software development strategy that can be applied in the space industry and academia, saving money and time for space organizations. It was conceived to be replicable by means of using only open-source software and openly available literature and standards.

The comparison among software frameworks constitutes a literature survey that might be an interesting starting point for other FSW projects, especially those in initial phases. Different selection criteria might alter the framework of choice for other missions or, in the case of similar missions, cFS selection rationale can simply refer to the analysis performed herein.

cFS compliance with space software standards is an analysis that can be inherited by other cFS-based missions because essentially the study was performed over the product line core, which is common to every cFS-based mission. Some non-conformances to the standards were observed, duly noted, and might be interesting points for framework enhancement.

The “Dan Templates Wrapper” developed tool code is open-source and is hosted in the author’s GitHub page[1]. It was developed using NASA’s OpenSatKit SDK environment. Besides the code, the proposed design requirements for new cFS applications can also be useful for the framework users.

In conclusion, the approach described in this dissertation may be particularly interesting for low-cost space missions, which can’t afford purchasing commercial solutions for the flight and ground software. Hiring a team of software experts to develop from scratch a customized solution might not also be a feasible option. This was the case for Visiona’s reference mission but certainly is also a latent need in the “New Space” paradigm.

1.5. **Document structure**

The dissertation is organized through chapters, each one hosting the following content:

- Chapter 1 introduces the present work, listing the motivation, objectives, methodology and the expected contributions.
- Chapter 2 presents VCUB1 reference mission that motivated the research and contextualizes it in the “New Space” paradigm.
- Chapter 3 brings a survey of the state-of-the-art FSW frameworks solutions found in the literature. A comparison among them is performed and the criteria behind cFS choice as the design paradigm for the reference mission is described and justified.
- Chapter 4 shows in details NASA cFS framework, its heritage, architecture and relevant tools.

[1] https://github.com/DaniloJFMiranda/OpenSatKit

- Chapter 5 presents cFS conformance to space standards related to on-board software services, tracing parallels between norms and cFS capabilities.
- Chapter 6 describes cFS mission-specific applications development, a domain where FSW developers spend considerable time in every satellite mission. This part of the document reports one of this research's main contributions: the proposed systematic approach for developing software applications. It aims to facilitate developers' job and help to improve cFS mission-specific applications quality.
- Finally, Chapter 7 contains the dissertation's conclusion and lists future work.

2 REFERENCE MISSION CONTEXTUALIZATION

As previously stated, this work adopts a reference space mission, with the intention of getting real requirements and checking the applicability of the proposal. The reference mission, VCUB1, is a CubeSat under development in a Brazilian space company called Visiona Space Technology, located in São José dos Campos.

The following definitions are fundamental concepts aiming to justify this dissertation's applicability not only to the chosen reference mission, but more generically to similar "New Space" missions.

2.1. **"New space" definition**

According to Paikowsky (2017), the changes caused by the greater involvement of the private sector in the global space activity in the past several years is being called "New Space".

This term is used to differentiate from traditional space projects, 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.

Under the "New Space" ecosystem, new and well-established companies are working to develop low-cost access to space and affordable space technologies and services, focusing on space as a resource and venue for profitable business.

Most "New Space" undertakings are private and commercial, offering various developmental and business models for innovative initiatives. They are inherently different from the traditional approaches to space activities. The fact that clients and investors are private actors triggers 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 (PAIKOWSKY, 2017).

Yet, according to Paikowski (2017), “in this context, the technological miniaturization of satellites enabled a decrease in the costs of developing and launching satellites. Satellites, systems, and components can now be purchased off the shelf. Development processes are shorter, and satellites spend relatively less time in orbit. As a result, project management in these fields is more inclined to take risks. It is tuned toward a ‘good enough’ R&D model and performing technological demonstrations while in service, instead of aiming for 100% success in orbit, as was the case for satellite development under the Old Space ecosystem.”

2.2. **CubeSat definition**

One of the most successful results of “New Space” are 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 x 10cm x 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, 2008).

CubeSat standard is maintained by Cal Poly university, in San Luis Obispo, USA. The 6U form factor is specified in “6U CubeSat Design Specification Revision 1.0” (CP-6UCDS-1.0)[2]. This specification for example is used as applicable document by many launch agencies, therefore affecting the entire CubeSat industry design.

[2] http://www.cubesat.org/

2.3. **FSW definition**

Flight Software (FSW) is a software that is running on a processor embedded in a spacecraft's avionics, being 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 usual computer systems as a result of the reduced accessibility to the hardware after launch, as well as the space environment intrinsic characteristics.

2.4. **VCUB1 reference mission**

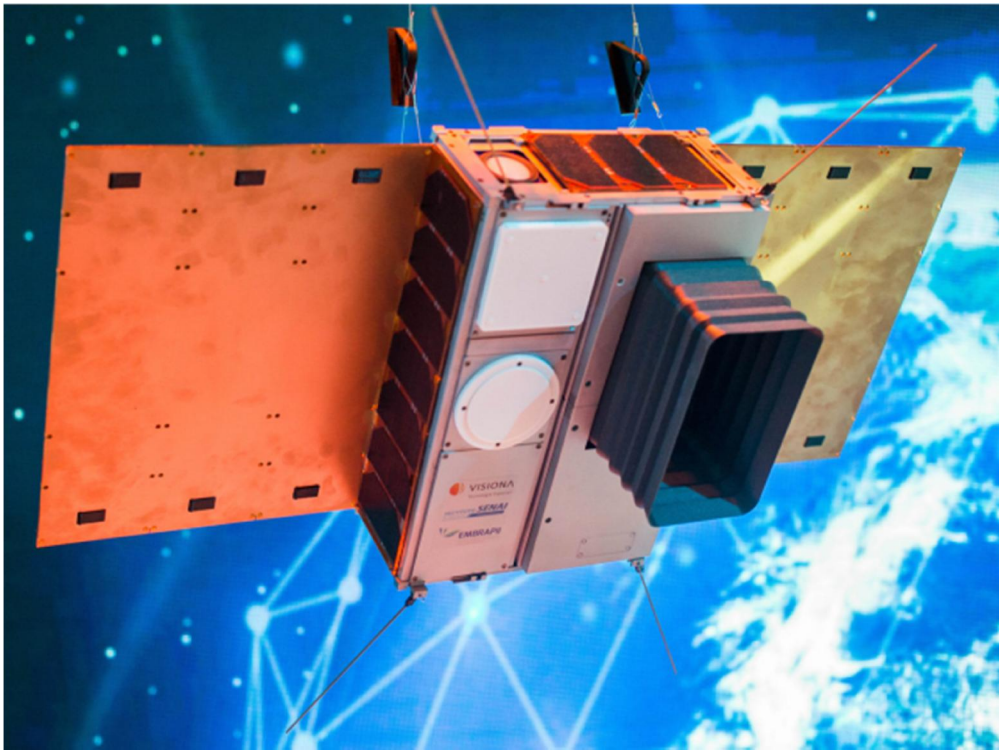
VCUB1 reference mission is an in-orbit demonstration CubeSat under development at the Brazilian space company Visiona Space Technology S.A, with target launch date foreseen for 2020. VCUB1 is fully compliant with 6U CubeSat specification (CP-6UCDS-1.0). A photo of the satellite can be seen in Figure 2.1.

The spacecraft contains two payloads. One is a multispectral optical camera, devoted to remote sensing, and 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 (DE CONTO et al., 2018).

This mission was targeted for being low-cost, fixed price, using several Commercial Off-The-Shelf (COTS) parts. It was initially conceived as a R&D project to validate in-orbit some company's software technologies, and therefore the project management philosophy is willing to accept some risk. The mission scope was later broadened to include remote sensing and IoT commercial

applications. The satellite will be launched as secondary payload, susceptible to the CubeSat launch opportunities available.

Figure 2.1 – VCUB1 mockup photo.



VCUB1 Spacecraft is a 6U CubeSat under development in São José dos Campos, Brazil. This photograph shows a spacecraft mockup in deployed configuration (operational mode) that was presented in the 8th Brazilian Industry Innovation Summit, June 2019.

Source: O Estado de São Paulo (2019).

Accordingly, it is a typical “New Space” mission following the CubeSat standard.

The Spacecraft’s FSW is running in a CubeSat COTS On-Board Computer (OBC). There is a single main OBC, which is responsible for interfacing directly with the TMTC radio, receiving commands and sending telemetry to the ground segment, and interfacing with all the spacecraft avionics. The OBC is also the bus controller for all digital data buses.

Thus, the strategy used on VCUB1 is centralized computing architecture, implying on more responsibility for the OBC, which is SPF (Single Point Failure). Consequently, the reliability required for the FSW is relatively high. To this aspect, one can sum up the low-cost and short-schedule VCUB1 management premises.

This scenario led the software team to look for a well-established open-source FSW framework compatible with CubeSat COTS On-Board Computers (OBC). The research on reliable FSW frameworks is subsequently presented in Chapter 3.

2.5. **Potential advantages of using frameworks for FSW**

According to Schmidt et al. (2004), the use of a well-established software framework leads to a shorter development program, with reduced costs and improved quality.

Besides that, going into the specificities of space software field, Birrane et al. (2009) affirms that frameworks are one of the five critical enablers for FSW systems.

He continues saying that isolation of individual tasks is a key motivator for spacecraft FSW frameworks. The reason for that is because it increases reuse of the software by allowing for unmodified tasks to be easily reused from spacecraft to spacecraft and reduces production costs as modules can be selected from a software library to be included in an architecture.

Responsive space systems are likely to have different mission requirements, different payload, and different supporting hardware and associated interfaces. This does not mean that large portions of the systems requirements specification cannot be stabilized. For example, the creation of standardized operating systems and frameworks has, in no way, restricted what can be done with modern computers. They have just defined the context in which operations must be performed.

Dos Santos (2008) performed further analysis into a broader scope than frameworks only, i.e. the higher-level problem of reusability, adaptability and variability in space software systems. He sustains that these systems require the use of abstraction, projection, and decomposition to be designed, understood, communicated, and maintained. As part of that effort, frameworks are sets of patterns that provide an architectural backbone that might incur in software reuse.

3 FLIGHT SOFTWARE FRAMEWORKS

The author performed a survey on the literature to find a representative set of FSW frameworks that could potentially be adopted in the reference satellite mission presented in Chapter 2.

During the research first steps, the author was skeptic if there were indeed reliable frameworks solutions that could be used in “New Space” CubeSat missions. Along the research, however, it was noticed that some of the architectures presented herein have even gained recognition and are starting to be accredited by CCSDS flight software standardization committee. That fact was an indication that such frameworks existed and motivated the engagement on further investigation.

Despite finding many academic and commercial framework solutions, this research ultimately channelized its effort on studying FSW architectures that had a minimum amount of available material, allowing further analysis and comparison.

3.1. **Comparison criteria**

There are many studies related to software performance, quality and cost assessment. As a disclaimer, this section cannot and does not intend to be exhaustive on software comparison criteria. The author, while investigating each framework documentation, collected and observed evaluation criteria that were judged significant, keeping in mind the intended use in the reference mission.

The chosen criteria, to be detailed in Table 3.1, are: 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 from 1 to 3 came from the CubeSat reference mission high-level requirements document. Nevertheless, these are typical requirements from space missions, especially in “New Space” paradigm.

Quality attributes (criteria 4) were taken from Wilmot et al. (2016). Not all fourteen quality attributes identified by NASA’s Software Architecture Review Board were taken, but only four of them, judged to be more closely related to software implementation. To those quality attributes, the author added “Specification Traceability” and “Well-Defined Semantics”, inspired in some framework’s documentation.

Criteria 5 to 7 were created by the author. They intend to assess the frameworks capability to evolve, improve (“User-Community Collaboration”) and standardize in the future (“CCSDS”), all that supervised by their sponsoring/maintaining institutions (“Long-Term Support”).

Table 3.1 – 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

#	Criteria	Description
4.2	Specification Traceability	Use of architectural rules or requirements to specify the software implementation and comprehensive traceability from user requirements down to the code
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

Source: Made by the author.

3.2. **Related work**

There are few comparisons of frameworks for spacecraft use in the literature. 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, Rexroat 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 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. This explains the differences in selection criteria, frameworks to be compared and results between the two researches. Still, valuable lessons were taken and applied in the present work.

3.3. **FSW frameworks summary**

The following subsections will present a summary of the six relevant FSW frameworks judged compatible with the reference mission and "New Space" characteristics described in Chapter 2.

3.3.1. GERICOS

The GEneRIC Onboard Software (GERICOS) framework was developed and qualified by LESIA (*Laboratoire d'études spatiales et d'instrumentation en astrophysique*), a French space laboratory, for the rapid development of payload flight software.

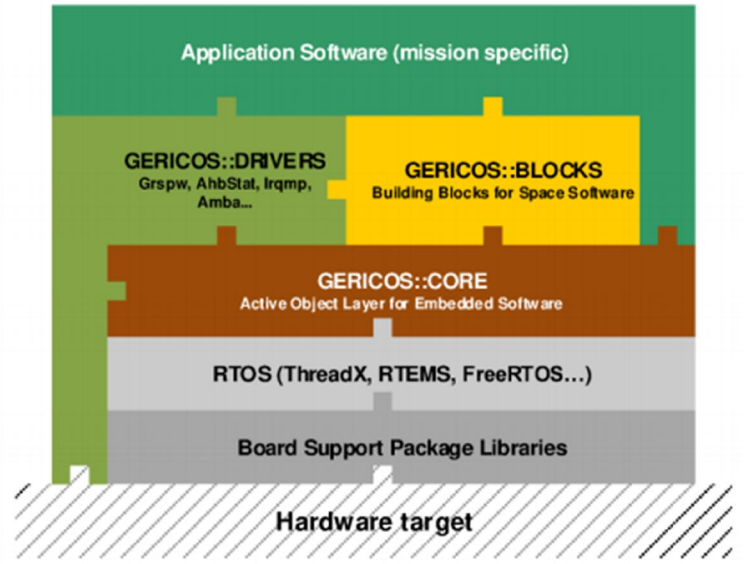
The GERICOS studies started in 2007. At the time there was no off-the-shelf software solution available, and at the end of 2015 the Radio Plasma Wave instrument flight software was delivered, built and qualified using GERICOS framework (PLASSON et al., 2016). This instrument will fly on ESA's Solar Orbiter 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 Figure 3.1:

- GERICOS::CORE: Lightweight (small memory footprint: 10 kB), 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 the concept of 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 of a specific real-time operating system and from a specific hardware target. Each active object (called task) has its own message queue and computational thread, which processes incoming messages one by one by executing the corresponding methods.
- GERICOS::BLOCKS: The second layer offers a set of reusable software components for building flight software based on generic solutions to recurrent functionalities: telecommand management, telemetry management, European Coordination for Space Standardization (ECSS) Packet Utilization Standard (PUS) service implementation, mode management, time management, CCSDS protocol management, etc.
- GERICOS::DRIVERS: The third layer implements software drivers corresponding to COTS IP cores used in the chip.

In order to formally model and document the different concepts provided by GERICOS, it was created a specific UML profile, called GERICOS UML profile. ESA's Solar Orbiter mission was completely modeled using this semantics.

Figure 3.1 – GERICOS Software Layers.



The French GERICOS software layers, from BSP up to application software.

Source: Obtained from Plasson et al. (2016).

With respect to specification traceability and conformance to standards, GERICOS::BLOCKS offers an implementation of PUS services and sub-services focused on what is required at payload level at ESA missions (PLASSON et al., 2016): PUS services 1, 3, 5, 6, 9 and 17. More details about ESA PUS services will be provided in Section 5.2.

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 processor target, that allowed GERICOS use in the context of CubeSats, as made in PicSat mission launched in January 2018 (LAPEYRERE, 2017). The code nevertheless is not yet open-sourced, preventing a more detailed analysis of the framework architecture.

3.3.2. NASA cFS

The core Flight System (cFS) platform is a 3-layered flight software architecture developed by NASA Goddard Space Flight Center (GSFC) that provides a hardware and operating system independent application runtime environment. The product line includes software lifecycle artifacts that can be configured to meet the user's requirements.

The cFS provides benefits including:

- Freely available high-quality flight software
- Provides common spacecraft flight software functionality
- Portable across many platforms
- Reduces project cost and schedule risks
- Users can focus on mission specific applications
- Active user community.

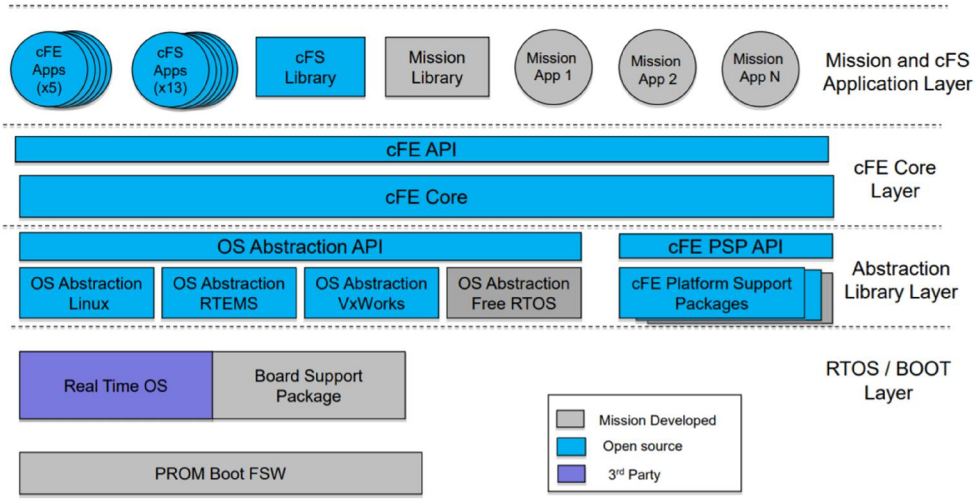
The United States of America space agency NASA has a Technology Transfer Program according to which they release from time to time some technology developments to the public. Some of these offsets are for restricted use and others for the wide-public. In the software world, the NASA Software catalog 2017-2018 lists all developments that were released in that period. The technologies related to cFS are shown as open source and licensed under the NASA Open Source Agreement Version 1.3, which guarantees the non-exclusive, world-wide, royalty-free source-code distribution (NASA SOFTWARE, 2017).

The cFS was originally developed as NASA Class B software for GSFC science missions. Nevertheless, since then there has been a wide range of adoption from manned space and scientific missions to CubeSats and even to a Lunar Mission (CORE FLIGHT SYSTEM, 2017).

NASA cFS is composed of three layers, as shown in Figure 3.2, 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 Card.
- 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.
- Mission and cFS application layers: composed of reusable and mission-specific applications. NASA has provided some cFS applications distributed as open source (NASA, 2017) that implement common spacecraft functionality. Additionally, a new mission must define and implement its mission-specific functionalities by means of cFS applications.

Figure 3.2 – NASA cFS Software Layers and Components.



NASA cFS is composed of three tiers, as shown in blue. The color code shows what software packages are open-sourced by NASA, what need to be developed by each mission and 3rd party provided software (RTOS)

Source: Obtained from NASA (2014).

With respect to flight heritage, cFS was initially developed by NASA's Goddard Space Flight Center (GSFC) over many years. It was based on the heritage of successful missions. The cFS components were incrementally developed and publicly released. The core Flight Executive (cFE) including the platform abstraction layer was first used on the Lunar Reconnaissance Orbiter (LRO) launched in 2009 and 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", led by Mr Jonathan Wilmot (GSFC/NASA), is in charge of conducting a project called "NASA cFS as a CCSDS Onboard Reference Architecture". The existence of this project highlights the importance of cFS. cFS code and documentation is available at

NASA's GitHub page[3] and there is also a community page[4] for getting help about cFS and questions and answers.

3.3.3. SAVOIR / COrDeT

The European Space Agency (ESA), inspired by the AUTomotive Open System Architecture (AUTOSAR), an open and standardized automotive software architecture started in 2002 that was released for the first time in 2005 (AUTOSAR, 2018), launched a space standardization initiative called Space AVionics Open Interface aRchitecture, a.k.a. SAVOIR (TERRAILLON, 2012).

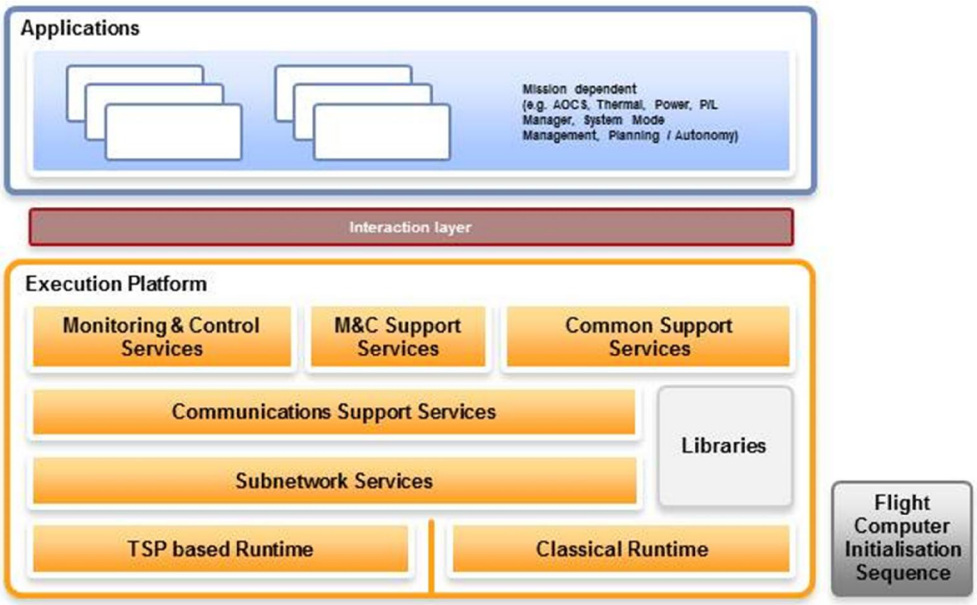
The SAVOIR working group was formally started in November 2008, as an outcome of ESA's yearly workshop called ADCSS (Avionics Data, Control and Software Systems) (SAVOIR, 2018).

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.

It was proposed an architecture (named COrDeT: Component Oriented Development Techniques) based on the segregation of the application software (mission-specific applications, which are normally independent from execution context) and the execution platform (basic services) (TERRAILLON, 2012). Figure 3.3 shows the static architecture of COrDeT reference architecture.

[3] https://github.com/nasa/cFE
[4] http://coreflightsystem.org/

Figure 3.3 – SAVOIR-COrDeT Software Reference Architecture.



The COrDeT software reference architecture shows mission-specific applications separated from execution platform services.

Source: Obtained from SAVOIR (2018).

It is worth to mention that COrDeT is part of a broader avionics reference architecture that the SAVOIR committee is promoting in Europe. Figure 3.4 presents the SAVOIR avionics reference architecture and gives an idea of the scope of this standardization task.

COrDeT output from SAVOIR group was a reference architecture with no particular implementation and API rules. Nevertheless, one of the companies of the industrial consortium funded by an ESA Contract, which goal was to specify COrDeT (P&P SOFTWARE, 2008), has provided a specification traceability of the framework and also a C-language implementation of COrDeT which is publicly available at the company's webpage (P&P SOFTWARE, 2018).

The P&P Software GmbH company from Switzerland is as of Dec 2019 extending the CORDET Framework implementation to support a subset of PUS pre-defined services (PASETTI, 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[5]. The existence of this project highlights the importance of COrDeT/SAVOIR.

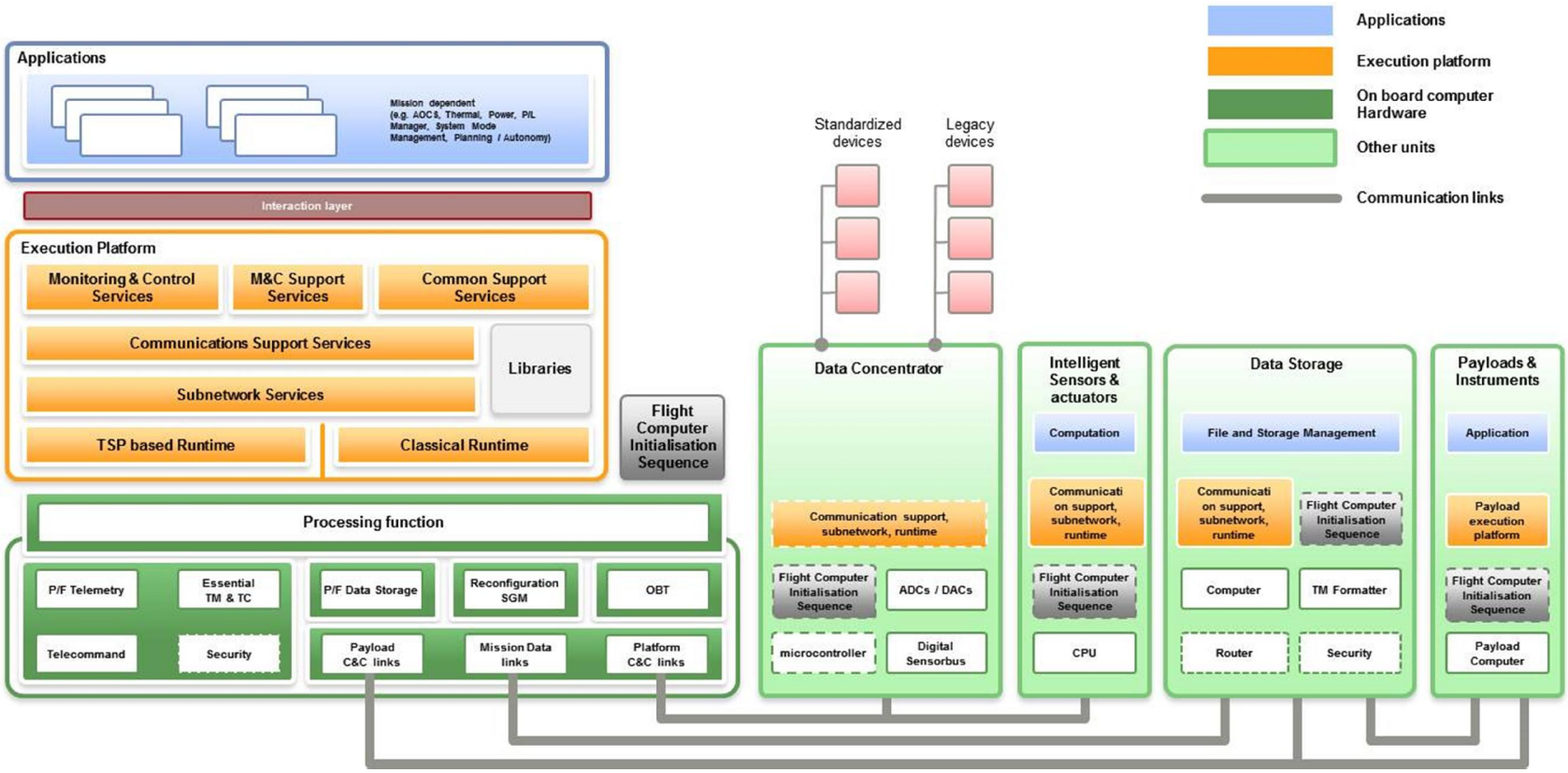
C2 implementation, which is a C-language open-source implementation of the CORDET Framework made by P&P Software GmbH, allowed the author to perform a technical analysis on the software source-code and in the COrDeT framework itself.

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, that will be flying in Dec 2019 (P&P SOFTWARE, 2018).

The author noticed that the mechanism through which messages are sent from one application to another is outside the scope of the framework. C2 framework assumes that a middleware layer is present which can be used to send and receive messages to and from other applications. Also, the author found no binding layer between C2 and the underlying real-time operating system. This connection probably will be made by the middleware layer that is missing in this framework.

[5] https://cwe.ccsds.org/fm/Lists/Projects/DispFormDraft.aspx?ID=547

Figure 3.4 – SAVOIR Avionics Reference Architecture.



The SAVOIR avionics reference architecture containing hardware and software standardization.

Source: Obtained from SAVOIR (2018).

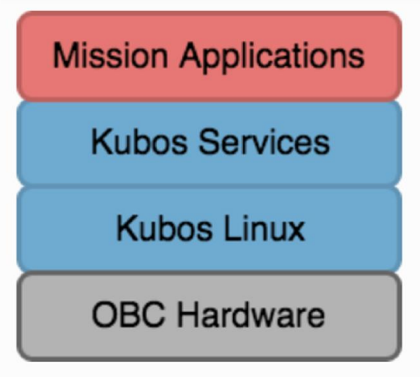
3.3.4. KubOS

Kubos is a space software start-up company located in Texas, USA. It was founded in 2014 and developed the KubOS flight software framework. The analyzed software is at version v1.5.0, and has BSP for 3 major CubeSat OBC's, and its full package is open-source. Satellite users can pay for additional features, like support services and software updates (KUBOS, 2018).

KubOS stack is shown in Figure 3.5. Each layer means:

- Kubos Linux: Kubos Linux is 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) and 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 that is 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 behavior of the satellite. They for example govern state management, accomplish scripted tasks and monitor onboard behavior. Each 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 app, or complex, such as a payload operations app (KUBOS, 2018).

Figure 3.5 – KubOS Stack.



The KubOS stack, starting from the OnBoard Computer (OBC), going to Kubos Linux Operating System, Kubos Services and finally Mission Applications.

Source: Obtained from Kubos (2018).

KubOS v.1.5.0 is open-sourced in a GitHub repository. Also, first time users can create its own KubOS project using a Software Development Kit (SDK) provided by the company. The author noticed that KubOS is light-weight and provides the basis for writing mission applications using Rust or Python languages. C and Lua are also claimed to be supported (KUBOS, 2018).

3.3.5. CAST software reference architecture

China Academy of Space Technology (CAST) is the main spacecraft development and production facility in China, subordinated to the China Aerospace Science and Technology Corporation (CASC), which is the state-owned main contractor for the Chinese space program, responsible for execution of the Chinese National Space Program. On the other side, the China National Space Administration (CNSA) is the main responsible for National space policy.

Integration of functions in spacecraft avionics system is a trend in the development of spacecraft (XIONGWEN, 2015). The problem of spacecraft avionics software reuse is mentioned as one of the main points to be solved.

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

According to Xiongwen (2015), the results of the test and validation demonstrate that the software architecture has great positive effects on software reuse. Under CNSA lead, this effort has gained CCSDS standardization status through a dedicated working group[6].

Figure 3.6 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, which has standard program interface and protocols, and can realize the data exchange and cross support among different hardware and operating system, as well as data exchange and cross support among different hardware and operating system. In order to make the middleware extendable and support technology upgrades, the middleware is divided into three configurable layers:
    - Subnetwork Layer: Defined to shield data links and service components to the upper layers. Its support includes onboard subnetwork components and space subnetwork components.

[6] https://cwe.ccsds.org/fm/Lists/Projects/DispForm.aspx?ID=595

    - Transfer Layer: Provides standard interface to the layer above to transfer data. CCSDS Space Packet Protocol is included and future extension is allowed
    - Application Support Layer: Provides the standard service components to support the application, which includes SOIS application support layer services and PUS services.
- Application Layer: contains most of the common functions of avionics systems. The implementation of this layer may be different among different projects.

In order to validate the software architecture, CAST has designed and developed all its components (CCSDS, 2016). Based on the hardware platform of avionics system requirements, these components were assembled and tested.

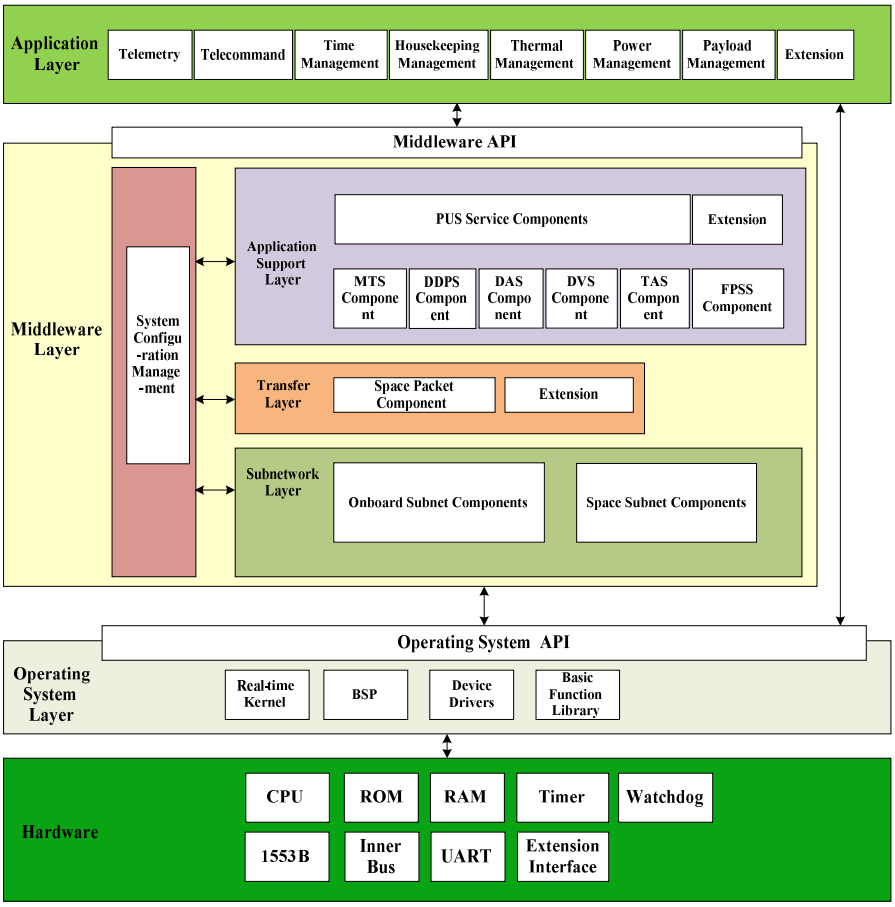
The FSW was assembled using general-purpose software components, with accordingly customizations in the runtime parameters and processes.

The total code size of the prototype software was 54542 lines, of which the code size of software components was 45995 lines, accounting for 84.3% of the total code lines (CCSDS, 2016). The code was programmed in C language (XIONGWEN, 2015).

A number of test cases showed that CAST flight software architecture based on CCSDS standard cannot only provide richer, more practical and powerful functions than traditional spacecraft software system, but also bring the change of whole software development mode, which can improve efficiency and reliability of software development (CCSDS, 2016).

The CAST source-code architecture was not found as open-source and the author also found no evidence of flight heritage.

Figure 3.6 – CAST Software Reference Architecture.



The CAST software reference architecture presented in layers and unit blocks.

Source: Obtained from CCSDS (2016).

3.3.6. 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 (to be launched in Dec 2019). This mission is intended to demonstrate the improvements in mission control capabilities that will arise when satellites can fly more powerful on-board computers.

NMF provides a standard on-board software framework for nanosatellites based on the CCSDS MO framework, that facilitates not only the monitoring and control of the nanosatellite software applications, but also

the interaction 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, 2016).

The CCSDS MO framework is defined in CCSDS 520.0-G-3 green book standard and consists of end-to-end services based on a service-oriented architecture and is intended to be used for mission operations of future space missions.

The layered MO service framework allows mission operation services to be specified in an implementation and communication agnostic manner. The core of the MO service framework is its Message Abstraction Layer (MAL) which ensures interoperability between mission operation services deployed on different framework implementations.

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++ light-weight software to Java multi-platform agnostic applications.

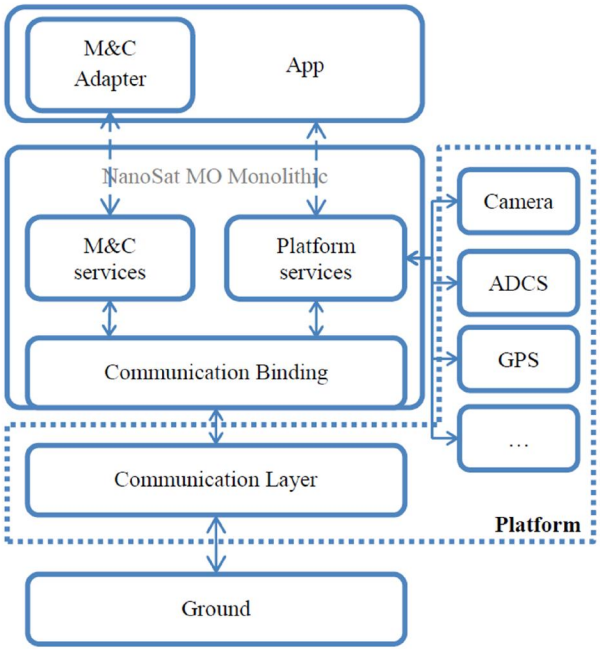
Figure 3.7 presents NMF architecture with one application only. For deploying multiple applications, that can be concurrent in the use of peripherals, NMF proposed the use of two new components: the NanoSat MO Supervisor and NanoSat MO Connector. The first one is responsible for managing the apps and providing Platform services that can be utilized by different applications.

The second is responsible for connecting to the Platform services, exposing them to the business logic of the application and exposing interfaces to monitor and control the app from ground or from other applications. The inter-apps architecture is shown in Figure 3.8.

Coelho (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.

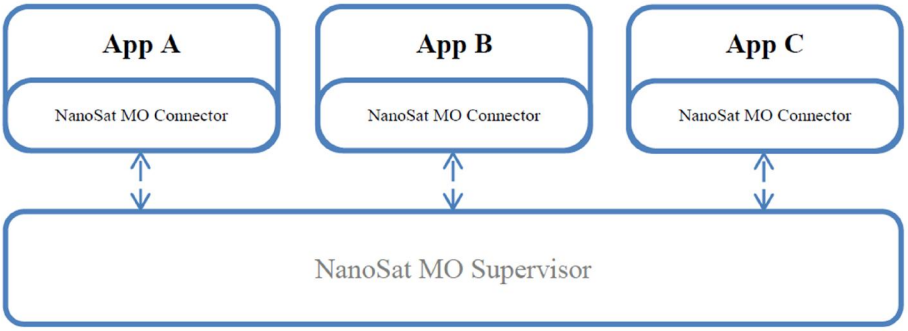
Figure 3.7 – NanoSat MO Framework monolithic simplified architecture.



NanoSat MO Framework architecture showing one application only (monolithic) stack, on top of the basic framework services.

Source: Obtained from Coelho (2016).

Figure 3.8 – NanoSat MO Framework inter-application architecture.



NanoSat MO Framework architecture showing multiple applications interacting through NanoSat MO Connector and MO Supervisor components.

Source: Obtained from Coelho (2016).

3.4. **Software architectures comparison**

It is not hard to notice a great similarity between the frameworks presented in Section 3.3. As a matter of fact, the space organizations that developed them had to overcome similar problems while developing their spacecraft missions, and probably investigated each other's projects.

The return of experience of space agencies and space system experts in their mission's software were already systematized and standardized by competent institutions, such as CCSDS and ECSS, through working groups, in space software standards (some of them will be detailed in Chapter 5). That's why most of the frameworks are based on or mention the same applicable space standards.

Even with these similarities, some differences among the frameworks were noticed and will be compared subsequently. The conformance to each selection criteria presented in Section 3.1 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.

The comparison chart between the flight software frameworks is presented in Table 3.2.

Table 3.2 – Software Architectures Comparison Chart.

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)	Not found	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 (This implementation requires a powerful OBC)
4.1	Reliability	Yes	Yes	Not found	Not found	Yes	Not found
4.2	Specification Traceability	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)

Criteria		cFS	COrDeT/SAVOIR (through C2)	CAST	GERICOS	KubOS	NanoSat MO
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 Processors)	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)	Yes (GERICOS::CORE allows that)	Yes (Portability to FreeRTOS, Linux and to some CubeSat OBC's)	Yes (Due to Java capabilities)

Criteria		cFS	COrDeT/SAVOIR (through C2)	CAST	GERICOS	KubOS	NanoSat MO
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 Compliant (Not standardized as a reference architecture by specific working group, but it is a spin-off of CCSDS 520.0-G-3 standard)

Source: Made by the author.

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 that fully specify 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 soon.

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 software suite but also an open-source ground control software tailored to interface with cFS and a CubeSat simulator example, providing then 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 3.2 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 has no flight heritage and it depends on a non-defined message-passing middleware to run. Therefore, it could not be immediately used as is.

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. The long-term support criteria is key for space organizations that want to rely on a 3rd-party framework.

Held since 2007, the Flight Software Workshop is considered the most important conference in spacecraft flight software in the USA and since 2015 there is an entire day dedicated to cFS, which shows the growing interest and utilization of cFS among space software developers. It is interesting to add that due to its notable capabilities, cFS is gradually becoming a wide-NASA standard, with growing interest from the American and international community.

3.5. **NASA cFS adoption for the reference mission**

According to the results presented in the frameworks comparison table 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.

cFS adoption in the reference nanosatellite mission involved a detailed understanding of the framework. An analysis of the cFS architecture, existing cFS core applications and cFS available high-level open-source applications was performed subsequently in order to allow the customization of cFS and the creation of mission-specific applications for VCUB1 mission. Next chapter will present the results of this study.

4 NASA cFS FRAMEWORK

This Section will present details about cFS, its architecture and layers, the open-source available applications and the concepts that will be used throughout this dissertation.

4.1. cFS heritage

4.1.1. cFS heritage at NASA

Prokop (2014) points out that several NASA science missions have been using cFS as framework for their flight software since 2009. Some missions are presented below along with their launch dates:

- Goddard Space Flight Center (GSFC) Missions:
    - Lunar Reconnaissance Orbiter (LRO) (2009)
    - Solar Dynamics Observatory (SDO) (2010)
    - Magnetospheric Multiscale Mission (MMS) (2014)
    - Global Precipitation Measurement (GPM) (2014)
- Ames Research Center Missions:
    - Lunar Atmosphere and Dust Environment Explorer (LADEE) (2013)
- Applied Physical Lab (APL) Missions
    - Radiation Belt Storm Probes (RBSP) (2012)
    - Solar Probe Plus (SPP) (2018)

The oldest technical article about cFS found on the literature research is Wilmot (2005). At that time the cFS core, the cFE, was still under development and testing and only the OSAL, which is a NASA GSFC stand-alone project, was open-sourced. Nevertheless, Ganesan et al.

(2009) mention that the development of the cFE/cFS flight software product line started even earlier, in 2003.

An interesting return of experience about cFS use in NASA projects was done by McComas (2015). He presents there some feedbacks from his experience as NASA Flight Software expert for 30 years, including cFS history, release process and cFS lessons.

McComas et al. (2015) pointed that OSAL, cFE and the cFS Applications Suite were then mature software products, classified as TRL 9, the highest level of technology maturity.

4.1.2. cFS heritage outside NASA

cFS is being adopted in several space organizations besides NASA, including the private sector, especially in the USA. For example, Lockheed Martin, the biggest defense company in the world in 2017 and 2018[7], adopted cFS as the flight software framework on its LM 50 small spacecraft platform series. They started to use cFE/cFS in 2014 (AKRE, 2017).

Outside the USA, there are few mentions found about cFS adoption. Some causes for this are that cFS public release is relatively recent and that some countries have their own software paradigm, as was presented in Chapter 3. Nevertheless, an interesting application outside the USA was found in South Korea, in Korean space agency KARI nanosatellite programs (CHOI, 2016).

4.1.3. cFS adoption in cubesat missions

VCUB1 mission is not the first CubeSat mission to adopt cFS as FSW framework. A search on the literature revealed that some nanosatellite

[7] *According to http://people.defensenews.com/top-100/*

missions such as NASA's Dellingr, CERES and STF-1, have already used cFS.

Dellingr is a GSFC engineering demonstration CubeSat mission, containing two heliophysics science instruments: a boom-mounted fluxgate magnetometer, and a novel gated time-of-flight ion-neutral mass spectrometer (INMS). This mission was deployed from ISS in Nov 2017 and faced several hardware anomalies and failures during its time in orbit (KEPTO et al., 2018). Some cFS built-in features enabled spacecraft maintenance, including:

- Upload files to the on-board file system;
- Verify the size and CRC of a file on the file system;
- Manage on-board files;
- Individually replace cFS applications;
- Compress cFS applications to reduce uplink bandwidth;

This return of experience in flight shows cFS flexibility, which allowed system maintenance, particularly important in "New Space" missions were we typically design for resiliency rather than for reliability. Still on Dellinger mission, Cudmore (2017) provided some FSW development feedback about cFS usage:

- Positive Lessons:
    - cFS brought a development environment, FSW framework, and process to the project
    - cFS allowed to focus on mission specific code and start to work on that immediately
    - cFE and cFS functionality added a lot to the mission with little effort
    - cFS cross platform capability allowed to develop and run on desktop Linux, Raspberry Pi, and other targets

- Negative Lessons:
    - The cFS was a poor fit for the chosen CubeSat COTS OBC, which had 2 MB of RAM and File System could hold at most 64 files. Experience was used to work around limitations of the platform
    - cFS experience helped with the selection and configuration of applications. Learning the cFS is beneficial before trying to develop a system

Among these negative lessons, particularly regarding the minimum size for running the full cFS bundle, McComas (2016) stated that on Global Precipitation Measurement (GPM) mission, an operational NASA Class B project that used cFS, there were 4 MBs of EEPROM and 24 MBs of SRAM on the main OBC. Nowadays, most CubeSat OBCs have memory performances far above those requirements, so cFS footprint will likely no longer be an issue for most nanosatellite missions.

A CubeSat mission launched on Dec 2018, called CeREs (the Compact Radiation belt Explorer), on the opposite side, made use of a high processing-power computer for its OBC and cFS as its FSW framework. Kanekal et al. (2018) stated that a computer called CSP (CHREC Space Processor), a development based on Xilinx Zynq 7020 platform, was used on-board and that all cFS applications were successfully reused with minor project-specific configuration changes.

Aphelion Orbitals, an American space start-up company, developed an Operating System called Perihelion, which is based on cFS. According to Aphelion (2018), users can benefit from an SDK and a software package repository where Perihelion applications can be downloaded and used in customers' nanosatellite missions.

A CubeSat called NUTS (NTNU Test Satellite) was developed by NTNU (Norwegian University of Science and Technology) students aiming at educational purposes.

Normann (2016) stated in his master's dissertation that Component-Based Software Engineering (CBSE) and Service-oriented Architecture (SOA)

are popular engineering methods of constructing software for small satellites. As part of his studies, he examined both NASA cFE/cFS and ESA SAVOIR as FSW framework candidates.

He stated that these two frameworks could be “advantageous to use for the NUTS satellite, but after further investigation it was discovered that neither of these systems had official support for the target operating system (FreeRTOS). Also, both of these projects are ‘quite large’ and may add a level of complexity that wasn’t required for the NUTS mission”.

cFS official support for FreeRTOS is not yet open-sourced, but was already performed in Dellingr mission (CUDMORE, 2017). Moreover, after technical investigation on NUTS satellite, it was noticed that Normann (2016) was considering as OBC candidates two very resources-constrained microcontrollers: UC3C and SAM V71, which is very uncommon in current CubeSat missions, with only 2 MB of external SRAM, same as Dellingr mission.

cFS isn’t indeed a good fit for microcontrollers. It was designed for microprocessors systems with operating systems that contain at least file systems support, as detailed in Section 4.2.1.

Nevertheless, Wilmot (2017) affirms that cFS footprint using FreeRTOS is less than 1 MB, which makes it suitable even for resource constrained processors.

Another NASA CubeSat called STF-1 (Simulation-To-Flight), launched in the same flight as CeREs, also used cFS as its FSW framework. This mission main goal was to develop and demonstrate the life-cycle value of a software-only small satellite simulator, called NOS3 (MORRIS et al., 2016). This simulator was developed to interface with cFS and has now been released open-source.

Araguz et al. (2018), in his Cat-1 CubeSat project from Technical University of Catalonia, launched in Nov 2018, stated that adopting cFS as middleware may ameliorate the robustness of the system since this product have been exhaustively tested and verified in-flight.

Nevertheless, he continues saying that “Regardless of this suite already being tested in NASA’s nanosatellite missions, its adoption is still not broad enough and many current developments are still implemented on top of simpler, commonly known operating systems (e.g. FreeRTOS or standard Linux kernels), which lack most of the reliability guarantees of additional, space-qualified middleware”.

Section 4.2.1 will present OSAL cFS port to commonly used RTOS systems. RTEMS and VxWorks are systems already used in several critical space missions worldwide. cFS port to them was already performed for some OBC processors, but not for most CubeSat’s COTS OBCs. Moreover, CubeSat community is used to work with simpler RTOS such as FreeRTOS and some Linux kernels.

Therefore, the effect of simpler RTOS overall effect on system reliability must be deeper analyzed in order to assess if it does not jeopardize cFS reliability guarantees. On the other hand, Dellingr mission used cFS over FreeRTOS for the first time in orbit with considerable success, cFS maintenance capabilities being required several times in spacecraft operations, aggregating resilience to the system (KEPTO et al., 2018).

4.2. **cFS architecture overview**

4.2.1. OSAL

The Operating System Abstraction Layer (OSAL) is one of the constituents of the cFS lower level software layer. It started as a stand-alone project at NASA GSFC, as mentioned in Section 4.1.1, conducted by computer engineer expert Alan Cudmore. The first OSAL release was in 2004 (YANCHIK, 2007). This dissertation uses OSAL v4.2.1, issued in 2016, as obtained from NASA open-source repository.

OSAL is a powerful software wrapper that isolates higher flight software layers from the underlying RTOS. With the OS Abstraction Layer, flight software such as cFS can run on several operating systems without

modification. It also allows execution of FSW on simulators and desktop computers.

The most common RTOS used in spacecraft flight software, such as RTEMS and VxWorks, have already been ported and are available open-source (CUDMORE, 2016). Also, there is an OSAL port for Linux, which allows to run cFS and develop cFS applications in a host machine.

More details about OSAL can be found in section A.1 of APPENDIX A.

4.2.2. PSP

PSP is a cFS software product that allows port and memory based I/O access to provide a common way of accessing hardware resources. This dissertation uses open-source PSP v.1.3.0.0 of May 24, 2016. PSP purpose is to isolate higher cFS layers from specific processor and memory calls.

More details about PSP can be found in Section A.2 of APPENDIX A.

4.2.3. cFE

After OSAL and PSP, which according to Figure 3.2, compose the Abstraction Library Layer, the subsequent upper layer in the cFS system is the core Flight Executive (cFE). This dissertation uses open source cFE v6.5.0.

cFE is composed of five core applications that provide the underlying services for mission applications to be built on. They were briefly explained in Section 3.3.2. McComas, Wilmot and Cudmore (2016) stated that cFE provides five services that were determined to be common across most FSW projects at NASA GSFC.

Each one of the five services are detailly explained in sections from A.3 to A.8 of APPENDIX A. The cFE API's knowledge is crucial to develop new cFS applications and to operate a spacecraft that is cFS-compliant.

4.2.4. cFS applications suite

Besides the cFE core applications mentioned in previous Section 4.2.3, NASA also open sourced on its GitHub page[8] some cFS applications that were judged to be recurrent in space missions (NASA, 2014). These applications have a strong correspondence with flight software standards proposed services, as will be shown in Chapter 5.

The cFS applications suite are presented in Table A.2 of APPENDIX A, along with the version used in this dissertation and a summary of their functionalities, which were extracted and adapted from NASA GitHub page.

Along this document, in all subsequent applications mentions, the implicit software versions are the ones referenced in Table A.2 if not otherwise specified.

4.2.5. cFS architecture design rules verification

Ganesan et al. (2009) performed a study in order to verify architectural design rules of the NASA cFS product line implementation. The goal of the verification was to check whether the implementation was consistent with the cFS' architectural rules derived from the cFE developer's guide, cFE/cFS requirements documents and cFS deployment guide documents.

The scope of the study was a subset of rules that were related to the cFS structure and its development environment, that could be statically verified (i.e., without executing the system).

Overall, 45 architectural rules were checked (e.g. a higher-layer component is allowed to call a lower-layer component, but the opposite is not allowed), most of them were indeed followed, but 14 violations were detected. Even though cFS is a safety-critical software, and consequently

[8] https://github.com/nasa?utf8=%E2%9C%93&q=cfs&type=&language=

undergoes extensive code reviews, still some of the detected violations escaped the manual review process. High-priority issues were addressed at the time and corrected on subsequent cFE/cFS versions.

4.3. **cFS unit tests**

In the context of a Software Product Line (SPL) such as cFS, unit testing is especially important and complex because of the challenges imposed by testing core modules and application modules without being dependent on the behavior of any other module.

The unit testing approach developed and used by the cFS product line team at the NASA GSFC incorporates practical and useful solutions such as allowing for unit testing without requiring hardware and special OS features in-the-loop by defining stub implementations of dependent modules (GANESAN et al., 2013). cFS open-source v6.5.0 code come along with a comprehensible unit testing framework, that contains unit tests and respective stubs for each software application.

Ganesan et al. (2013) considered cFS unit testing framework an example of good software engineering. For instance, NASA addressed the problem of modules dependability by introducing stubs, counter-value pattern and abstraction layers for handling variability in SPLs, even with C language lack of built-in assertion capabilities and dependency injection. Even though, Ganesan et al. (2013) found some issues that can be optimized. For example, he mentioned that one example of possible cFS unit testing optimization would be to separate nominal and off-nominal behaviors that are currently mixed in several of the unit tests.

4.4. **Reliability analysis of a cFS-compliant FSW**

In collaboration with NASA Goddard Space Flight Center (GSFC), Sukhwani et al. (2016) analyzed the defect reports for the cFS-based FSW of a GSFC space mission that has was recently launched at the time. For

their analysis, they used the defect reports collected during the FSW development. The software was developed in multiple releases, each release spanning across all software life-cycle phases.

The analysis has shown that some of the major software problems included integration issues with new hardware platforms, issues with new releases of COTS software (cFE/cFS) and problems encountered in hardware test environment.

Regarding this dissertation scope, it is mostly of greater interest the defects that have cFE/cFS as the root cause. Sukhwani et al. (2016) pointed that three different releases of cFE were used during the mission development. The releases replacement caused some issues such as a change in the code directory structure in one occasion and schedule delay due to new cFE release await. Even with these minor problems identified, Sukhwani et al. (2016) stated that Product line approach in Flight Software is the future of flight software development for years to come.

4.5. **cFS from a closed environment to an open ecosystem**

The concept of product line for software products is gradually being growing. This is particularly true in consumer products electronics and has a strong correlation with partially or completely open-sourcing the software product (HARTMANN, 2015).

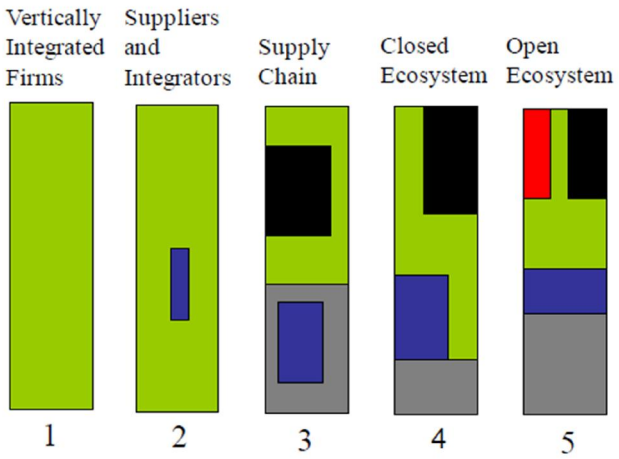
FSW can also benefit from product line concept. According to Dos Santos (2008), the process of developing a complex software can be, due to the use of product line, reduced to the comparatively simple activities of feature selection and composition.

Lindvall et al. (2016) considered cFS a good example of system that was designed as product line due to its success in being reused by many NASA missions. This author also states that cFS was not an a-priori established product line that later evolved. On the contrary, it was defined after several similar systems were developed and it evolved in parallel and where the need for organized and planned reuse was identified.

Hartmann (2015) presents 5 types of structure typically adopted by the industry for software development, as shown in Figure 4.1, from type 1 (more closed model) to type 5 (more open model). In this diagram, external software is represented as one or more colorful boxes that are aggregated to the green original software system developed by the host organization.

cFS environment can be understood as gradually evolving from model 1 to model 5 of Hartmann (2015) categorization, which is a trend on software development in several areas.

Figure 4.1 – Models of structure for software development.



This figure shows the 5 models proposed by Hartmann (2015) for software development in software industry. Type 1 is completely verticalized and entirely made by the developing organization while type 5 is a model type that is much more flexible, relying on the contributions of 3rd party organizations.

Source: Obtained from Hartmann (2015).

4.6. **OpenSatKit**

An example of cFS now being projected as an open ecosystem is an initiative called OpenSatKit[9], which is a complete software suite created to address cFS deployment challenges and to serve as a platform training for new users. McComas (2018) states that this SDK is freely available, as it contains only open-source software, and includes preconfigured cFS applications and provides tools for creating and integrating mission-specific applications.

The fact that cFS components are maintained separately by NASA gives to the user the responsibility to configure, integrate, and deploy them as a cohesive functional system. McComas (2018) considered this activity very challenging, especially for organizations with minimal FSW development experience, such as universities that are building CubeSats.

OpenSatKit contains, besides a subset of Section 4.2.4 cFS applications suite, Ball Aerospace's command and control system called COSMOS, and a flight dynamics simulator called 42. Starting with an operational flight and ground systems makes the FSW developer's job much easier, because users can focus on tailoring the kit's cFS components to their needs, adding new cFS-compliant applications, porting the cFS to their target platform, and verifying the system.

OpenSatKit v1.0 is the cFS development kit adopted in this work because it was the most "user-friendly" and complete cFS open source distribution the author found. Moreover, it is a self-contained environment, allowing complete FSW design and testing with no need for external tools.

The cFS theoretical review presented in this Chapter provides the framework background to better understand the cFS conformance with space standards, that will be presented in Chapter 5, and mission-specific applications generation that will be addressed in Chapter 6. Advanced cFS

[9] https://github.com/OpenSatKit/OpenSatKit

background contained in APPENDIX A provides deeper details that might be useful on the continuity of this work.

5 cFS AND SPACE SOFTWARE STANDARDS

This Section is devoted to present the main spacecraft operations standards, data communication protocols and flight software standards and in which ways cFS responds to or is related to these standards. This activity will provide interesting insights on what cFS can and cannot do, allowing its users to assess cFS quality and better understand the mission-specific applications generation that will be presented in Chapter 6.

Standards serve as a common language among specialists and as a lessons learned repository. Therefore, they can be used as reference for attesting capabilities and functionalities and also as benchmark for software projects development.

As presented in Section 2.1, “New Space” is characterized by private sector protagonism, which naturally leads to space projects cost reduction. This is an apparent contradiction with extensive space standards utilization. Nevertheless, “New Space” missions cannot give up on product quality and for that purpose compliance with relevant standards provides insightful metrics.

Also, evaluating cFS product line core compliance, i.e. cFE and cFS applications suite, is a one-time task. Next cFS-based FSW projects can inherit the analysis performed here and extend it to their mission-specific software parts.

cFS concepts presented in Chapter 4, especially architecture details given in Section 4.2, will be extensively used throughout this chapter and should be consulted whenever necessary.

5.1. **cFS and CCSDS**

Particularly regarding CCSDS standards, a world-wide recognized space standardization committee, Wilmot (2017) advocates that they can be

used to speed missions' schedules and reduce their costs. He states that cFS has been using several CCSDS standards since its inception.

In this Section, cFS compliance with CCSDS SOIS will bring up cFS existing software services.

It is important to remark that CCSDS also functions as an ISO Subcommittee, which is Subcommittee 13 (SC13 - Space data and information transfer systems), part of Technical Committee 20 (TC20 - Aircraft and space vehicles). ISO standards numbering differs from CCSDS. In this dissertation, however, CCSDS IDs only will be referenced due to the likelihood of space community familiarity.

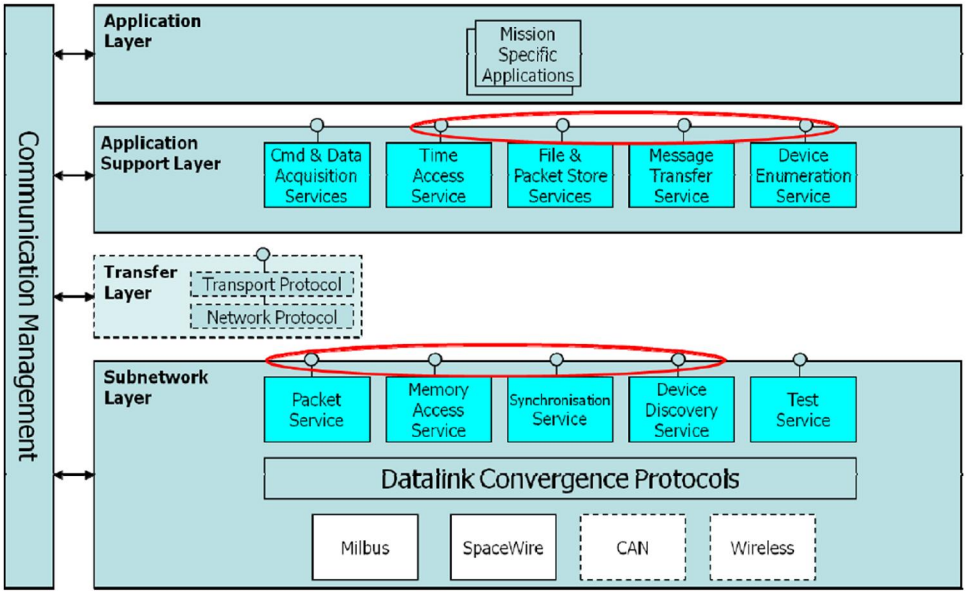
5.1.1. cFS and CCSDS SOIS

The Onboard Interface Services (SOIS) is one of the 6 CCSDS areas, being devoted to developing standard service interfaces that are provided to onboard software applications.

SOIS specification is done in CCSDS (2013), registered with the code CCSDS 850.0-G-2. This reference performs an analysis of the cFE framework mapping with respect to SOIS services, as shown in the highlighted interfaces in Figure 5.1.

Going further into the analysis of the cFS built-in CCSDS SOIS services, each service defined in the standard is associated with the judged correspondent cFE application (if applicable). The result is shown in Table 5.1.

Figure 5.1 – CCSDS SOIS services that have potential alignment with NASA cFE/cFS framework.



This CCSDS SOIS architecture block diagram shows CCSDS (2013) analysis of the potential SOIS services that can be aligned with cFE/cFS functions.

Source: Obtained from CCSDS (2013).

Table 5.1 – CCSDS SOIS standard correspondence with cFS functions.

CCSDS SOIS Layer	CCSDS SOIS Services	cFS Function
Applications Support Layer	Command and Data Acquisition Services (DDPS, DAS, DVS)	One has to implement by means of mission-specific cFS applications Not implemented
	Time Access Services	cFE Time Implemented
	Message Transfer Services	cFE Software Bus Implemented
	File and Packet Store Services	cFS Data Storage app, cFS File Manager app Implemented
	Device Enumeration Services	Not implemented
SOIS Subnetwork Services	Packet Service	cFS Software Bus Network app Implemented
	Memory Access Service	cFS Memory Manager app Implemented
	Synchronization Service	cFE Time Implemented
	Device Discovery Service	Not implemented
	Test Service	Applications health check and network connectivity using NOOP command Partially implemented
N/A	Electronic Datasheets	Not implemented

Source: Made by the author.

The Device Discovery (DDS) and the Device Enumeration (DES) services are being evaluated for inclusion in the cFE framework (CCSDS, 2013). This same standard also mentioned that the “Command and Data Acquisition” services are not present by default in cFE/cFS but are

commonly implemented by means of software drivers or hardware libraries.

Test service was not judged to be implemented in cFS by CCSDS (2013). However, another way to see it is that indeed it is partially performed by means of NOOP command presence in every application, which is the same idea behind PUS Service 17, that will be explained in Section 5.2.

Finally, cFE v6.6.0, which was publicly released on May 2019[10], has CCSDS EDS Support (STREGE, 2017).

5.1.2. cFS and CCSDS procotols

cFS compliance with CCSDS most used data transfer protocols, namely the Space Packet Protocol, Data Link Layer Protocols and CFDP are complimentary shown in 0.

This analysis can be used as a reference of cFS natively built-in CCSDS protocols functions. McComas (2015) pointed out that NASA cFS compliance with CCSDS packet formats is also an advantage for interfacing with NASA legacy ground systems. This remark is also true for several other CCSDS signatories' organizations around the world, particularly INPE, in Brazil.

5.2. cFS and ECSS PUS services

The European Coordination for Space Standardization (ECSS) is an initiative to uniformize space standards across European organizations. It has a broader scope compared to CCSDS, because it proposes standards to all areas related to a space project, not only to data systems.

[10] Mail by David McComas at NASA Core Flight Software Product Mailing List <cfs-community@lists.nasa.gov> on May 8, 2019

One ECSS particular standard, the ECSS-E-ST-70-41C (ECSS, 2016), is of great interest to FSW developers, because it is a precursor of CCSDS SOIS, being one of the first available standards to deal with FSW services.

This standard is called “Telemetry and Telecommand packet utilization”, also known as PUS (Packet Utilization Standard), whose first issue was published in 2003. At the time, 19 services were proposed (3 being reserved for future specification), and this number has grown to 23 in current issue C (the same 3 services still kept reserved). A comprehensive analysis on PUS-C features was performed by Clerigo et al. (2018).

These services are mission-agnostic aiming at the remote monitoring and control of spacecraft subsystems and payloads. The PUS should be viewed as a “Menu” from which the applicable services and respective levels are selected for a given mission (ECSS, 2016).

ECSS PUS, besides the on-board services, also specifies the TC and TM packet templates that should host a request, or a report derived from the intended on-board service.

The cFS TC and TM packets secondary header is presented in Section B.1 in 0, and is not compliant with ECSS PUS proposed secondary header format. Also, some services in cFS are done through the uplink of table files, as defined in Section A.6 of APPENDIX A, and not necessarily through TC packets only. The counterpart TM reports are sometimes cFE event messages, information dump to files (that should be later downloaded to ground) or should be inferred through user-requested TM reports.

This makes cFS not compliant with ECSS PUS proposed packet formats of ECSS (2016) in terms of headers and application data field content. Nevertheless, from the services perspective only, many similarities can be found, as summarized in Table 5.2.

Table 5.2 – ECSS PUS Services correspondence with cFS applications.

PUS Service		cFS
ID	**Name**	**cFS Application or Function**
1	Request Verification	cFE EVS API's and cFS application template Partially compliant
2	Device Access	Not implemented
3	Housekeeping	cFS SCH and TO Partially compliant
4	Parameter Statistics Reporting	Not implemented
5	Event Reporting	cFE EVS Compliant
6	Memory Management	cFS MM Partially compliant
7	*[reserved]*	N/A
8	Function Management	e.g. cFS wake_up commands Compliant
9	Time Management	cFE TIME Compliant
10	*[reserved]*	N/A
11	Time-based scheduling	cFS SCH Partially compliant
12	On-board monitoring	cFS LC and cFS HS Compliant
13	Large Packet Transfer	cFS CF Compliant
14	Real-time Forwarding Control	cFS TO Compliant
15	On-Board Storage and Retrieval	cFS DS Partially compliant
16	*[reserved]*	N/A
17	Test	cFS apps contain NOOP commands Compliant
18	On-board Control Procedure (OBCP)	Not implemented

PUS Service		cFS
ID	**Name**	**cFS Application or Function**
19	Event-Action	cFS HS Compliant
20	Parameter Management	cFE TBL Partially compliant
21	Request Sequencing	cFS SC Compliant
22	Position-Based Scheduling	Not implemented
23	File Management	cFS FM and cFS CF Partially compliant

Source: Made by the author.

APPENDIX C brings more details about the compliance analysis summarized in Table 5.2.

For comparison purposes, GERICOS framework presented in Section 3.3.1 implements Services 1, 3, 5, 6, 9 and 17. These services are the ones usually required at payload level in ESA missions (PLASSON et al., 2016). Also, SAVOIR/COrDeT and CAST frameworks implement PUS services, as mentioned in Sections 3.3.3 and 3.3.5.

It is not necessary for a mission to fully implement PUS services in order to be successful or operable, as recognized by Clerigo et al. (2018). Each mission must tailor the standard to its operational requirements.

The analysis of cFS conformance to the most used FSW services and data protocol standards provides to the cFS developers insights on the framework built-in functionalities. cFS users can then focus on non-standardized functionalities, which will be subsequently exposed in Chapter 6.

6 MISSION-SPECIFIC APPLICATIONS DEVELOPMENT

6.1. Overview of FSW design effort in cFS-based missions

As already presented in cFS architecture described in Sections 3.3.2 and 4.2, cFS framework provides several generic FSW built-in functionalities, based on NASA GSFC experience and legacy missions. These functionalities are in strong correlation with the most used FSW standards, as exposed in Chapter 5, 0 and APPENDIX C.

In a mission that uses cFS, FSW developers typically work on the following areas: Boot and BSP software; low-level software drivers; OSAL and PSP port; cFE/cFS core applications tunning; and Development of mission-specific cFS applications.

Boot and BSP software development is normally a task performed by OBC vendors and is cFS agnostic, but occasionally there is some necessary tailoring to be performed by mission FSW developers.

Low-level software drivers development is an activity that is partially dependent on the operating system, partially provided by OBC and equipment vendors, and some design is still made by FSW developers. There is a great dependence on hardware definitions, which makes it hard to standardize this part.

OSAL and PSP port to the target processor and operating system, despite requiring considerable engineering effort, is an area that can be delegated to 3rd-party, and eventually will be done by NASA or the cFS community due to growing framework number of users and subsequent cFS-developers demands for support. At the end, there is a small number of embedded operating systems and processors of interest.

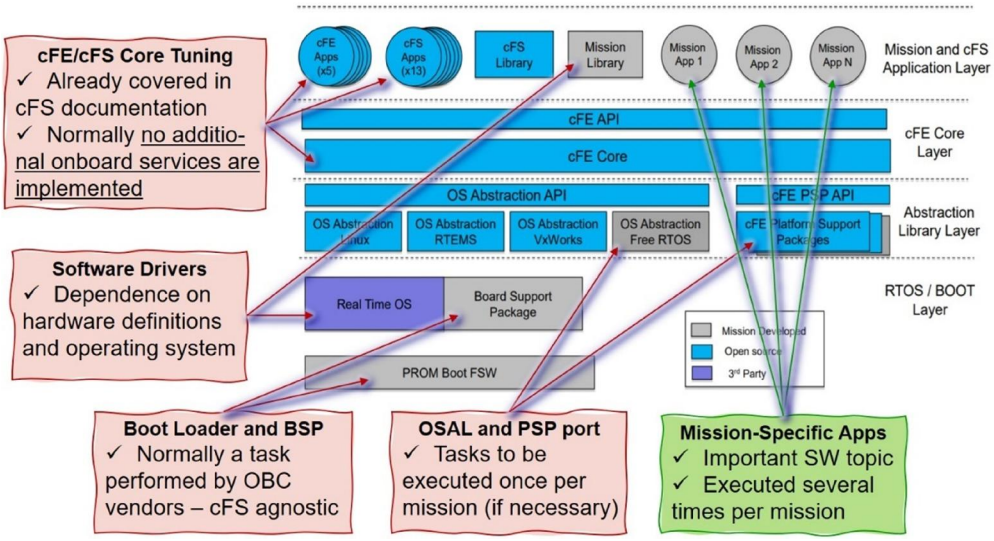
cFE and cFS core applications parameters tunning is a delicate activity that has great correlation with mission definitions. It is complicated to develop a process or methodology to standardize this area. Furthermore, cFS documentation already brings information on that subject.

On the other side, development of mission-specific cFS applications and libraries to accomplish the mission's tasks will still be a central software

topic because satellite missions can largely vary in objective and complexity, even though cFS applications keep the same core architecture. This topic is the subject of analysis along this Chapter.

Figure 6.1 summarizes the development effort associated with each cFS-based FSW layer or component.

Figure 6.1 – cFS-based FSW components and layers with respective comments regarding their development effort.



This figure summarizes the development effort associated with each layer or component of a FSW that uses cFS framework. Note that “Mission-Specific Apps” development is highlighted (in green box) as the topic of interest, because this is the area that the author identified as the best candidate for standardization.

Source: Author adapted from NASA (2014).

6.2. **Motivation for a systematic development of cFS applications**

While creating new mission-specific cFS applications for the reference mission, it was observed a great commonality among them. They have a similar architecture, coding schemes and its basic functions make use of recurring cFE core API's.

Also, when creating new cFS applications, due to the reasonable amount of mandatory variable initializations and API calls in specific parts of the algorithm, it isn't hard to eventually forget some of them, causing considerable waste of time on troubleshooting and benchmarking with other "well-behaved" applications.

In order to help avoiding such problems, this dissertation brings, as part of the proposed "New Space" approach for FSW, design rules that must be observed on cFS applications coding and design

These rules were formulated based on the author analysis on inherited cFS applications, space software standards guidelines and the experience gained when writing new applications for the reference mission.

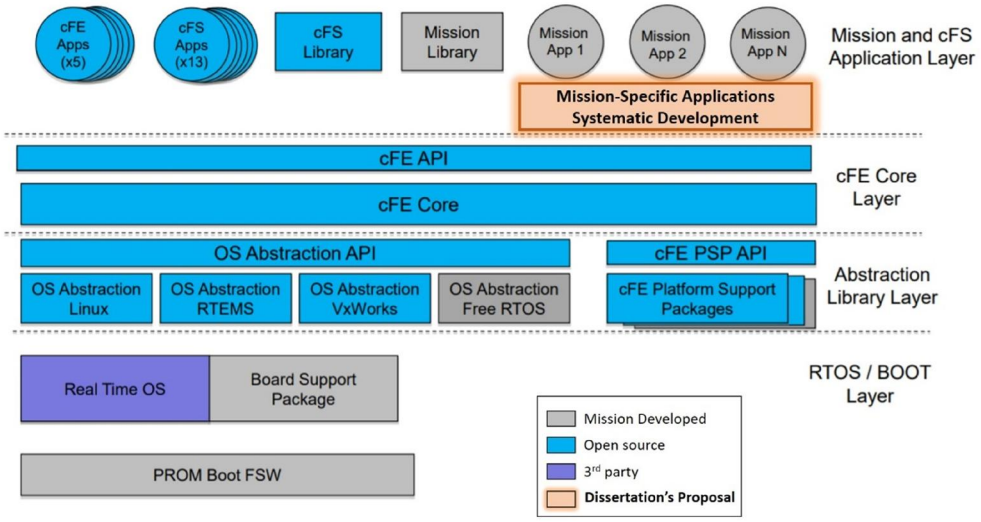
As a return of experience from the reference mission, the adoption of design rules by itself wasn't immediately followed by quality increase on new cFS applications development. It was necessary to create a tool that facilitated the employment of these rules, sparing the designer of memorizing and manually inspecting their adoption on his software products. The proposed tool is a tailorable cFS applications template generator.

This systematic approach for cFS applications development, consisting of design rules creation, adoption and the use of a corresponding cFS template generator, can be abstracted with respect to the standard cFS architecture as shown in Figure 6.2. This figure is an adaptation of Figure 6.1, already described in Section 3.3.2, but adds the proposed systematic development approach as the orange box under the mission-specific apps.

This systematic development objectives are:

1. Improve new cFS applications quality, predictability and conformance to the standards through a well-defined set of design rules.
2. Help to quickly prototype cFS applications that are in conformance with the proposed design rules.

Figure 6.2 – NASA cFS Software Layers and Components along with proposed applications development abstraction.

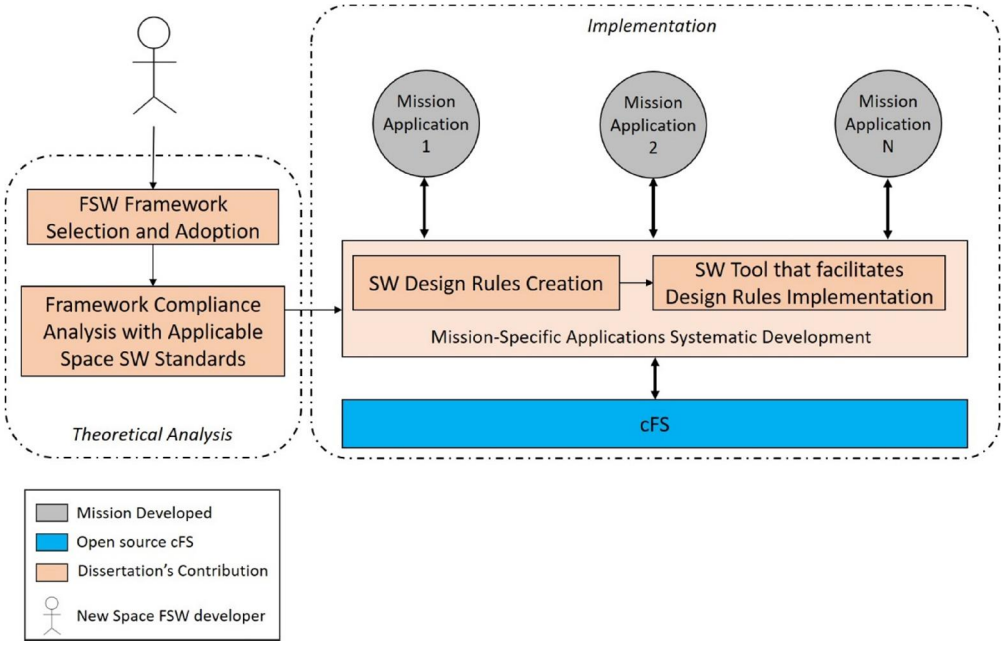


The dissertation's cFS applications development approach is located in the "Mission and cFS Application Layer" level, highlighted in orange.

Source: Author adapted from NASA (2014).

Figure 6.3 shows this dissertation's overall contribution to cFS-compliant FSW development, from framework selection up to applications systematic development.

Figure 6.3 – The dissertation's contributions for a cFS-based FSW project.



The dissertation's contribution is presented as a sequential and logical chain, with theoretical contributions from Chapters 3, 4, 5 and the implementation/"hands-on" contribution from Chapter 6.

Source: Made by the author.

6.3. **Proposed design rules for new cFS applications**

Design rules were created to deal with the complexity of creating cFS applications in the reference mission. Differently from the architectural rules presented in Section 4.2.5, the rules proposed herein are more closely related to cFS applications implementation.

Table 6.1 organizes all created design rules, their description, rationale and categorization.

Most of them were abstracted from cFS applications suite detailed study and cFS Sample application inference and therefore their rationale can be described as "cFS best practices" or "registering or handling of core services". Their use is consecrated in cFS framework and they are

deemed to be operationally significant and are, therefore, kept as legacy design rules.

There are also original design rules that were created in order to prevent development errors, uniformize application schemes and help to improve software overall quality. They consist of both guidelines provided by space software standards presented in Chapter 5 and lessons learned from the reference mission. Each rule is individually justified in Table 6.1.

The design rules were categorized into two groups: “basic rules” and “tailorable patterns”. The basic group congregates design rules that can be immediately codified in every cFS application, as they require no particular tailoring. Examples of basic rules are some mandatory cFE API calls and performance benchmarks placement.

The second group, tailorable patterns, consists of rules whose implementation may vary in each cFS application. Nevertheless, they can still be guided by implementation examples, or patterns, to aid developers’ job. Examples of these rules are commands implementation and tables management.

In terms of implementation, next section will present “Dan Templates Wrapper” (DTW), a tool that facilitates implementation of both design rules categories. Basic rules are immediately codified in the templates that are made available by DTW. Tailorable patterns are also codified by DTW, providing implementation examples and pointing out through comments later customizations to be done by the software developer.

Table 6.1 – Proposed Design Rules for new cFS Applications.

Number	Name	Description	Rationale	Category
DR-010	cFS Application Template	The cFS application shall be generated from "Dan Templates Wrapper" (DTW).	Minimum viable cFS-compliant application, containing mandatory calls that guarantee integration with rest of cFS and that is operationally significant. Available at GitHub page https://github.com/DaniloJFMiranda/OpenSatKit.	Basic rules
DR-020	cFE APIs Calls	The cFS application shall make use of cFE, OSAL and PSP available APIs as presented in "cFE - FSW Application Developers Guide v5.4" reference document, instead of referentiating directly the underlying operating system.	Integration of these applications with cFS lower layers. Also, direct calls to the Operating System API's will prevent the FSW modularity and portability.	Basic rules
DR-030	cFE ES Registering	The cFS application shall be successfully registered in cFE ES services in order to be started and be integrated in runtime with cFE core.	Mandatory cFE call.	Basic rules
DR-040	App Data Struct	The cFS application shall organize all its relevant internal variables in a comprehensive app data struct	A single global data struct facilitates data retrieval in memory and optimizes memory allocation efficiency.	Tailorable patterns
DR-100	App Initialization and Registration	The cFS application shall perform application specific initialization procedures after cFE ES Register and before application main process loop	cFS application's initialization and registration procedure that starts all app variables and necessary functions before the application main process loop.	Tailorable patterns
DR-110	Critical Variables Initialization in Power-On	The cFS application shall initialize its critical variables with initial values in case of power-on start, if application is CDS-compliant.	cFS best practice.	Tailorable patterns
DR-120	App Data Struct Initialization	The cFS application shall initialize application data struct variables in application initialization.	cFS best practice.	Tailorable patterns

Number	Name	Description	Rationale	Category
DR-130	cFE EVS Initialization and Registering	The cFS application shall initialize its event filter table and register to cFE EVS core services.	In order to be able to send event messages as per PUS 5 service and also to filter EVS messages according to user-defined criteria.	Tailorable patterns
DR-140	cFE SB Message Pipe Creation	The cFS application shall create a software bus message pipe and subscribe to the messages that it will be supposed to receive.	This is the way in cFS for the applications to receive command messages using cFE SB services.	Basic rules
DR-150	cFE TBL Registering and Loading	The cFS application shall register its tables and successfully load them using cFE TBL services, if application has tables	This is the way in cFS to register for using cFE Table services. This requirement is only applicable for table-driven applications.	Tailorable patterns
DR-160	cFE ES CDS Data retrieval or initialization	The cFS application shall restore its critical data, if existent, or register to cFE ES CDS services during initialization procedures, if application is CDS-compliant.	This is the way in cFS to register for using cFE ES CDS services, in order to recover critical data in case of processor reset. This requirement is only applicable for applications that make use of these services.	Tailorable patterns
DR-170	Application Version Identifier in initialization	The cFS application shall output an informational event-message containing application version after a successful initialization process.	cFS best practice.	Basic rules
DR-200	Application Main Process Loop	The cFS application shall contain a main process loop, where it shall remain indefinitely after successful initialization if cFE ES services correctness is confirmed every loop.	Mandatory cFS applications structure. cFS applications should be nominally killed only by cFE ES service. Also, this is the exit point for cFS applications.	Basic rules
DR-210	Idle State Indefinitely	The cFS application shall remain in idle state indefinitely while waiting for a software bus message reception, in case of asynchronous or soft periodic applications.	In soft periodic applications case, cFS SCH app shall contain the respective WAKE_UP message that will provide the pace to the desired cFS application.	Basic rules

Number	Name	Description	Rationale	Category
DR-220	Idle State up to Time-Out	The cFS application shall remain in idle state up to a user-defined time-out figure while waiting for a software bus message reception, in case of time-out periodic applications.	Benchmark extracted from cFS Suite time-out applications.	Basic rules
DR-230	Noop Command	The cFS application shall be capable of receiving a no-operations command from the ground	A NOOP function is basically to "ping" a certain application to attest its aliveness by increasing the APP's command counter. This is aligned with ECSS PUS 17.	Basic rules
DR-240	Application Version Identifier in TM	The cFS application version shall be outputted in an informational event-message when the Noop command is received.	cFS best practice.	Basic rules
DR-250	Command Success Counter	The cFS application shall maintain a command success counter that is increased when a ground command message is successfully received.	cFS best practice.	Tailorable patterns
DR-260	Command Error Counter	The cFS application shall maintain a command error counter that is increased whenever a command message is not successfully received or processed.	cFS best practice. The error sources could be command incorrect length or any invalid parameters that are out of range or incompatible with the expected TC profile.	Tailorable patterns
DR-270	Reset Command	The cFS application shall have a reset command which causes an application's internal counters to reset to zero.	cFS best practice.	Basic rules
DR-280	Bad Command Rejection	The cFS application shall validate the format and length of each received command and reject the command if the validation fails.	cFS best practice.	Tailorable patterns

Number	Name	Description	Rationale	Category
DR-290	Ground Commands	The cFS application shall be able to receive commands from the ground and respond to them as defined in the applications operations profile.	This is one of the tailoring points that missions shall work on.	Tailorable patterns
DR-300	Housekeeping Request Command	The cFS application shall be able to periodically receive housekeeping request commands from the scheduler application.	cFS best practice.	Basic rules
DR-310	WakeUp Command	The periodic cFS applications shall be able to periodically receive wakeup commands from the scheduler application and respond to them activating one or a fixed number of cycles of their internal functions, in case of soft-critical or time-out periodic applications.	cFS best practice.	Tailorable patterns
DR-320	Housekeeping Packet	The cFS application shall be able to send their housekeeping telemetry packets after a successful reception of a housekeeping command.	cFS best practice. Housekeeping messages are expected to be periodically transmitted by cFS applications. They should contain minimum application operationally significant information.	Tailorable patterns
DR-330	Event Messages Reporting	The cFS application shall report debug, informational, error and critical messages through EVS services.	cFE Event Service that are based on ECSS PUS 5 service.	Tailorable patterns
DR-340	Command Reception Acknowledgement	The cFS application shall generate a debug or informational event-message to attest a command correct reception.	cFS best practice.	Tailorable patterns

Number	Name	Description	Rationale	Category
DR-350	Command Reception Error	The cFS application shall generate an error event-message to attest a command incorrect reception, containing an error code that identifies the root cause.	cFS best practice. The error sources could be command incorrect length, MID or any invalid parameters that are out of range or incompatible with the expected TC profile.	Tailorable patterns
DR-360	Command Partial Execution	The cFS application shall generate debug event-messages to attest a command partial execution status report whenever the command effect is not immediate.	Commands that trigger actions that can take a long time on-board (like cFS FM Copy File) should issue progress event messages. This is a lesson learned from ECSS PUS 5 service.	Tailorable patterns
DR-400	Performance Benchmarks	The cFS application shall make use of cFE ES performance benchmarks.	This allows to access cFS application CPU usage using a same uniform API.	Basic rules
DR-410	Performance Benchmark entry after application register	The cFS application shall have a performance benchmark entry after application registry in initialization procedures.	FS best practice.	Basic rules
DR-420	Performance Benchmark entry after cFE SB receive message API execution	The cFS application shall have a performance benchmark entry after cFE SB Receive Message API call	cFS best practice.	Basic rules
DR-430	Performance Benchmark exit before cFE SB receive message API execution	The cFS application shall have a performance benchmark exit before cFE SB Receive Message API call	cFS best practice.	Basic rules
DR-440	Performance Benchmark exit before application main function end	The cFS application shall have a performance benchmark exit before cFE ES Exit API call, in the end of the application main function	cFS best practice.	Basic rules

Number	Name	Description	Rationale	Category
DR-500	Table Management	The cFS application shall make use of cFE Table APIs in order to manage binary files that can be changed in runtime.	This is part of cFS architecture. Requirements in this section are only applicable for table-driven applications.	Tailorable patterns
DR-510	Table Update in Periodic Applications	The cFS application shall check for table updates during its periodic cycles if it's a time-out or soft-critical periodic application.	cFS best practice.	Tailorable patterns
DR-520	Table Update in Asynchronous Applications	The cFS application shall check for table updates whenever receiving a "send housekeeping" request if it is an asynchronous application.	cFS best practice. In case of asynchronous applications, "Send HK" message is maybe the only periodic request that the app possesses. cFS applications may take advantage of that to send their housekeeping packets to the ground.	Tailorable patterns
DR-530	Table Validity Functions	The cFS application shall validate a new table against user defined sanity check functions, if applicable, before turning it operational.	cFS best practice. Table validation function is a topic that has to be tailored depending on the user-defined table.	Tailorable patterns
DR-600	CDS Data Update	The cFS application shall update its critical data using cFE CDS services whenever there is critical data update available.	cFS best practice. For safety purposes. This requirement is only applicable for CDS-compliant applications.	Tailorable patterns

Source: Made by the author.

6.4. **cFS "Dan templates wrapper" (DTW)**

As explained in previous sections, the existence of the 39 design rules presented in Section 6.3, by itself, didn't satisfactorily improve software design lifecycle agility and quality as expected in the reference mission. "New Space" missions typically have tight schedule and budget, so there is always the counter argument that too many rules, especially those not tied to the code, are a burden for software developers.

A pragmatic and quick solution to address this problem was proposed and very well accepted in the reference mission, being to use a cFS templates generator tool to aid in the proposed design rules implementation.

An immediate challenge was encountered in the beginning of the templates design, though: a single application template would not be enough to deal with the variability of the created design rules, especially the "tailorable patterns" category, described in Table 6.1.

6.4.1. cFS template types

The consistent use of legacy NASA cFS applications, presented in Chapter 4.2.4, and the design rules implementation in the reference mission led to the perception that a few types of cFS application templates would be convenient for FSW design.

They are very similar in their core, but contain differences depending on the application scheduling, critical data storage and tables processing needs, which are:

1. Periodicity: Time-Out Periodic, Soft-Critical Periodic or Asynchronous application.
2. Existence or not of CDS (Critical Data Storage) for storing persistent data in case of computer processor reset.
3. Existence or not of cFS Tables, which are binary files that organize information in tables or matrices, being able to be changed in run-time

Time-Out periodic refer to the applications that do not rely exclusively on cFS Scheduler in order to receive a periodic pace or wake up message. Instead, they count on cFE Software Bus time out service, which is more reliable. They execute periodic tasks that shall carry on even on cFS Scheduler failure. The Command Ingestor (CI) and the Telemetry Output (TO) applications are typical examples of time-out periodic applications.

Soft-Critical periodic refer to the applications that do rely only on cFS Scheduler in order to receive a periodic pace or wake up message. This strategy allows to stop application periodicity or even change the periodicity rate according to a given spacecraft operational phase. If cFS Scheduler temporarily fails, the soft-critical periodic applications will not receive expected messages and therefore will not cycle, but due to their nature it is not considered a catastrophic event. An example of such type of application is CFDP app (CF).

Asynchronous applications refer to those that do not have periodic tasks and are only awaken when there is a ground command or a send housekeeping request. The File Manager (FM) application is a typical example.

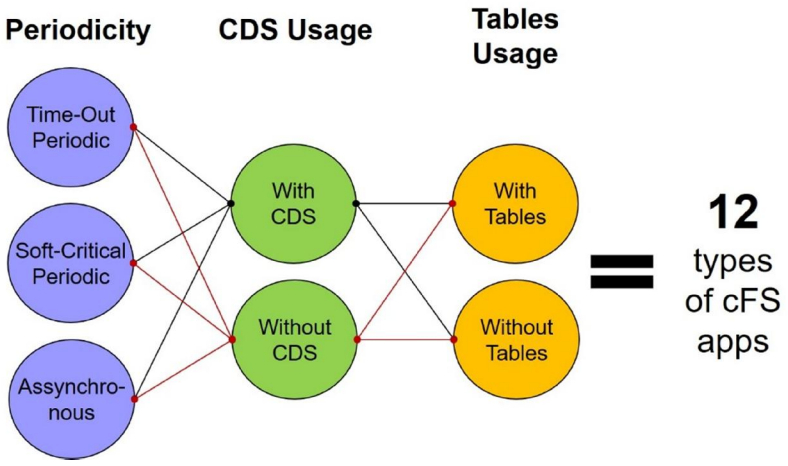
CDS is a cFE Executive Service (ES) optional service, as shown in Section A.3, which preserves user-defined critical data in case of computer reset. Applications that make use of it may call a few APIs in certain application places.

cFE Table is also an optional service that applications might or not use. The benefits of using it were shown in Section A.6. If chosen to be applied, table-driven applications should correctly call cFE Table API's.

The correct use of cFE API's and the basic structure of a cFS application were derived from Bartholomew and Kobe (2014). In this reference, there is a generic cFS template called "QQ_app", which formed the basis for the dissertation's proposed templates. QQ_app however doesn't take into account the different applications' scheduling needs, and the presence or not of CDS and Tables.

These points of variability lead to 12 types of cFS application templates, as shown in Figure 6.4.

Figure 6.4 – The 12 types of cFS application templates rationale.



This figure shows the combinatorial analysis that leads to the 12 required cFS applications templates.

Source: Made by the author.

In order to confirm that indeed these 12 templates were useful, the 17 legacy NASA cFS high-level applications presented in Section A.9 were classified according to these templates. Among the 12 identified types, 9 had at least one representative, as shown in Table 6.2. This shows that in fact it is advantageous to have available all 12 foreseen templates.

Table 6.2 – NASA cFS legacy applications classified according to the proposed template types.

#	cFS Application Type	NASA Legacy Applications
1	Time-Out Periodic with CDS storage with tables	HS
2	Time-Out Periodic with CDS storage without tables	--
3	Time-Out Periodic without CDS storage with tables	TO
4	Time-Out Periodic without CDS storage without tables	CI, CI_lab, SBN, TO_lab*
5	Soft-Critical Periodic with CDS storage with tables	CS, LC
6	Soft-Critical Periodic with CDS storage without tables	--
7	Soft-Critical Periodic without CDS storage with tables	CF, HK, MD, SC
8	Soft-Critical Periodic without CDS storage without tables	SCH_lab
9	Asynchronous with CDS storage with tables	DS
10	Asynchronous with CDS storage without tables	--
11	Asynchronous without CDS storage with tables	FM
12	Asynchronous without CDS storage without tables	MM

Source: Made by the author.

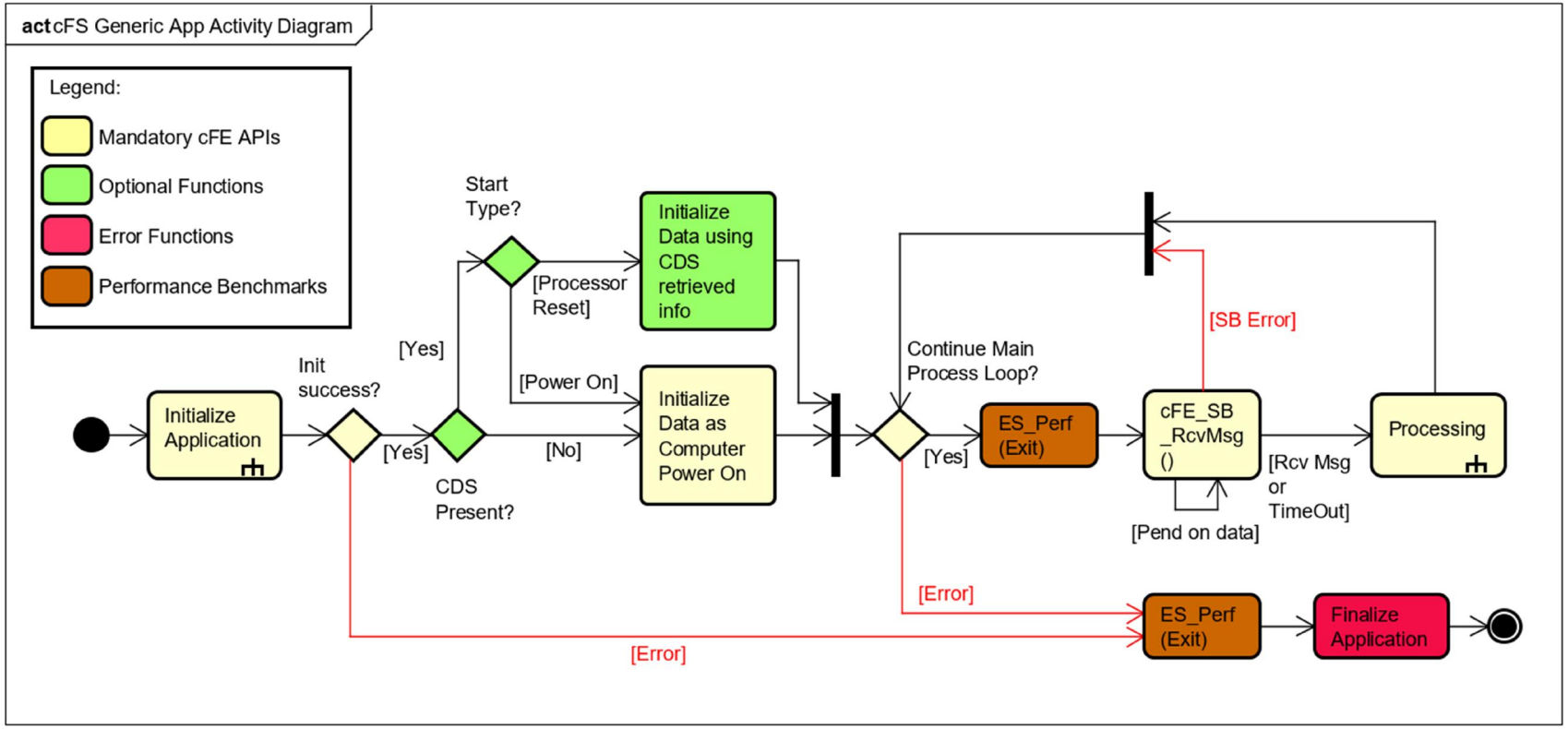
* TO_lab application can be considered a time-out periodic application, but with a different implementation, which makes use of OSAL task delay API.

cFS Scheduler (SCH) was the only cFS suite application that could not be categorized according to the criteria proposed in this work. This is expected because typically mission schedulers use operating system semaphores or equivalent functions to guarantee its periodicity.

6.4.2. cFS applications UML model

The UML activity diagram of the proposed generic cFS application template is shown in Figure 6.5 and Figure 6.6.

Figure 6.5 – cFS Generic Application UML Activity Diagram.



This figure shows cFS Generic Application functional diagram. Functions in green are optional, benchmarks are in brown and errors in red.

Source: Made by the Author.

Every cFS application starts with “Initialize Applications” procedures, which are responsible for registering to the cFE core services and creating a global data struct.

After successful initialization, the application “Initialize Data” with hard-coded defaults if a computer power on happened, or from cFE ES CDS if critical data was preserved.

Then, the application goes to an infinite loop (“Main Process Loop”) where it is supposed to stay for the entire mission if there is no error or deliberate operator command to kill that app. First thing in this loop is to check if there are incoming command messages (cFE_SB_RcvMsg).

This Software Bus API is the basic mandatory function that makes an application wait until data income, poll a certain pipe or channel for messages and, in case of time-out periodic applications, to establish a time-out to carry on its tasks even if no messages are received.

Subsequently to cFE_SB_RcvMsg(), the application will iterate a cycle of the “Processing” activity. “Processing” contains all commands, periodic tasks, telemetry generation and parameter updates that the application performs. After this cycle, the main process loop is restarted.

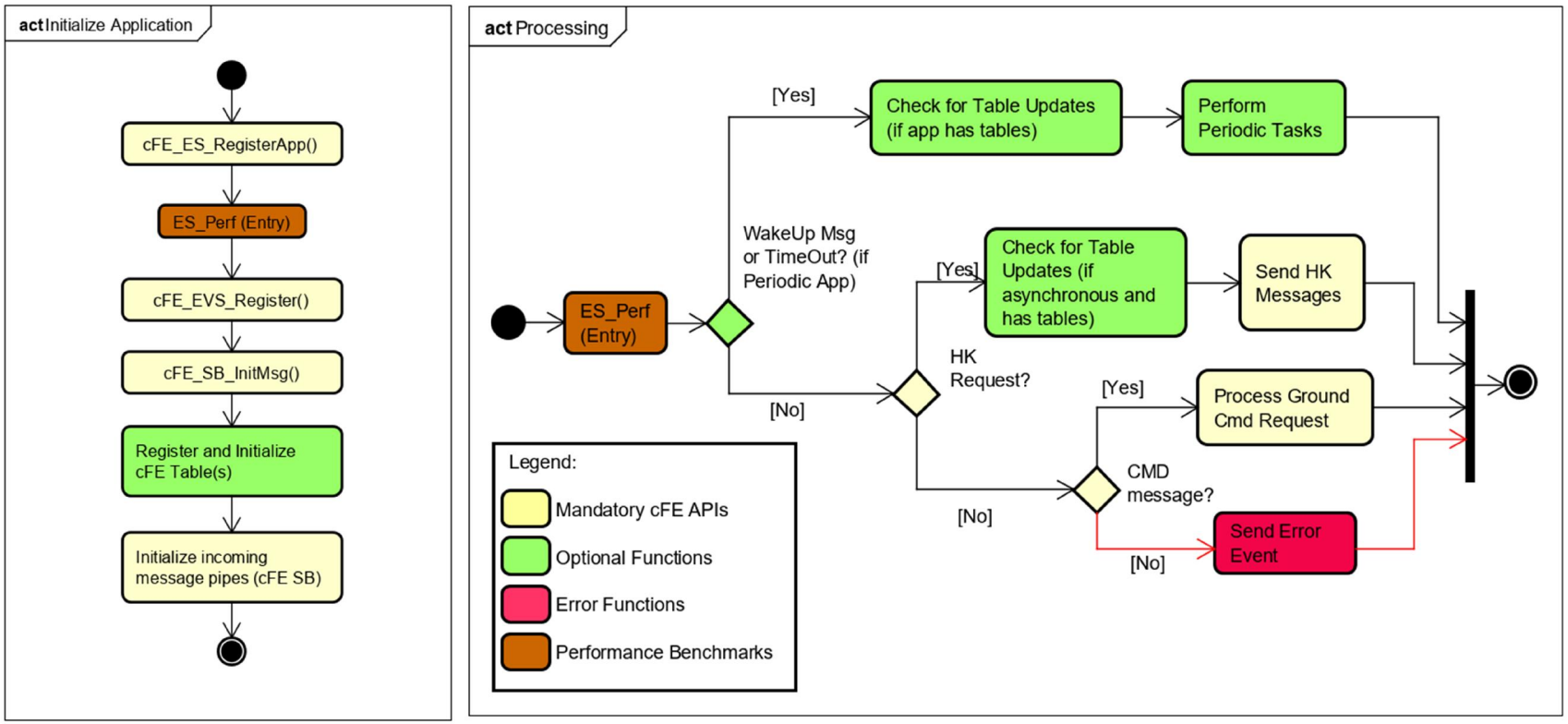
Figure 6.5 and Figure 6.6 color code is further detailed below:

- Activities in yellow are mandatory APIs calls or functions in every cFS application.
- Activities in green are optional functions that should be called if application is table-driven or CDS-driven.
- Activities in red are mandatory APIs calls or functions in every cFS application that represent the path in case of errors.
- Activities in brown represent the location of cFE ES performance benchmark APIs. These benchmarks generate time-stamped events that can be later post-processed to verify the application CPU usage time.

Many cFE API's that are not mandatory in a cFS-compliant application were not used on the proposed template. They should be used by FSW developers on a case-by-case analysis.

Two exceptions are cFE Table and CDS APIs, which are optional cFE APIs that are recurrently used on several cFS applications and therefore were adopted in the proposed tool. In order to assess their applicability to a new cFS application, a small questionnaire is queried to the developer, before the application creation, as will be explained in Section 6.4.4.

Figure 6.6 – cFS Generic Application UML Activity Diagram – 2nd level.



This figure details the cFS Generic Application Activity Diagram by means of expanding “Initialize Application” and “Processing” activities.

Source: Made by the Author.

6.4.3. DTW tool presentation

OpenSatKit SDK, already described in Section 4.6, contains a generic cFS application generator that implements some proposed templates. None of them takes into account the variability explained in Section 6.4.1, but they can be used as start point for developing DTW.

OpenSatKit GitHub project[11] was then forked (i.e. copied) to the author's proprietary GitHub page[12], which is also open source. This was done to facilitate users to check and compare the modifications made on the original OpenSatKit code. A local Git version was finally cloned on an Ubuntu Virtual Machine for local work, compilation and testing.

The design activity was performed exclusively on the OpenSatKit cFS create app tool version 1.0, part of cFS starter kit branch, which was written in the ground control system tool COSMOS. The rest of the legacy code was preserved.

One main advantage to work with this NASA SDK is that it provides a fully functional simulated closed loop environment that contains a FSW instance that runs cFS and a ground control software that can send commands and scripts to test and operate the FSW. This closed-loop setup is sufficient to validate the generated cFS applications.

Two main activities were performed in the template generator design: template generator front-end and the 12 cFS application templates skeleton elaboration.

OpenSatKit already provides 5 template options to the user, as shown in Figure 6.7. These templates, however, didn't provide the necessary flexibility for the reference mission and, if adopted, would imply in applications manually tailored after creation.

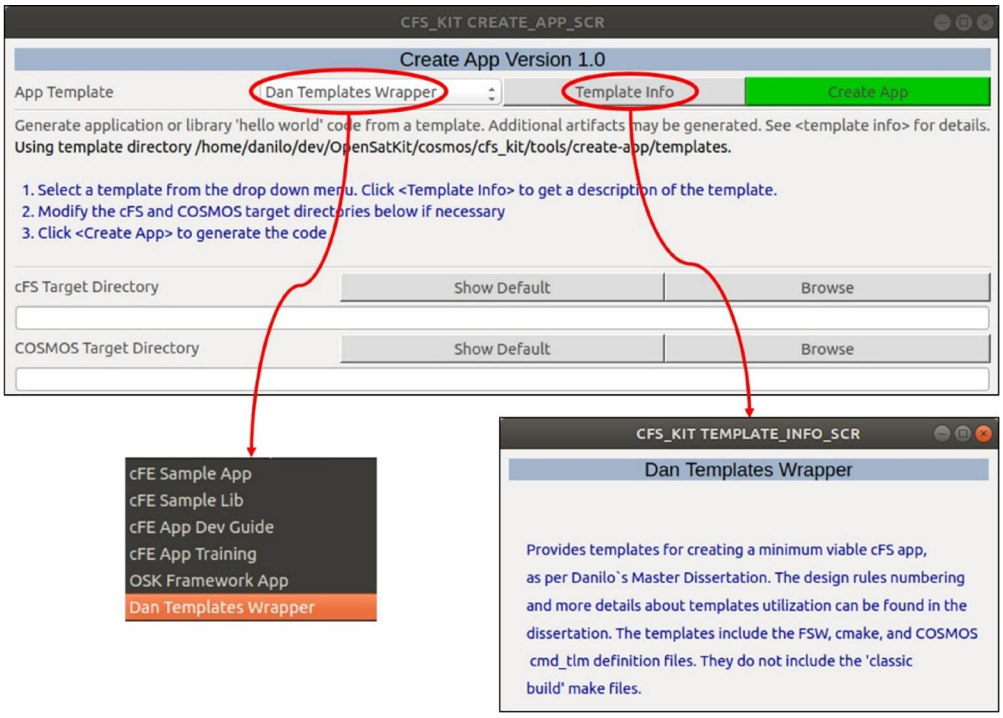
The front-end effort consisted of first modifying the already existing "Create App" tool to accommodate a new template configuration, as

[11] https://github.com/OpenSatKit
[12] https://github.com/DaniloJFMiranda/OpenSatKit

shown in Figure 6.7. DTW is not itself a template, but a wrapper that can lead to 1 of 12 possible templates, also following the design rules described on Table 6.1.

Therefore, when selecting “Dan Templates Wrapper” and clicking on “Create App” button, as shown in Figure 6.8, another screen is presented to the user, containing a few checkboxes that will gather information to pick up the right cFS template among the 12 possible options listed in Section 6.4.1.

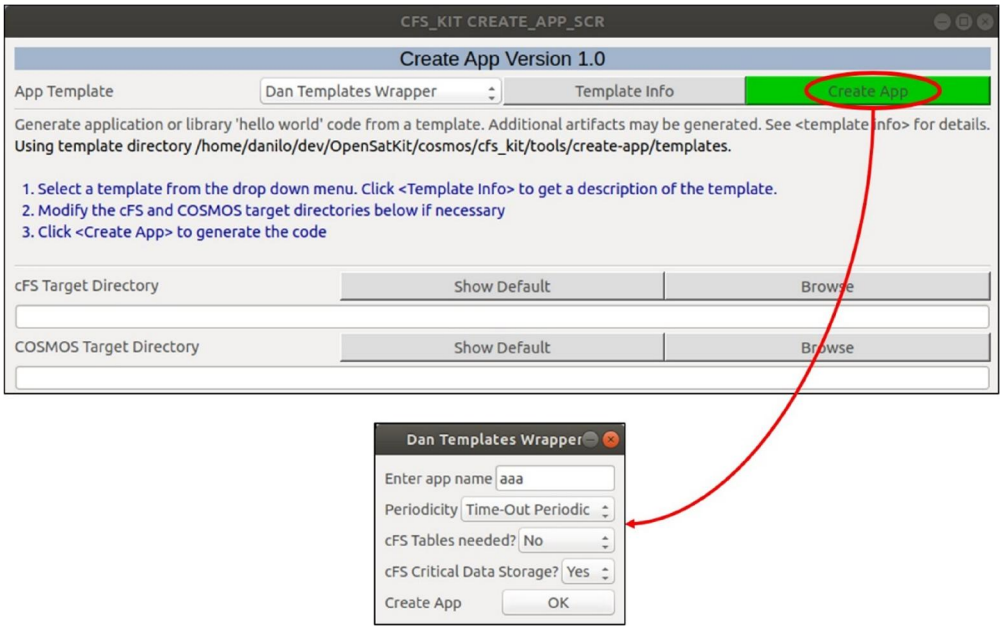
Figure 6.7 – OpenSatKit Create App screen with “Dan Templates Wrapper” selection.



This is a printscreen from NASA’s OpenSatKit Create App tool version 1.0 front-end. DTW is a new feature added in there.

Source: Made by the author.

Figure 6.8 – “Dan Templates Wrapper” questionnaire.



DTW creation pops-out another screen to the user, which is a questionnaire that will gather information to pick up the right cFS application template.

Source: Made by the author.

The second activity regarding DTW design was the elaboration of the 12 application templates prototypes. They were implemented according to the UML profile presented in Section 6.4.2. Design rules were annotated as comments along the code. This is particularly important in the case of the “tailorable pattern” rules, in order to point out to the developer the parts of the software that must be later customized.

Finally, OpenSatKit code modifications made due to DTW addition were submitted as a git pull request to the NASA’s master OpenSatKit repository and are waiting for review and approval.

6.4.4. DTW questionnaire and apps creation

A small questionnaire is proposed to the cFS application developer before the automatic template generation. It contains the following questions:

- App name: The application chosen name is used along the application template in several variables and functions declarations.
- Periodicity: A checkbox with three options will appear to the developer, being “Hard Periodic”, “Soft Periodic” and “Asynchronous”. Their meanings were explained in the beginning of Section 6.4. Basically, this function will change cFE_SB_RcvMsg() function arguments, and place periodic tasks and table management calls accordingly (if application is periodic or tables are present).
- cFS Tables Needed: A checkbox with two options will appear to the developer, being “Yes” or “No”. cFE Table APIs will be placed or not in the cFS application template depending on user choice. Also, table management depends on application periodicity.
- cFS CDS: A checkbox with two options will appear to the developer, being “Yes” or “No”. cFE ES CDS APIs will be placed or not in the cFS application template depending on user choice.

After the developer has filled the check boxes with the corresponding information and authorizes application creation, the algorithm automatically picks up one of the 12 developed templates depending on the choices and fills it with the application chosen name in the markups. All necessary files for the minimum viable cFS app are created in a folder in the same directory of existing applications.

6.4.5. DTW verification strategy in the reference mission

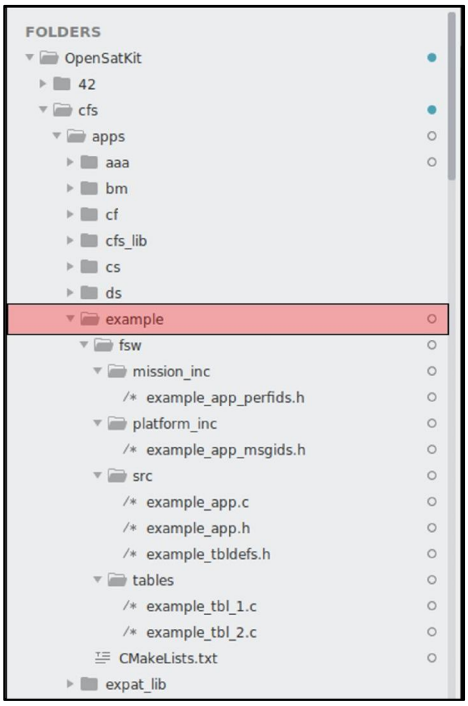
As explained in Section 6.4.3, NASA OpenSatKit SDK brings all necessary features to test the implementation of the proposed templates. The verification strategy for DTW in the reference mission was performed in 4 steps:

1. Generating an application source code from the existing templates and performing visual inspection to check if all expected files were correctly created.
2. Compiling the generated application source code into cFS application object files. At this moment, check for errors or warnings that the compiler may inform.
3. Running the FSW including the new cFS application. This will account for compatibility with cFS lower layers and discover eventual runtime issues.
4. Performing operability and integration tests in the new cFS application sending the available telecommands and checking for prompt messages and telemetry status change using OpenSatKit ground control software.

Applications generation (step 1) was presented in Sections 6.4.3 and 6.4.4. After the developer has filled the questionnaire, the minimum viable corresponding cFS application files are generated in the newly created app folder. Figure 6.9 shows an example of directory structure for an example app, that was created using template #9. A visual inspection can be performed in the files created inside example app folder to check if there are no files missing and also for files correctness. In example app particular case, there were no errors.

Next, the generated application is compiled into cFS object files and table binary files (if present). This is performed using cFS native CMake tool. Figure 6.10 shows the resulting files of example app compilation.

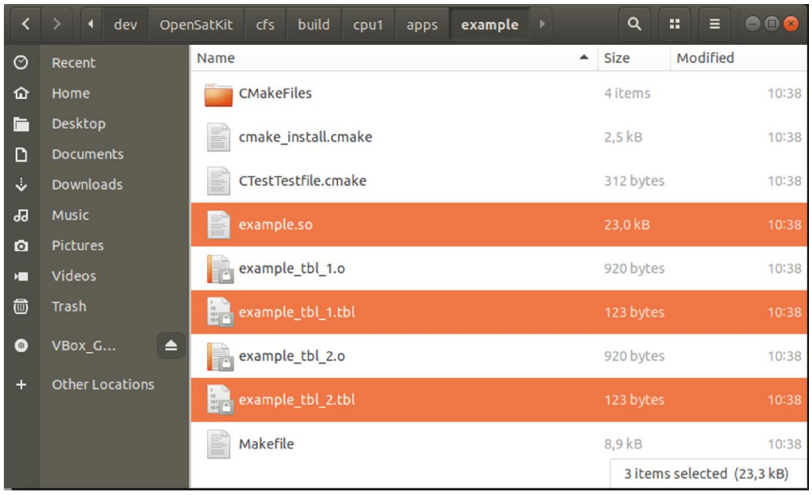
Figure 6.9 – Example application directory structure.



Example application source code is created in /cfs/apps in OpenSatKit directory. Inside this folder, there are all necessary files to have a minimum viable app.

Source: Made by the author.

Figure 6.10 – Example application after compilation.



Example app compilation files (application object file and table files).

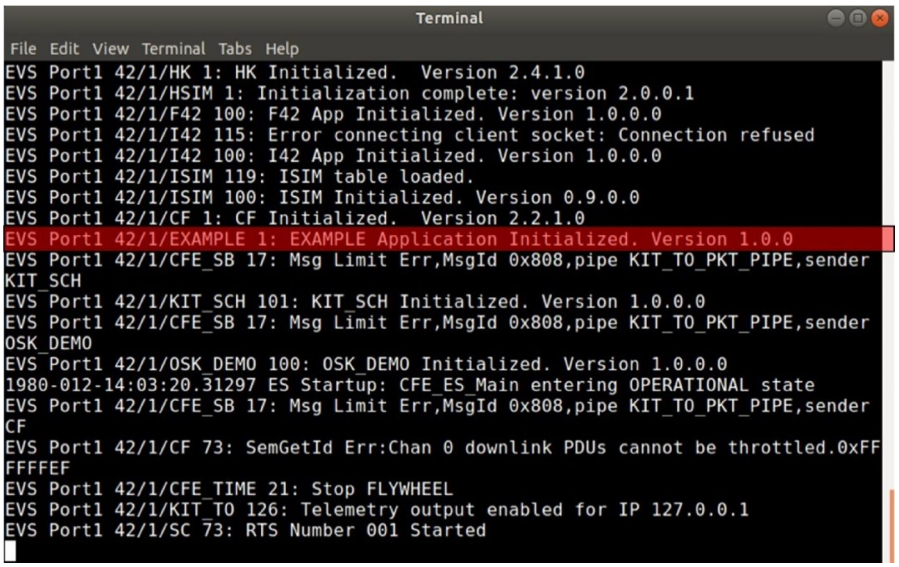
Source: Made by the author.

The object file “example.so” is the result of compilation of “example_app.c” source code plus its header files. The table files “example_tbl_1.tbl” and “example_tbl_2.tbl” are the result of compilation of “example_tbl_1.c” and “example_tbl_2.c” respectively.

The compiler showed no warnings or compilation errors for this particular example. If any errors were noticed in compilation time, the developer should go back to the source code and correct the issues found.

Subsequently, cFS-based FSW is run with the new applications. In the particular case of example_app, Figure 6.11 presents the FSW terminal with an event message attesting the correct initialization of the application, as per design rule DR-170.

Figure 6.11 – FSW terminal with cFS core running along with example app.



Example app running in cFS environment. The highlight shows the cFE EVS Event Message that is issued after example_app correct initialization.

Source: Made by the author.

In addition, when an application is correctly running, its telemetry package is periodically received on the ground system every 4 seconds (default

DTW apps telemetry periodicity). For the example app, it was correctly received.

The last verification step consists of integration tests. As per design rules DR-230, DR-270 and DR-290, every application generated with DTW comes natively with 3 telecommands, being “Noop Command”, “Reset Counters” and “Example Ground Command”. These commands are detailed in Table 6.3.

This minimum set of commands offers a starting point for developers in their FSW project. Besides, these commands responses were used as a test means to verify generated applications behavior in integration tests. The commands correct execution is assessed through application telemetry packets or cFE EVS event messages, as explained in Table 6.3.

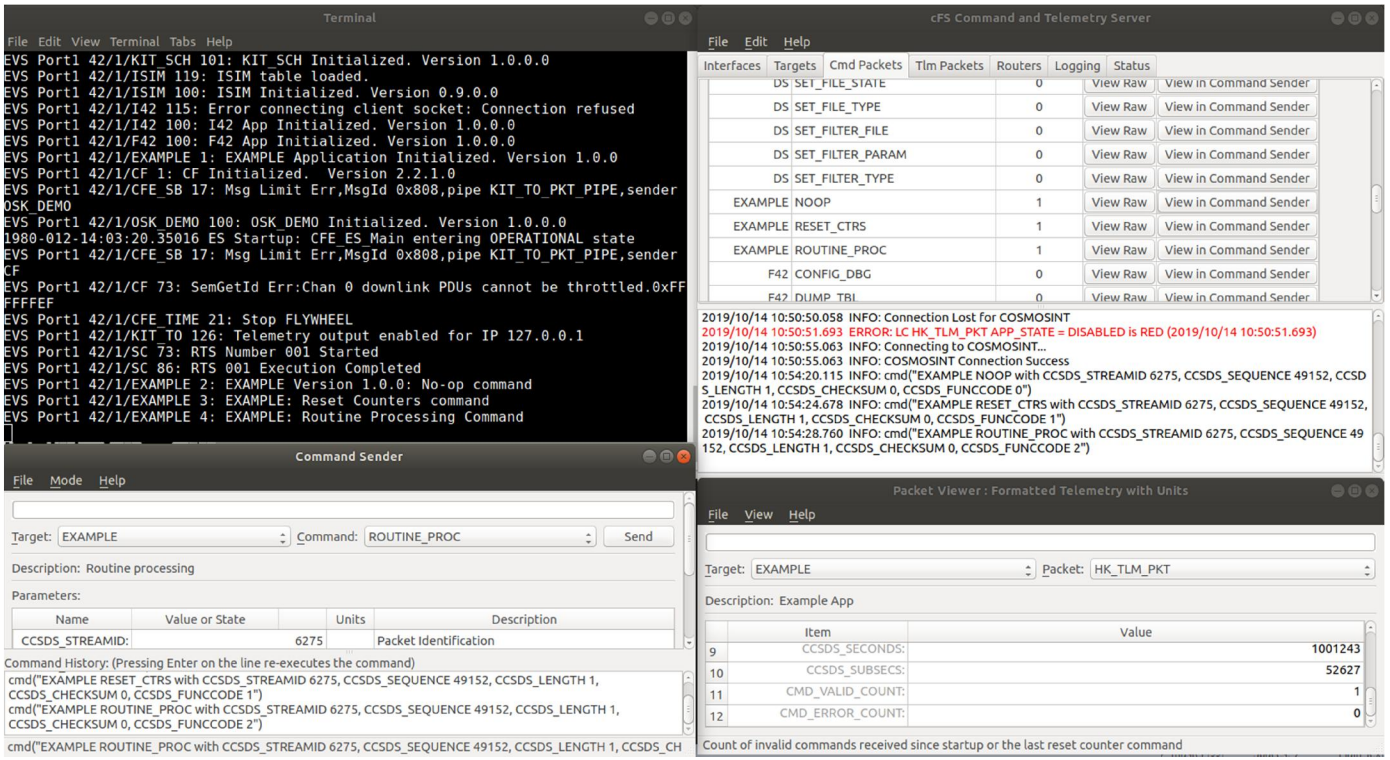
Table 6.3 – Telecommands present in DTW generated cFS applications.

Command	Description	Evidence of Execution
Noop	This command is an “are-you-alive” ping, used to test if application is functioning (DR-230)	- Command Success Counter is increased (DR-250), which can be checked on app telemetry packet - cFE EVS event message is issued (DR-240), which can be checked on cFE EVS app telemetry or on FSW terminal
Reset Counters	This command sets to 0 the internal counters “command success counter” and “command error counter” (DR-270)	- Command Success Counter and command error counter are equal to 0, which can be checked on app telemetry packet - cFE EVS event message is issued (DR-340), which can be checked on cFE EVS app telemetry or on FSW terminal
Ground Command Example	This command is an example of ground command to be tailored by FSW developers (DR-290)	- Command Success Counter is increased (DR-250), which can be checked on app telemetry packet - cFE EVS event message is issued (DR-340), which can be checked on cFE EVS app telemetry or on FSW terminal

Source: Made by the author.

Figure 6.12 shows a printscreen of the environment used for runtime and integration tests (steps 3 and 4) of the generated cFS applications. It is possible to see on the figure’s terminal that the 3 telecommands sent to example app successfully raised 3 corresponding cFE EVS event messages (three last terminal lines).

Figure 6.12 – DTW testing environment.



This picture presents a printscreen of OpenSatKit SDK showing on the top left the terminal that runs the FSW, on the top right the ground control software server, on the bottom left a ground software telecommand sender GUI and on the bottom right the telemetry packet viewer.

Source: Made by the author.

6.4.6. DTW adoption in the reference mission

In the reference mission, DTW was used to prototype new applications to serve several purposes, such as middleware, to emulate faults, to emulate equipment response, and serve to other system-level testing purposes.

Table 6.4 presents three relevant reference mission applications created using DTW, their purpose and the corresponding DTW template ID.

Table 6.4 – Three relevant DTW generated applications used in the reference mission.

cFS application	Purpose	DTW Template ID
Example app	First DTW generated cFS application. It has no flight purpose but helped to create DTW test plan and applications generation philosophy	9
Equipment emulator app	Simulate a reference mission equipment response to commands	7
Mode Manager app	Flight application that manages system level modes in the reference mission	10

Source: Made by the author.

Example app was previosly presented in the last sections. Its purpose was to validate the tool usage, to help create applications generation philosophy and for new team members training.

Equipment emulator app was an application created several times at software development and also system-level testing. This application is soft-periodic (template #7), allowing to emulate equipment periodic

response to the On-Board Computer (OBC), and to insert failures in the equipment response.

As a return of experience, equipment emulator apps were very suitable in the initial phases of FSW development, when real hardware was not available. However, they were not representative enough to evaluate the equipment low-level hardware library correctness, only high-level commands response.

cFS applications were realized to be too high level to allow drivers comprehensive testing. Nevertheless, using them was interesting as a schedule shortcut, especially because they could be, thanks to DTW, quickly prototyped.

Mode manager app, in its turn, is a flight application. In the reference mission, it is an asynchronous cFS application that is permanently listening to the software bus, waiting for certain pre-defined messages. If it realizes that a certain trigger was issued by other application (or even by ground command), it modifies spacecraft system mode accordingly.

Mode manager application template was generated by DTW and further tailored by reference mission FSW developers. As a critical FSW application, it is being extensively tested and so far, no errors were encountered with respect to missing APIs or integration with cFS lower layers.

As a summary, DTW is being used in the reference satellite mission with good acceptance, especially because it points out to the user the minimum necessary cFE/cFS API calls, helping to mitigate errors and typos, but also because it reduces the amount of necessary verification effort in an application. These consequences were extremely beneficial in terms of development time and product quality.

7 CONCLUSIONS

"New Space" missions are experiencing a growing phase nowadays. In Dec 2018, the CEO of the United States Chamber of Commerce said that after maritime system and aviation, space is now the new economic frontier[13].

On the other side, space engineering technical knowledge is mostly residing in government organizations, academia and big corporations. There is a significant lack of information in small companies that inhibits some of them to develop innovative projects.

This dissertation tackled a space engineering field that presents one of the greatest challenges for newcomers: Flight Software (FSW). The pragmatic way found to do so was via a well-established framework.

7.1. **Main contributions**

The author started this work by surveying and comparing relevant FSW frameworks, adopting selection criteria suitable to "New Space" missions, which culminated on the choice for NASA cFS. This analysis can be seen as the first of the main contributions, and was published on JATM journal [MIRANDA, FERREIRA, KUCINSKIS and MCCOMAS (2019)].

Subsequently a technical overview of cFS was given to serve as technical review and background, in order to elucidate the concepts that would be later applied.

A new mission that is willing to adopt cFS on its FSW project must first know what the framework's capabilities are and how it can ease the engineers' job. This was presented in the framework background review (Chapter 4) as well as in the cFS comparison against relevant space software standards (Chapter 5). This description, along with the contents of APPENDIX A, 0 and APPENDIX C, is another contribution.

[13] https://spacenews.com/space-the-new-economic-frontier/

Two standards deserve to be highlighted here: CCSDS SOIS and ECSS PUS. They normatize on-board software services that are deemed relevant to space missions. It was verified that cFS possesses a strong correlation with them, as cFS inherits many years of NASA GSFC research and past scientific missions' errors and successes.

In the reference mission FSW design activities, the area that was most time-consuming was applications design and testing. In order to optimize this activity and avoid time waste, this work proposed another contribution: a systematic development of cFS applications by means of using well-defined design rules and an accompanying template generator (DTW).

This systematic approach is being successfully used in the reference mission but could also be useful and applicable to other cFS-based missions.

7.2. **Future work**

The following topics are a natural evolution of the research performed in this dissertation and will be investigated in future work. They are organized by subject.

General FSW Review Topics:

- cFS impact on operations philosophy, such as files and tables driven operations, is a more advanced topic that would need special attention
- cFS impact on ground control software, such as CFDP engine, cFS apps compiler, tables interpreter, etc, is a more advanced topic that would also need careful attention

cFS and Space Software Standards:

- cFS compliance with CCSDS SOIS EDS standard was not deeply analyzed because it was implemented only at cFE v6.6.0, and this research used cFE v6.5.0

- The document “NASA cFS as a CCSDS Onboard reference architecture” would be a very interesting reference, in great alignment with this work. However, it is still an on-going work by CCSDS Application Support Services Working Group and could not therefore be analyzed.
- CCSDS Mission Operations Services Concept is a Green Book recommendation (CCSDS 520.0-G-3) which presents a Service Oriented Architecture that brings the power of distributed systems to space missions. Using this recommendation, it is possible to re-distribute functionality between space and ground, between spacecrafts, or between nodes of the ground system. cFS compliance to this standard implied a detailed and perhaps interesting study but would not contribute much to this work and was therefore postponed.
- When comparing cFS services to CCSDS SOIS and ECSS PUS, it was noticed that a few proposed services are not implemented in the framework. This could be a potential topic for further development. In especial, cFS tickets could be opened to address the following topics: CCSDS SOIS DDS and DES services implementation; PUS Services 4, 18 and 22 implementation; cFS CF classes 3 and 4 and filestore management capability in destination peer implementation; cFS SCH, LC and DS apps enhancement by fully implementing PUS Services 11, 12 and 15 respectively.

Dan Templates Wrapper (DTW):

- An interesting increment for the applications generator is to automatically create a few unit tests for commands and functions that are recurrently present.
- A 4th-type of cFS application type could exist with respect to periodicity in DTW: time-critical periodic. These applications are even more critical with respect to their execution deadlines than a time-out periodic application and shall rely on an OSAL timer

semaphore pace, for example. This idea will be implemented in version 2 of DTW.

7.3. **Final thoughts**

The FSW development approach proposed here, starting from framework choice and justification, followed by comparison to standards, and finally utilization facilitation due to systematic development of software applications, based on design rules and a tool to help implementing them, is a viable path for cFS-based FSW projects.

If the reference mission had this walk-through review available since its beginning, considerable time would have been saved.

The lack of specialized FSW literature for “New Space” missions was a constant difficulty during the whole research. The information organized herein condenses three years of research and use of cFS framework and applicable FSW literature and standards.

However, all this information was originally conceived for traditional space projects, so there has been a great effort to digest, simplify, adapt and keep what is essential to a “New Space” mission, given the applicable constraints.

cFS usage in CubeSat missions is a brand-new topic, the first one of them being launched in November 2017. Moreover, as far as the author has found, all cFS-based CubeSats launched so far are NASA missions, not being then fully applicable to “New Space” categorization. These facts help to understand the shortage of available academic literature, especially dissertations and thesis.

The author truly hopes that the work here, besides the academic contributions, may be found useful by other “New Space” missions, saving time and condensing minimum information for quick start of FSW projects and space missions themselves.

REFERENCES

AKRE, D. Core flight system onunique missions & experiments: lockheed Martin presentation. In: FLIGHT SOFTWARE WORKSHOP, 10., 2017, Laurel, MD, USA. **Proceedings…** 2017. Available from: http://flightsoftware.jhuapl.edu/files/2017/cFS-Day/cFS17-B4-LMCO_cFS_2017.pptx. Access in: Jan 8, 2018.

APHELION. **Perihelion OS website**. 2018. Available from: http://www.aphelionorbitals.com/perihelionos. Access in: Feb 9, 2019.

ARAGUZ, C.; MARÍ, M.; BOU-BALUST, E.; ALARCON, E.; SELVA, D. Design guidelines for general-purpose payload-oriented nanosatellite software architectures. **Journal of Aerospace Information Systems**, v. 15, n. 3, p. 107-119, 2018. Available from: https://doi.org/10.2514/1.I010537. Access in: Feb 10, 2019.

AUTOSAR. **History**. 2018. Available from: https://www.autosar.org/about/history/. Access in: May 1, 2018.

BARTHOLOMEW, M. O.; KOBE, D. L. **Core Flight Executive (cFE): f**light software application developers guide. July 2014, version 5.4. 89 p. (NASA 582-2007-001). Available from: https://github.com/nasa/cFE/blob/master/cfe/docs/cFE%20Application%20Developers%20Guide.doc. Access in: July 15, 2018.

BIRRANE, E. J.; MAXIMOFF, J.; KRUPIARZ, C.; BAUER, B. A roadmap for responsive software systems. In: AIAA RESPONSIVE SPACE CONFERENCE, 7., 2009, Los Angeles, CA. **Proceedings…** AIAA, 2009. Available from: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.533.2902&rep=rep1&type=pdf. Access in: May 25, 2019.

CONSULTATIVE COMMITTEE FOR SPACE DATA SYSTEMS - CCSDS. **CCSDS TBD.0-O-0 (draft)**: CAST flight software as a CCSDS onboard reference architecture: draft Organge book. Washington, Oct 2016. 54 p. Available from: https://cwe.ccsds.org/sea/docs/SEA-SA/Meeting%20Materials/2016/Fall%202016%20Rome/CAST%20ORANGE%20BOOK.pdf?Mobile=1&Source=%2Fsea%2F_layouts%2Fmobile%2Fview.aspx%3FList%3Da4763d0e-d9be-4b24-b0a7-1e10edb1303c%26View%3D50b434a7-bb62-4e03-a971-45271e7c0b86%26RootFolder%3D%252Fsea%252Fdocs%252FSEA-SA%252FMeeting%2520Materials%252F2016%252FFall%25202016%2520Rome%26ViewMode%3DDetail%26CurrentPage%3D1. Access in: May 1, 2018.

CONSULTATIVE COMMITTEE FOR SPACE DATA SYSTEMS - CCSDS. **CCSDS 131.0-B-3**: TM synchronization and channel coding. Washington, Sept 2017. 97 p. Available from: https://public.ccsds.org/Pubs/131x0b3e1.pdf. Access in: May 28, 2019.

CONSULTATIVE COMMITTEE FOR SPACE DATA SYSTEMS - CCSDS. **CCSDS 132.0-B-2**: TM space data link protocol. Washington, Sept 2015a. 102 p. Available from: https://public.ccsds.org/Pubs/132x0b2.pdf. Access in: Feb 10, 2019.

CONSULTATIVE COMMITTEE FOR SPACE DATA SYSTEMS - CCSDS. **CCSDS 133.0-B-1**: space packet protocol. Washington, Sept 2003. 49 p. Available from: https://public.ccsds.org/Pubs/133x0b1c2.pdf. Access in: Jan 29, 2018.

CONSULTATIVE COMMITTEE FOR SPACE DATA SYSTEMS - CCSDS. **CCSDS 232.0-B-3**: TC space data link protocol. Washington, Sept 2015b. 123 p. Available from: https://public.ccsds.org/Pubs/232x0b3.pdf. Access in: Feb 10, 2019.

CONSULTATIVE COMMITTEE FOR SPACE DATA SYSTEMS - CCSDS. **CCSDS 301.0-B-4**: time code formats. Washington, Nov 2010. 43 p. Available from: https://public.ccsds.org/Pubs/301x0b4e1.pdf. Access in: Jan 29, 2018.

CONSULTATIVE COMMITTEE FOR SPACE DATA SYSTEMS - CCSDS. **CCSDS 727.0-B-4**: CCSDS file delivery protocol. Washington, Jan 2007. 146 p. Available from: https://public.ccsds.org/Pubs/727x0b4.pdf. Access in: Jan 29, 2019.

CONSULTATIVE COMMITTEE FOR SPACE DATA SYSTEMS - CCSDS. **CCSDS 850.0-G-2**: spacecraft onboard interface services. Washington, Dec 2013. 88 p. Available from: https://public.ccsds.org/Pubs/850x0g2.pdf. Access in: Jun 06, 2019.

CHIN, A.; COELHO, R.; BROOKS, L.; NUGENT, R.; PUIG-SUARI, J. Standardization promotes flexibility: a review of CubeSats' success. In: RESPONSIVE SPACE CONFERENCE, 6., 2008, Los Angeles, CA. **Proceedings…** AIAA, 2008. Available from: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.559.7055&rep=rep1&type=pdf. Access in: Jan 5, 2019.

CHOI, W. S.; KIM, J. H.; KIM, H. D. A study on developing flight software for nano-satellite based on NASA CFS. **Journal of the Korean Society for Aeronautical & Space Sciences**, v. 44, n. 11, p. 997-1005, 2016. Available from: http://www.dbpia.co.kr/Journal/ArticleDetail/NODE07050684. Access in: Apr 10, 2018.

CLERIGO, I.; ACCOMAZZO, A.; COLLINS, P.; MARDLE, N.; MONTAGNON, E.; MORALES, J.; TANCO, I. One standard to rule them all: the tailoring of PUS-C for future ESA missions. In: SPACEOPS CONFERENCE, 2018, Marseille, France. **Proceedings…** AIAA, 2018. Available from: https://arc.aiaa.org/doi/pdf/10.2514/6.2018-2399?download=true. Access in: Sept 9, 2019.

COELHO, C.; KOUDELKA, O.; MERRI, M. NanoSat MO framework: achieving on-board software portability. In: SPACEOPS CONFERENCE, 2016, Daejeon, Korea. **Proceedings…** AIAA, 2016. Available from: https://doi.org/10.2514/6.2016-2624. Access in: Sept 23, 2018.

CORE FLIGHT SYSTEM - CFS. **About the technology and why cFS**. 2017. Available from: http://coreflightsystem.org/about-the-technology-why-cfs/ Access in: May 12, 2018.

CUDMORE, A. **core Flight Software (cFS) OS abstraction layer library**. NASA GSFC Flight Software Branch (Code 582): Greenbelt, MD, USA: Jan 2016. 133 p. Available from: https://sourceforge.net/projects/osal/files/osal-4.2.1a-release.tar.gz/download. Access in: Jun 28, 2017.

CUDMORE, A. Porting the core Flight System to the Dellingr Cubesat. In: FLIGHT SOFTWARE WORKSHOP, 10., 2017, Laurel, MD. **Proceedings…** 2017. Available from: https://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/20170011566.pdf. Access in: Oct 28, 2018.

DE CONTO, A.; MATTEI, A. P.; SAQUIS-SANNES, P.; CARVALHO, H.; MIRANDA, D.; BALBINO, F. Use of SysML and Model-Based System Engineering in the development of the Brazilian Satellite VCUB1. In: EUROPEAN CUBESAT SYMPOSIUM, 10., 2018, Toulouse, France. **Proceedings…** 2018. Available from: http://oatao.univ-toulouse.fr/20740/. Access in: Jan 6, 2019.

DOS SANTOS, W. A. **Adaptability, reusability and variability in software systems for space on-board computing**. 2008. 233 p. Thesis (PhD) - Aeronautics Institute of Technology (ITA), São José dos Campos, 2008. Available from: http://www.bdita.bibl.ita.br/tesesdigitais/lista_resumo.php?num_tese=000551325. Access in: Jan 31, 2019.

DVORAK, D. L. NASA study on flight software complexity. In: AIAA INFOTECH@AEROSPACE CONFERENCE, 2009, Seattle. **Proceedings…** Seatlle: Aerospace Conference, 2009. Available from: https://arc.aiaa.org/doi/10.2514/6.2009-1882. Access in: May 20, 2018.

EUROPEAN COOPERATION FOR SPACE STANDARDIZATION - ECSS. **ECSS-E-ST-70-41C:** telemetry and telecommand Packet Utilization (PUS). The Netherlands: ESA Publications Division, Apr 2016. 656 p. Available from: https://ecss.nl/standard/ecss-e-st-70-41c-space-engineering-telemetry-and-telecommand-packet-utilization-15-april-2016/. Access in: Feb 2, 2018.

FEMMER, H. **Equivalence analysis for software abstraction layers**. 2012. 141 p. Thesis (Master's) - Augsburg and Munich University, Munich, 2012. Available from: https://wwwbroy.in.tum.de/~femmer/works/femm12.pdf. Access in: Feb 1, 2018.

GANESAN, D.; LINDVALL M.; ACKERMANN, C.; MCCOMAS, D.; BARTHOLOMEW, M. Verifying architectural design rules of the flight software product line. In: INTERNATIONAL SOFTWARE PRODUCT LINE CONFERENCE, 13., 2009, San Francisco, USA.. **Proceedings…** 2009. P.1061-170. Available from: https://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/20090016208.pdfhttps://www.researchgate.net/profile/Dharmalingam_Ganesan/publication/220789647_Verifying_architectural_design_rules_of_the_flight_software_product_line/links/00b7d5280c858f2b56000000/Verifying-architectural-design-rules-of-the-flight-software-product-line.pdf?origin=publication_detail. Access in: Dec 19, 2017.

GANESAN, D.; LINDVALL M.; MCCOMAS, D.; BARTHOLOMEW, M.; SLEGEL, S.; MEDINA, B.; KRIKHAAR, R.; VERHOEF, C.; MONTGOMERY, L. P. An analysis of unit tests of a flight software product line. **Science of Computer Programming**, v. 78, n.12, p. 2360-2380, 2013. Available from: https://www.sciencedirect.com/science/article/pii/S0167642312000317. Access in: Aug 23, 2018.

HARTMANN, H. **Software product line engineering for consumer electronics:** keeping up with the speed of innovation. 2015. 281 p. Thesis (PhD) - University of Groningen, Groningen, 2015. Available from: http://www.rug.nl/research/portal/en/publications/software-product-line-engineering-for-consumer-electronics(6aa1ffab-abf6-4d3e-97cb-1e23daed93c5).html. Access in: Jun 03, 2017.

HARTSELL, C.; KARSAI, G.; LOWRY, M. Timing analysis of a middleware-based system. In: WORKSHOP ON ADAPTIVE AND REFLECTIVE MIDDLEWARE. Las Vegas-USA: Dec, 2017. **Proceedings**… 2017. Available from: https://dl.acm.org/citation.cfm?id=3152886. Access in Apr 15, 2018.

KANEKAL, S. et al. CeREs: the compact radiation belt explorer. In: ANNUAL AIAA/USU CONFERENCE ON SMALL SATELLITES, 32., 2018, Logan, UT. **Proceedings…** 2018. Available from: https://digitalcommons.usu.edu/cgi/viewcontent.cgi?article=4071&context=smallsat. Access in: Feb 9, 2019.

KEPTO, L.; SANTOS, L.; CLAGETT, C.; AZIMI, B.; CHAI, D.; CUDMORE, A.; STARIN, S.; MARSHALL, J.; LUCAS, J. Dellingr: reliability lessons learned from on-orbit. In: Annual AIAA/USU CONFERENCE ON SMALL SATELLITES, 32., 2018, Logan, UT, USA. **Proceedings…** 2018. Available from: https://digitalcommons.usu.edu/smallsat/2018/all2018/250/. Access in: Feb 2, 2019.

KUBOS. **KubOS webpage**. 2018. Available from: https://docs.kubos.com/1.5.0/index.html. Access in: Jul 10, 2018.

KUCINSKIS, F. N. **Uma arquitetura de software embarcado do segmento espacial para habilitar a operação de missões baseada em objetivos**. 2012. 194 p. Tese (Doutorado em Engenharia e Tecnologia Espaciais) - Instituto Nacional de Pesquisas Espaciais, São José dos Campos, 2012. Available from: http://urlib.net/8JMKD3MGP7W/3BEQBSP. Access in: May 01, 2017.

KUCINSKIS, N, F.; FERREIRA, VIERA, G, M. On-board satellite software architecture for the goal-based Brazilian mission operations. **IEEE Aerospace and Electronic Systems Magazine**, v. 28, p. 32-45, 2013. Available from: https://ieeexplore.ieee.org/document/6575409. Access in: Jan 20, 2019.

LEPEYRERE, V.; LACOUR, S.; DAVID, L.; NOWAK, M.; CROUZIER, A.; SCHWORER, G.; PERROT, P.; RAYANE, S. PicSat: a Cubesat mission for exoplanetary transit detection in 2017. In: ANNUAL AIAA/USU CONFERENCE ON SMALL SATELLITES, 31., 2017, Logan, UT. **Proceedings…** 2017. Available from: https://digitalcommons.usu.edu/cgi/viewcontent.cgi?article=3617&context=smallsat. Access in: Oct 29, 2018.

LINDVALL M.; BECKER M.; TENEV V.; DUSZYNSKI S.; HINCHEY M. Good change and bad change: an analysis perspective on software evolution. In: STEFFEN, B. (Ed.). **Transactions on foundations for mastering change I**. Berlin: Springer, 2016. Available from: https://link.springer.com/chapter/10.1007%2F978-3-319-46508-1_6. Access in: Jan 07, 2019.

MCCOMAS, D.; STREGE, S.; WILMOT, J. **core Flight System (cFS):** a low cost solution for smallsats. Aug 2015. Available from: https://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/20150018075.pdf. Access in: Nov 15, 2017.

MCCOMAS, D. Increasing flight software reuse with OpenSatKit. In: IEEE AEROSPACE CONFERENCE, 2018. **Proceedings…** IEEE, 2018. Available from: https://ieeexplore.ieee.org/document/8396631. Access in: Oct 1, 2018.

MCCOMAS, D. **Lessons from 30 years of flight software**. Sept. 2015. Available from: https://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/20150019915.pdf. Access in: Dec 1, 2017.

MCCOMAS, D.; WILMOT, J.; CUDMORE, A. The Core Flight System (cFS) community: providing low cost solutions for small spacecraft. In: ANNUAL AIAA/USU CONFERENCE ON SMALL SATELLITES, 30., 2016, Logan, UT. **Proceedings…** 2016. Available from: https://ntrs.nasa.gov/search.jsp?R=20160010300. Access in: Oct 10, 2017.

MIRANDA, D.; FERREIRA, M.; KUCINSKIS, F.; MCCOMAS, D. A comparative survey on flight software frameworks for 'new space' nanosatellite nissions. **Journal of Aerospace Technology and Management**, v. 11, e4619, 2019. Available from: http://www.jatm.com.br/ojs/index.php/jatm/article/view/1081/785. Access in: Nov 2, 2019.

MORRIS, J. et al. Simulation-to-Flight 1 (STF-1): a mission to enable CubeSat software-based verification and validation. In: AIAA AEROSPACE SCIENCES MEETING, 54., 2016, San Diego, CA. **Proceedings…** AIAA, 2016. Available from: https://arc.aiaa.org/doi/abs/10.2514/6.2016-1464. Access in: Feb 10, 2019.

NATIONAL AERONAUTICS AND SPACE ADMINISTRATION - NASA. **NASA software catalog 2017-2018**. 2017. Available from: https://software.nasa.gov/NASA_Software_Catalog_2017-18.pdf. Access in: Nov 25, 2017.

NATIONAL AERONAUTICS AND SPACE ADMINISTRATION - NASA. **core Flight System (cFS) background and overview**. 2014. Available from: https://cfs.gsfc.nasa.gov/cFS-OviewBGSlideDeck-ExportControl-Final.pdf. Access in: Feb 5, 2017.

NORMANN, M. A. **Software design of an onboard computer for a nanosatellite**. Dissertation (Master in Cybernetics and Robotics) - Norwegian University of Science and Technology (NTNU), Trondheim, 2016. Available from: https://brage.bibsys.no/xmlui/bitstream/handle/11250/2413318/14533_FULLTEXT.pdf?sequence=1. Access in: Aug 21, 2018.

O ESTADO DE SÃO PAULO. **Indústria revela projetos inovadores**. Available from: https://economia.estadao.com.br/noticias/geral,industria-revela-projetos-inovadores,70002874897. Access in: Jul 30, 2019.

PAIKOWSKY, D. What is new space? the changing ecosystem of global space activity. **New Space**, v. 5, n. 2, p. 84-88, 2017. Available from: https://doi.org/10.1089/space.2016.0027. Access in: Jan 12, 2019.

PASETTI, A. **The CORDET framework PUS extension**. Revision 0.2 (In Progress). Tägerwilen: P&P Software GmbH, Apr 17, 2018. Available from: https://github.com/pnp-software/cordetfw/blob/pus_ext/doc/pus/PusExtension.pdf. Access in: May 1, 2018.

PLASSON, P.; CUOMO, C.; GABRIEL, G.; GAUTHIER, N.; GUEGUEN, L.; MALAC-ALLAIN, L. GERICOS: a generic framework for the development of on-board software. In: 'DASIA 2016' DATA SYSTEMS IN AEROSPACE, 2016, Tallin, Estonia. **Proceedings…** 2016. Available from: http://adsabs.harvard.edu/abs/2016ESASP.736E..39P. Access in: Jan 17, 2018.

P&P SOFTWARE. **The CORDET project**. Sept, 2008. Available from: https://www.pnp-software.com/cordet/home.html. Access in: May 1, 2018.

P&P SOFTWARE. **CORDET C2 implementation download**. 2018. Available from: https://www.pnp-software.com/cordetfw/download.html. Access in: May 1, 2018.

PROKOP, L. **CFS training slides:** presentation given at Johnson Space Center (JSC). Nov. 2014. Available from: https://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/20140017040.pdf. Access in: July 03, 2017.

REXROAT, J. T. **Proposed middleware solution for resource-constrained distributed embedded networks**. 2014. 160 p. Dissertation (Master in Electrical Engineering) – University of Kentucky, Lexington, 2014. Available from: https://uknowledge.uky.edu/ece_etds/63/. Access in: Sept 14, 2018.

SAVOIR. **SAVOIR outputs**. 2018. Available from: http://savoir.estec.esa.int/SAVOIROutput.htm. Access in: May 1, 2018.

SCHMIDT, D. C.; GOKHALE, A.; NATARAJAN, B. Leveraging application frameworks: why frameworks are important and how to apply them effectively. **ACM Queue**, V. 2, n. 5, Aug. 31, 2004. Available from: https://queue.acm.org/detail.cfm?id=1017005. Access in: Oct 22, 2018.

SCHULZE, C.; GANESAN, D.; LINDVALL, M.; MCCOMAS, D.; CUDMORE, A. Model-based testing of NASA's OSAL API — An experience report. In: INTERNATIONAL SYMPOSIUM ON SOFTWARE RELIABILITY ENGINEERING (ISSRE), 24., 2013, Pasadena, CA, USA. **Proceedings…** 2013. p 300-309. Available from: https://ieeexplore.ieee.org/document/6698883/. Access in: Jan 10, 2018.

STREGE, S. **Core flight software version description document**. cFE v6.6.0. NASA, Dec. 2017. Available from: https://github.com/nasa/cFE/blob/rc-6.6.0a/docs/cFE_6_6_0_version_description.pdf. Access in: June 20, 2019.

SUKHWANI, H.; ALONSO, J.; TRIVEDI, K. S.; MCGINNIS, I. Software reliability analysis of NASA Space Flight software: a practical experience. In: INTERNATIONAL CONFERENCE ON SOFTWARE QUALITY, RELIABILITY AND SECURITY, 2016. **Proceedings…** IEEE, 2016. p 386–397. Available from: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5737753/pdf/nihms922844.pdf. Access in: June, 10, 2018.

SWARTWOUT, M. The first one hundred CubeSats: a statistical look. **Journal of Small Satellites (JoSS)**, v. 2, n. 2, p. 213-233. 2013. Available from: http://web.csulb.edu/~hill/ee400d/Project%20Folder/CubeSat/The%20First%20One%20Hundred%20Cubesats.pdf. Access in: Jan 6, 2019.

TAKADA, M.; TAKADA, H.; NARITA, T.; KUSANO, M.; FUKUDA, S.; MATSUZAKI, K.; ISHIDA, T.; ISHIHAMA, N.; NOMACHI, M. Porting cFE to spacecraft onboard with SpaceWire Engine and RTOS based on uITRON specification. In: FLIGHT SOFTWARE WORKSHOP, 10., 2017, Laurel, MD. **Proceedings…** 2017. Available from: http://flightsoftware.jhuapl.edu/files/2017/Day-3/03-Takada-PortingcFE2uITRON.pdf. Access in: Jan 10, 2018.

TERRAILLON, J. L. SAVOIR: Reusing specifications to improve the way we deliver avionics. In: EMBEDDED REAL TIME SOFTWARE AND SYSTEMS, 6., 2012, Toulouse-France. **Proceedings…** Toulouse, France: ERTS², 2012. Available from: http://web1.see.asso.fr/erts2012/Site/0P2RUC89/6C-1.pdf. Access in: Apr 28, 2018.

WILMOT, J. A core flight software system. In: CODES+ISSS INTERNATIONAL CONFERENCE ON HARDWARE/SOFTWARE CODESIGN AND SYSTEM SYNTHESIS, 3., 2005. **Proceedings…** Jersey City, 2005. p. 13-14. Available from: https://dl.acm.org/citation.cfm?id=1084842. Access in: May 20, 2018.

WILMOT, J.; FESQ, L.; DVORAK, D. Quality attributes for mission flight software: a reference for architects. In: IEEE AEROSPACE CONFERENCE, Mar 2016, Big Sky, MT, USA. **Proceedings… IEEE, 2016.** Available from: https://ieeexplore.ieee.org/document/7500850/. Access in: Sept 8, 2018.

WILMOT, J. Use of CFDP in NASA/GFSC's flight SW architecture. In: ESA ADCSS, 6., 2012. **Proceedings…** ESA/ESTEC, 2012. Available from: https://indico.esa.int/event/67/contributions/3057/attachments/2442/2815/1655_-_Use_of_CFDP_in_NASA-GSFCs_flight_SW_architecture.pdf. Access in: June 20, 2019.

WILMOT, J. Using CCSDS standards to reduce mission costs. In: ANNUAL AIAA/USU CONFERENCE ON SMALL SATELLITES, 31., 2017, Logan, UT, USA. **Proceedings…** 2017. Available from: https://digitalcommons.usu.edu/cgi/viewcontent.cgi?referer=https://www.google.com/&httpsredir=1&article=3686&context=smallsat. Access in: June 7, 2019.

XIONGWEN, H. et al. Design and implementation of spacecraft avionics software architecture based on spacecraft onboard interface services and packet utilization standard. In: INTERNATIONAL ASTRONAUTICAL CONGRESS, 66., 2015, Jerusalem. **Proceedings…** 2015. Available from: http://iafastro.directory/iac/archive/browse/IAC-15/D1/4/28023/. Access in: May 2, 2018.

YANCHIK, N. J. Operating system abstraction layer. In: FLIGHT SOFTWARE WORKSHOP, 1., 2007, Laurel, MD, USA. **Proceedings…** 2007. Available from: https://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/20080040870.pdf. Access in: Aug, 2017.

APPENDIX A – cFS ARCHITECTURE ADVANCED TOPICS

This appendix brings cFS architecture topics that are complimentary to Chapter 4. These advanced topics can be useful to better understand Chapters 5 and 6 technical details.

Moreover, this Chapter brings deeper concepts about cFS utilization and structure that might be valuable for missions interested in applying the same software development approach described in this research.

A.1. OSAL

One question that a software expert might ask is why one don't just create a generic POSIX compliant OS interface layer instead of starting a new generic operating system abstraction layer from scratch. This question is fair since portability among several operating systems is what motivated POSIX development. Besides, RTEMS and VxWorks are fully POSIX-compliant, and Mac OS X, Cygwin and Linux systems are mostly POSIX-compliant. NASA experts had this same insight more than 2 decades ago.

Femmer (2012) corresponded with Alan Cudmore, which said that in the beginning the OSAL research project was called "POSIX flight software" project. Nevertheless, POSIX turned out to not be a sufficient OS abstraction layer for space mission purposes and then OSAL was created as result. The reasons for that can be found at Femmer (2012). That author also addressed the problem of software abstraction layer equivalence among the several OSAL ports. OSAL v3.2 POSIX (Unix), VxWorks and RTEMS ports were analyzed and some equivalence issues were found, most of them of low severity and the others were acknowledged as bugs by the OSAL team and fixed in the most recent release.

OSAL v4.2.1 contains approximately 100 APIs, which are divided into three major sections: Real Time Operating System APIs, File System APIs, and Interrupt/Exception APIs.

The Real Time Operating System APIs cover typical operating system functions such as Tasks, Queues, Semaphores, Interrupts, etc.

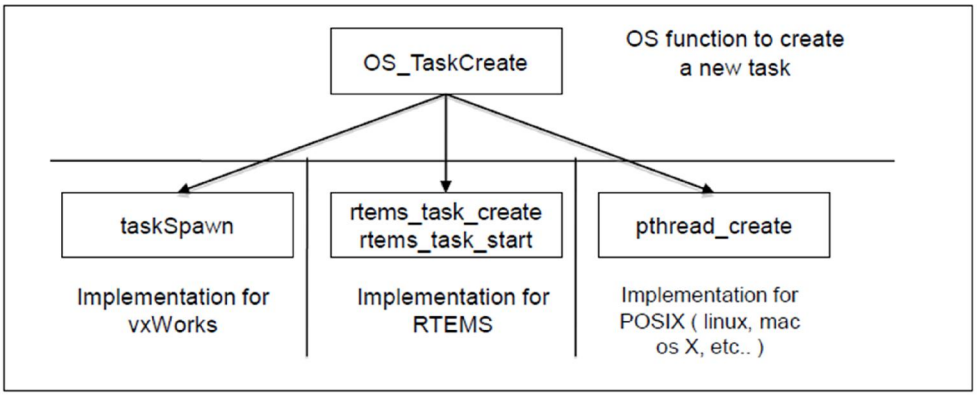
The File System APIs abstract the file systems that may be present on a system and can simulate multiple embedded file systems on a host desktop machine for development and testing. It is worth to mention that very light embedded systems, for example the ones which uses microcontroller systems with limited resources, usually can't hold a filesystem and therefore are not candidates for using OSAL and consequently cFS. Also, some RTOS with very small footprint and which have no integrated filesystem shall be first combined with a filesystem, such as FAT, to then be successfully wrapped by OSAL. This is the case for FreeRTOS which is very popular in the CubeSat world.

The Interrupt/Exception APIs are for configuring interrupt and exception handlers. In older OSAL versions, there were also Hardware APIs, which purpose was configuring for example I/O and Memory Access APIs. They are now part of PSP, that will be detailed in Section A.2.

An example of OSAL capacity is shown in Figure A.1. Different operating systems have different APIs for creating a task. In VxWorks, the function is *taskSpawn*. In RTEMS two functions are necessary, first *rtems_task_create* and then, if successful, *rtems_task_start* to place the task in "ready for scheduling" state. In POSIX, the function is *pthread_create*. In each operating system, the equivalent function has a different number of arguments and returns different outputs and error codes. OSAL merit is to wrap all that so that cFS developers only have to care about a generic function *OS_TaskCreate* independently of the underlying operating system.

If someone wants to work with an RTOS not already ported to OSAL, it will be necessary to develop a specific OSAL port. That's what Takada et al. (2017) did in Japan. They developed an OSAL port to the Japanese TOPPERS operating system so that they could use cFS with this system that they were familiar with.

Figure A.1 – OSAL OS_TaskCreate() API example.



OS_TaskCreate() is an API that wraps RTOS specific calls to create new tasks.

Source: Obtained from NASA (2014).

OSAL has being subject to several independent tests. Schulze et al. (2013) showed that OSAL is a high-quality product by means of model-based extensive testing. The generated test cases achieved about 96% of code coverage. Some few previously unknown “corner-case” bugs and issues were found, but with minor impact in OSAL overall reliability.

A.2. PSP

As mentioned in Section A.1, the Platform Support Package (PSP) layer is a cFS software layer forked from OSAL. Now it is a standalone cFS product. PSP contains about 30 API’s such as:

- Memory read, write, copy and critical memory area management functions
- Processor reset functions
- Watchdog functions
- Timer functions

The functions PSP wraps are usually found in the BSP (Board Support Package). A BSP is developed by a hardware or computer manufacturer who wishes to support a particular RTOS.

Each mission is supposed to get the BSP for its target RTOS from the on-board computer vendor and then customize PSP with BSP content. An example of this task can be seen in Figure A.2.

Figure A.2 – PSP CFE_PSP_WatchdogEnable() API example.

```
/************************************************************************
** Function: CFE_PSP_WatchdogEnable()
**
** Purpose:
**   Enable the watchdog timer
**
** Arguments:
**
** Return:
*/
void CFE_PSP_WatchdogEnable(void)
{
        //Put specific BSP Watchdog calls here

        //For IBM PowerPC750 processor family with BSP for Windriver's
VxWorks v6.4, the specific calls would be:
        /* Arm the WDT2 control register */
        //PCI_OUT_BYTE(0xFEFF0068, 0x55);
         /* The enable/disable bit is bit 15, a setting of 1 enables the timer.*/
        //PCI_OUT_LONG(0xFEFF0068,0xFFFFFFAA);

}
```

CFE_PSP_WatchdogEnable() is a BSP function wrapper. For PowerPC750 processor with VxWorks 6.4 as RTOS, the calls to enable processor watchdog timer are the ones inside the function.

Source: Obtained from PSP v.1.3.0.0 code.

Some PSP ports come by default in PSP open-sourced v.1.3.0.0, such as the port for PowerPC750 processor and VxWorks 6.4 RTOS. This port for instance is a legacy development made by NASA GSFC for its Global Precipitation Measurement (GPM) mission. Some other legacy ports can be found in the PSP open-source version.

As for the OSAL, if someone wants to port PSP for a specific set of processor card plus operating system not already done, a customization effort is necessary. Takada et al. (2017) for example ported PSP to their space-grade system-on-a-chip processor SOISOC3 and TOPPERS RTOS.

A.3. cFE executive services (ES)

The cFE Executive Services (ES) provides the runtime environment that allows applications to be managed as an architectural component (MCCOMAS et al., 2016). It is responsible for:

- cFE startup (power-on or processor reset);
- Start, restart and delete cFS applications;
- Manage the Critical Data Storage (CDS) which can be used to preserve data in case of processor reset;
- Load shared libraries;
- Device Drivers support;
- Log information related to resets and exceptions;
- Manage a system log for capturing information and errors;
- Performance Analysis support

It is composed of approximately 40 API's as of cFE v6.5.0. An example of cFE ES API is shown in Figure A.3. The cFE ES API's can be used by cFS application developers in order to call specific cFE ES services. For example, cFE ES CFE_ES_GetResetType() function returns the type of the last cFE reset, which can be helpful to see if the on-board computer was last reset in power-on mode, meaning that volatile memory was erased, or in processor-reset mode, meaning that a specific part of volatile memory, known as Critical Data Storage (CDS), was preserved if configured. This kind of knowledge is important, for example, for the initialization of variables in cFS applications.

Figure A.3 – cFE ES CFE_ES_GetResetType() API example.

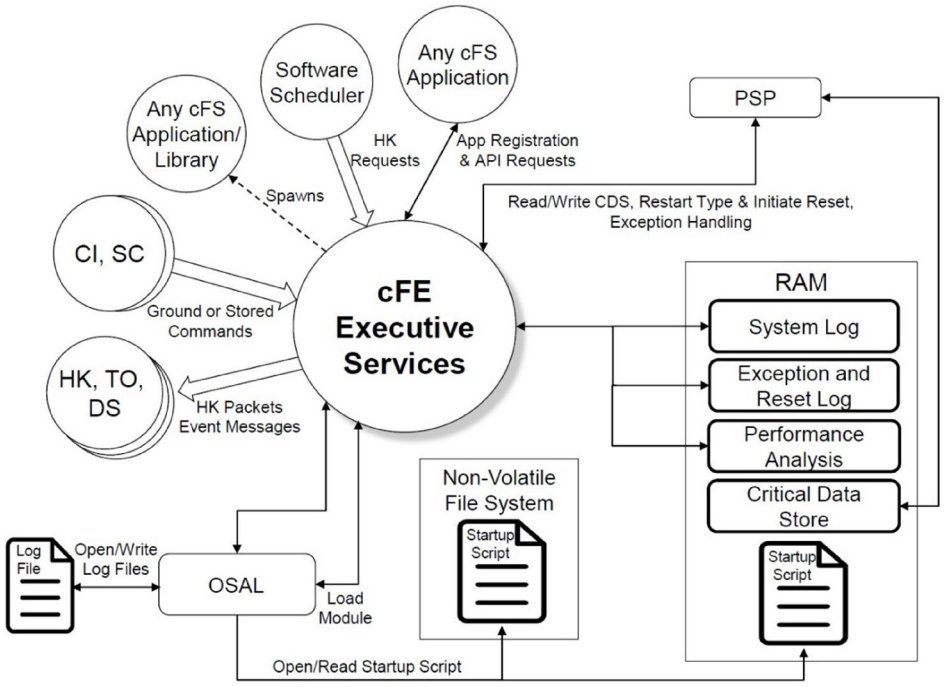
```
/*
** Function: CFE_ES_GetResetType
**
** Purpose:  Return The Type of reset the cFE had.
**       The function will return the start type
** which is CFE_PSP_RST_TYPE_POWERON or
**CFE_PSP_RST_TYPE_PROCESSOR.
**          The sub-type is optional and will be returned if a non-NULL
** pointer is passed in to the function.
*/
int32 CFE_ES_GetResetType(uint32 *ResetSubtypePtr)
{
   if ( ResetSubtypePtr != NULL )
   {
      *ResetSubtypePtr = CFE_ES_ResetDataPtr->ResetVars.ResetSubtype;
   }

   return(CFE_ES_ResetDataPtr->ResetVars.ResetType);

} /* End of CFE_ES_GetResetType() */
```

CFE_ES_GetResetType() is a cFE ES API which returns the last reset type cFE had (CFE_PSP_RST_TYPE_POWERON or CFE_PSP_RST_TYPE_PROCESSOR).

Source: Obtained from cFE v6.5.0 Executive Service code.

The relationship between cFE Executive Services and the remaining cFS products can be seen in its context diagram, presented in Figure A.4.

Figure A.4 – cFE ES Context Diagram.



This context diagram shows the relationships between cFE ES, OSAL, PSP and the cFS applications.

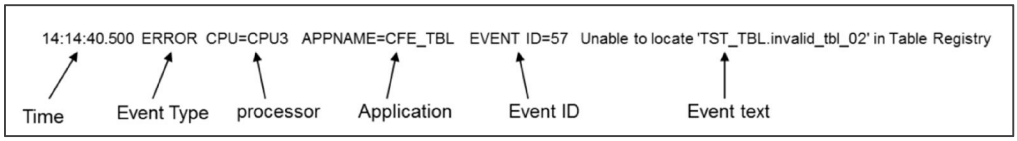
Source: Obtained from NASA (2014).

A.4. cFE event services (EVS)

The cFE Event Services (EVS) allows applications to send time-stamped parameterized text messages. Four message classes based on severity are defined and filtering can be applied on a per-message and per-class basis (MCCOMAS et al., 2016).

The four message classes ordered by crescent severity are: Debug, Information, Error and Critical. Event messages are sent asynchronously, i.e., whenever a defined trigger occurs and are forwarded to the software bus or even to user-defined hardware message ports. An example of event message is shown in Figure A.5.

Figure A.5 – cFE EVS Event Message example.

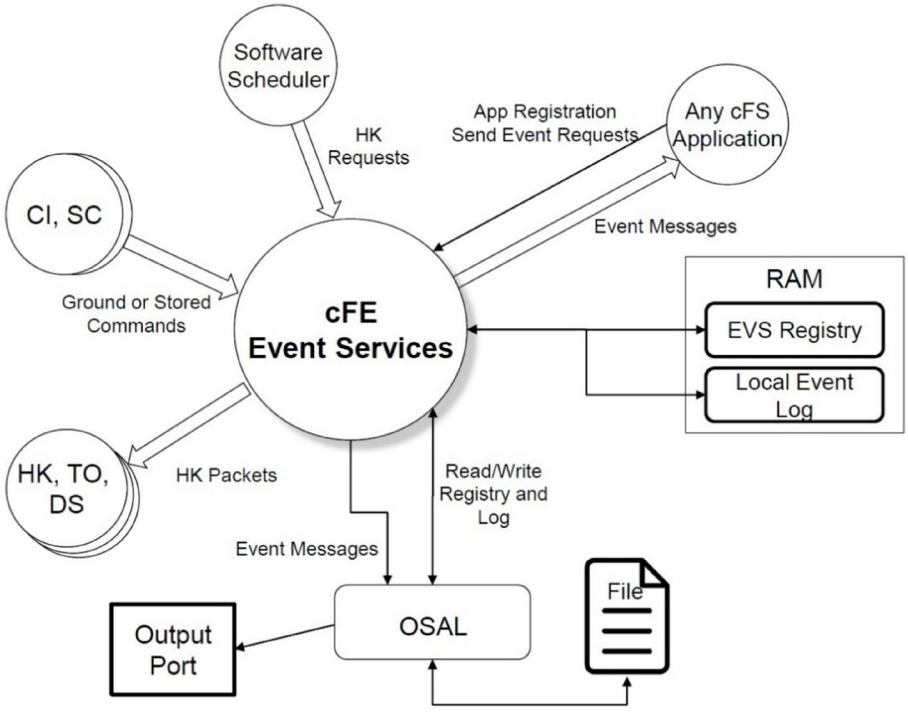


This event message is an example of error type event message generated by cFE_TBL application due to a certain invalid action.

Source: Obtained from NASA (2014).

cFE EVS is composed of 7 API's as of cFE v6.5.0. The relationship between cFE Event Services and the remaining cFS products can be seen in its context diagram, presented in Figure A.6.

Figure A.6 – cFE EVS Context Diagram.



This context diagram shows the relationships between cFE EVS, OSAL and the cFS applications.

Source: Obtained from NASA (2014).

A.5. cFE software bus (SB)

The cFE Software Bus (SB) is the cFE core service that provides a publish-and-subscribe CCSDS standards-based inter application messaging system that supports single and multi-processor configurations (MCCOMAS, WILMOT and CUDMORE, 2016).

cFE SB routes a certain application message to all applications that have subscribed to the message. Subscriptions are done at application startup, but message routing can be added or removed at runtime.

cFE SB contains 23 API's as of cFE v6.5.0. The Software Bus application implements the CCSDS Space Packet Protocol standard (CCSDS, 2003). A packet primary header, containing 6 bytes, is fully-compliant with CCSDS (2003) standard, as shown in 0, Section B.1.

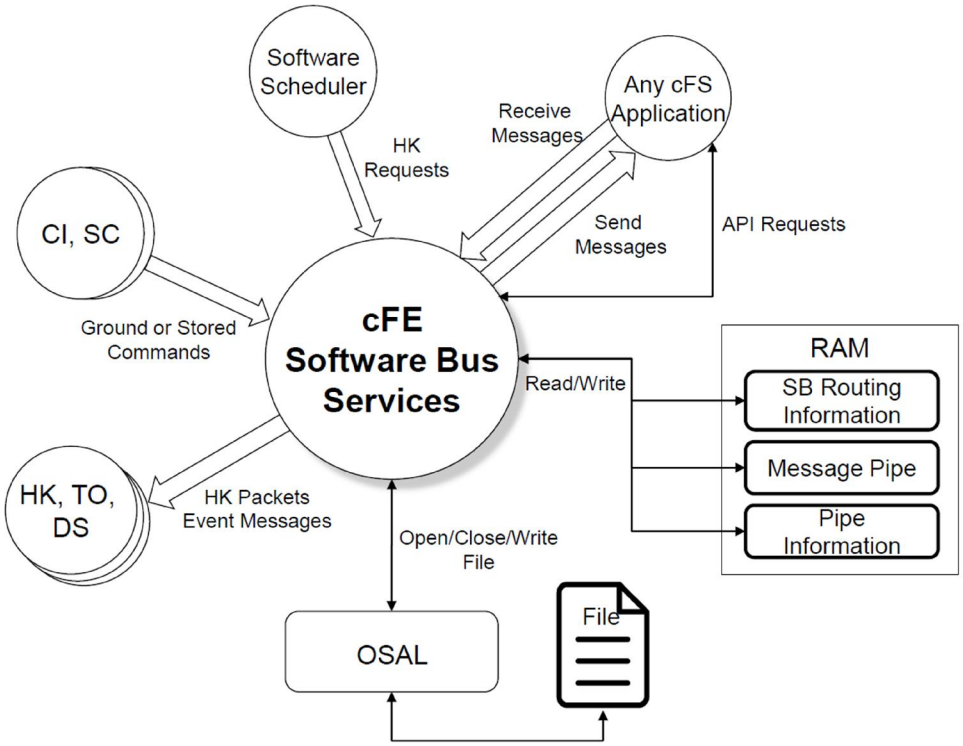
The secondary header, which is variable as per the standard, is implemented in SB according to cFE specificities: 2 bytes for telecommands (containing the command function code and a checksum, calculated by ground system) and 6 or 8 bytes for telemetries (containing time-stamp information).

Time-information is compliant with CCSDS Unsegmented time Code (CUC) (CCSDS, 2010), with 4 bytes for the elapsed time in seconds as the basic time unit and 2 or 4 bytes for subseconds (1 second = 2^32 subseconds) representing the elapsed binary fraction of the basic time unit.

Information exchange between applications is primarily done via Software Bus messages. This feature enhances cFE/cFS modularity. Also, this characteristic allows spacecraft operators to easily perform override commands, because cFE Software Bus publish-subscribe method is to subscribe to a message ID and not to a message sender. As an example, application A subscribing to a certain type of message typically sent from application B could receive this same message sent from the ground segment without application A knowledge.

The relationship between cFE Software Bus and the remaining cFS products can be seen in its context diagram, presented in Figure A.7.

Figure A.7 – cFE SB Context Diagram.



This context diagram shows the relationships between cFE SB, OSAL and the cFS applications.

Source: Obtained from NASA (2014).

A.6. cFE table services (TBL)

Tables are binary files containing groups of application defined parameters that can be changed during runtime. The cFE Table Services (TBL) provides an interface for loading and dumping an application's table (MCCOMAS et al., 2016).

In cFE, each application is not required to manage its own tables. It can make use of the 16 cFE TBL API's as of cFE v6.5.0 in order to manage its tables.

cFS applications can be table driven, which allows for scalable system integration and parameters set patch at runtime. For instance, Guidance, Navigation and Control (GNC) or AOCS (Attitude and Orbit Control System) applications generally make use of a large quantity of tunable parameters (control gains, filter gains, control flags, control bias, etc).

When one wishes to change this set of parameters, cFE allows to validate and update a new table at runtime instead of sending a software patch with hard-coded or statically configured parameters, and then stopping and restarting AOCS application.

An example of typical cFS table is shown in Figure A.8.

Figure A.8 – cFS Table Example.

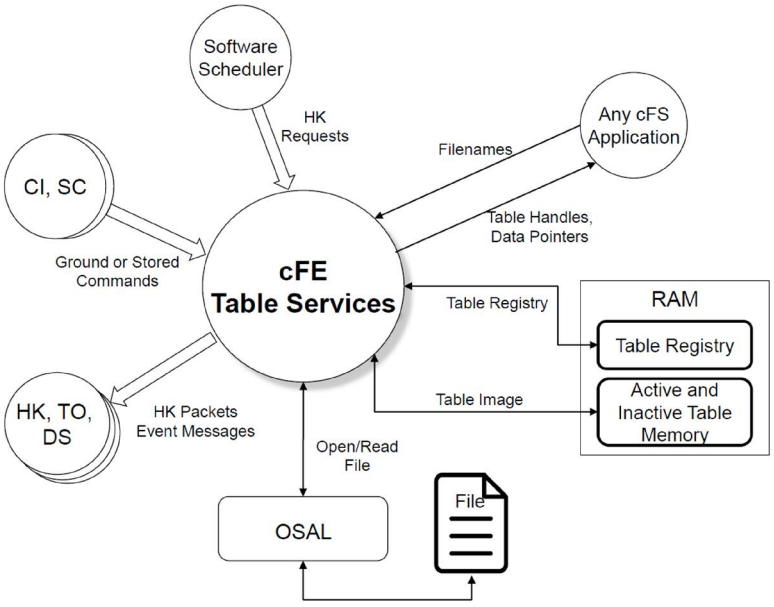
```
/*
** Default schedule table data
*/
SCH_ScheduleEntry_t SCH_DefaultScheduleTable[SCH_TABLE_ENTRIES] =
{

/*
** Structure definition...
**uint8 EnableState -- SCH_UNUSED, SCH_ENABLED, SCH_DISABLED
** uint8    Type        -- 0 or SCH_ACTIVITY_SEND_MSG
** uint16   Frequency    -- how many seconds between Activity execution
** uint16   Remainder    -- seconds offset to perform Activity
** uint16   MessageIndex -- Message Index into Message Definition table
** uint32   GroupData    -- Group and Multi-Group membership definitions
*/

  /* slot #0 */
/*{       SCH_DISABLED,    SCH_ACTIVITY_SEND_MSG,       1,       0,    24,
SCH_GROUP_MD_WAKEUP }, */ /* MD Wakeup */
  { SCH_UNUSED,   0,      0,  0, 0,  SCH_GROUP_NONE},
  { SCH_UNUSED,   0,      0,  0, 0,  SCH_GROUP_NONE},
  { SCH_UNUSED,   0,      0,  0, 0,  SCH_GROUP_NONE},
  { SCH_UNUSED,   0,      0,  0, 0,  SCH_GROUP_NONE},
  { SCH_UNUSED,   0,      0,  0, 0,  SCH_GROUP_NONE},

/* slot #1 */ /* (…) */
/* slot #99 */
};
```

This piece of code shows a typical cFE/cFS table. This particular case is the cFS Scheduler (SCH) table data, responsible for periodically generating software bus messages. This table unitary information is the struct SCH_ScheduleEntry_t, which contains five times the following information ‘uint8 EnableState, uint8 Type, uint16 Frequency, uint16 Remainder, uint16 MessageIndex, uint32 GroupData’. This table contains 100 instances of the struct SCH_ScheduleEntry_t.

Source: Obtained from cFS Scheduler (SCH) v2.2.1 source code (2019).

The relationship between cFE Table Services and the remaining cFS products can be seen in its context diagram, presented in Figure A.9.

Figure A.9 – cFE TBL Context Diagram.



This context diagram shows the relationships between cFE TBL, OSAL and the cFS applications.

Source: Obtained from NASA (2014).

A.7. cFE time services (TIME)

The cFE Time Services (TIME) is the cFE core service that provides time services for applications (MCCOMAS et al., 2016). It contains 25 API's as of cFE v6.5.0, mainly for cFS applications to query the time. cFE TIME also distributes a "time at the tone" command packet, containing the correct time at the moment of the tone signal, typically at 1Hz.

Table A.1 – cFE TIME Definitions.

Concept	Definition
Mission Epoch	An absolute time reference that remains fixed. Example: January 1, 2000
MET (Mission Elapsed Time)	The number of seconds since an arbitrary epoch and is maintained by an on-board oscillator. This is the raw source of time on the spacecraft, because represent a running count of clock ticks since the hardware was initialized
STCF (Spacecraft Time Correlation Factor)	A numeric value used to correlate the MET with the Mission Epoch to obtain the current time
TAI (International Atomic Time)	MET + STCF
UTC (Coordinated Universal Time)	TAI - Leap Seconds

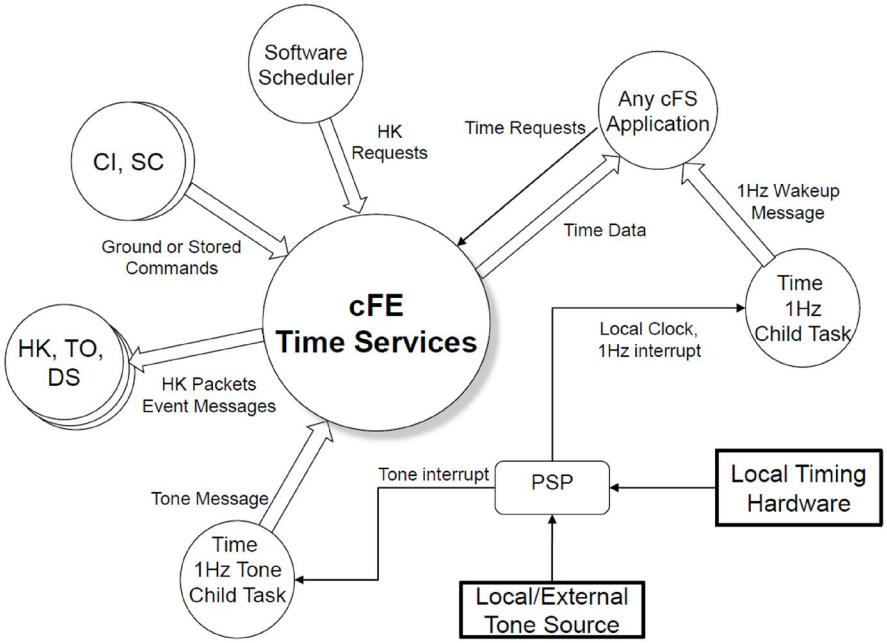
Source: Obtained from Bartholomew and Kobe (2014).

The cFE Time Service nominally correlates time to the International Atomic Time (TAI), but nothing in the cFE Time Service precludes the user from setting the epoch and STCF (Spacecraft Time Correlation Factor) to correlate to a time standard other than TAI (BARTHOLOMEW and KOBE, 2014). The cFE Time Service defines a Spacecraft Time Correlation Factor (STCF) that is applied to the Mission Elapsed Time (MET) to relate the MET and the epoch to the current time.

In addition to TAI, Coordinated Universal Time (UTC) is also commonly desired, so the cFE Time Service provides a UTC value as well. Universal Time (UT) is based on the Earth's rotation and TAI is based on highly precise atomic clocks (BARTHOLOMEW and KOBE, 2014).

The relationship between cFE TIME Services and the remaining cFS products can be seen in its context diagram, presented in Figure A.10.

Figure A.10 – cFE TIME Context Diagram.



This context diagram shows the relationships between cFE TIME, OSAL, PSP and the cFS applications.

Source: Obtained from NASA (2014).

A.8. cFE file services (FS)

The cFE File Services (FS) is a cFE core service that is part of cFE core implementation as of cFE v6.5.0. It is not generally mentioned as the 6th cFE core application because its API's are only used by cFE core services to edit and manipulate file headers and also by cFS File Manager (FM) application to decompress gzip zipped files sent by the ground on-board.

A.9. cFS applications suite

Table A.2 shows cFS applications suite used in this research.

Table A.2 – cFS Open Source Applications Suite.

Acronym	Name	Version	Function
CF	CCSDS File Delivery Protocol (CFDP)	2.2.1	Exchange file data with the ground system using Protocol Data Units (PDUs), being partially compliant with the CFDP standard protocol defined in the CCSDS 727.0-B-4 Blue Book. More information on Section B.3.
CI	Command Ingest	1.0.0	Receive CCSDS TC frames from an external source (such as the ground station or AIT EGSE) over a transport channel and forward the commands to the appropriate application over the cFE Software Bus (SB). More details can be seen on Section B.1.
CI_lab	Command Ingest (lab version)	2.2.0	This is a simplified version of CI application and is not intended for flight. It accepts CCSDS telecommand packets over a UDP/IP port, for quick laboratory tests. It does not provide a full CCSDS Telecommand stack implementation such as CI application.
CS	Checksum	2.4.0	Ensure the integrity of onboard memory. CS calculates Cyclic Redundancy Checks (CRCs) on the different memory regions and compares the CRC values with a baseline value calculated at system start up. CS can ensure the integrity of cFE applications, cFE tables, the cFE core, the onboard operating system (OS), onboard EEPROM, as well as, any memory regions specified by the users.

Acronym	Name	Version	Function
DS	Data Storage	2.5.1	Store user-defined software bus messages in files. These files are generally stored on a device such as a solid-state recorder, but they could be stored on any file system. Another cFS application, such as CF, must be used in order to transfer the files created by DS from their onboard storage location to where they will be viewed and processed.
FM	File Manager	2.5.2	Provide onboard file system management services by processing ground commands for copying, moving, and renaming files, decompressing files, creating directories, deleting files and directories, providing file and directory informational telemetry messages, and providing open file and directory listings.
HK	Housekeeping	2.4.1	Build and send combined telemetry messages (from individual applications) to the software bus for routing. Combining messages is performed in order to minimize overhead and therefore downlink telemetry bandwidth and is also useful for organizing certain types of data packets together.
HS	Health and Safety	2.3.0	Provide functionality for Application Monitoring, Event Monitoring, Hardware Watchdog Servicing, Execution Counter Reporting (optional), and CPU Aliveness Indication (via UART).
LC	Limit Checker	2.1.0	Monitor telemetry data points in a cFS system and compares the values against predefined threshold limits. When a threshold condition is encountered, an event message is issued and a Relative Time Sequence (RTS) command script, from cFS SC, may be initiated to respond/react to

Acronym	Name	Version	Function
			the threshold violation.
MD	Memory Dwell	2.3.1	Monitor memory addresses accessed by the CPU. This task is used for both debugging and monitoring unanticipated telemetry that had not been previously defined in the system prior to deployment.
MM	Memory Manager	2.4.1	Load and dump system memory via command parameters, as well as, from files. MM provides an operator interface to the memory manipulation functions contained in the PSP (Platform Support Package) and OSAL (Operating System Abstraction Layer). It supports symbolic addressing.
SBN	Software Bus Network	1.0.0	Extend the cFE Software Bus (SB) publish/subscribe messaging service across partitions, processes, processors, and networks.
SC	Stored Command	2.5.0	Allow a system to be autonomously commanded 24 hours a day using sequences of commands that are loaded to SC. Each command has a time tag associated with it, permitting the command to be released for distribution at predetermined times. SC supports both Absolute Time tagged command Sequences (ATSs) as well as multiple Relative Time tagged command Sequences (RTSs).
SCH	Scheduler	2.2.1	Provide a method of generating software bus messages at pre-determined timing intervals. This

Acronym	Name	Version	Function
			allows the system to operate in a Time Division Multiplexed (TDM) fashion with deterministic behavior. The TDM major frame is defined by the Major Time Synchronization Signal used by the cFE TIME Services (typically 1 Hz). The Minor Frame timing (number of slots executed within each Major Frame) is also configurable.
SCH_lab	Scheduler (lab version)	2.2.0	This is a simplified version of SCH application and is not intended for flight. It generates periodic hard-coded defined messages with one second resolution, for quick laboratory tests.
TO	Telemetry Output	1.0.0	Transmits CCSDS TM frames to an external destination (such as the ground station or AIT EGSE) over a transport channel, containing programmable cFE Software Bus (SB) messages. More details can be seen on Section B.1.
TO_lab	Telemetry Output (lab version)	2.2.0	This is a simplified version of TO application and is not intended for flight. It sends CCSDS telemetry packets over a UDP/IP port, for quick laboratory tests. It does not provide a full CCSDS Telecommand stack implementation such as TO application.

Source: Made by the author.

APPENDIX B – cFS AND CCSDS DATA PROTOCOLS

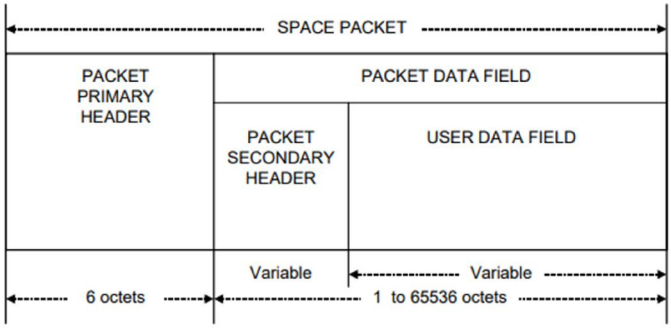
B.1. cFS and CCSDS space packet protocol

CCSDS Space Packet Protocol (CCSDS 133.0-B-1) standardizes the data protocol to be exchanged over a network that involves a ground-to-space or space-to-space communications link (CCSDS, 2003).

The protocol data unit is the Space Packet, presented in Figure B.1. It is composed of a mandatory primary header of 6 bytes (or octets), presented in Figure B.2, an optional variable-size secondary header and the variable-size user data field.

The secondary header, if present, is positioned contiguously after the primary header. It is composed of at least time or ancillary information, or both, as shown in Figure B.3, in that order.

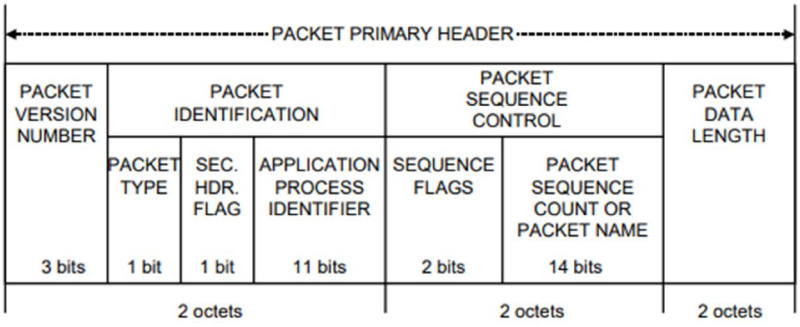
Figure B.1 – CCSDS Space Packet.



CCSDS Space Packet is the protocol data unit to be used in message exchanges according to the protocol.

Source: CCSDS 133.0-B-1, CCSDS (2003).

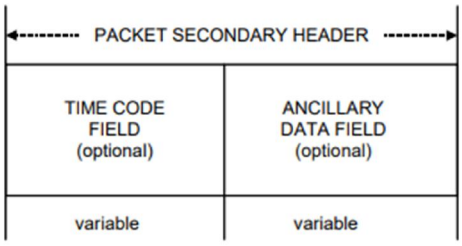
Figure B.2 – Space Packet Primary Header.



The Space Packet Primary Header constitutes the first 6 bytes of the CCSDS Space Packet. It contains packet identification, sequence control and data length.

Source: CCSDS 133.0-B-1, CCSDS (2003).

Figure B.3 – Space Packet Secondary Header.



The Space Packet Secondary Header, if present, comes just after the primary header in the CCSDS Space Packet. It is of variable size and contains time code or/and ancillary information.

Source: CCSDS 133.0-B-1, CCSDS (2003).

The CCSDS headers definitions used in cFS are present in cFE Software Bus core application, which can be seen in Section A.5. This application intermediates all message exchanges between software applications in cFS framework.

It is possible to see in *ccsds.h* include file of cFE that cFS command headers (CCSDS_CmdPkt_t) are constituted of a compliant CCSDS Primary Header (CCSDS_PriHdr_t) and a tailored secondary header (CCSDS_CmdSecHdr_t), comprised of 2 bytes, that contains function code and checksum.

Also, cFS telemetry headers (CCSDS_TlmPkt_t) are constituted of a compliant CCSDS Primary Header (CCSDS_PriHdr_t), same as for commands, and a secondary header (CCSDS_TlmSecHdr_t) that contains time information only.

It is immediate to conclude then that cFS Software Bus is fully compliant with CCSDS Space Packet Protocol.

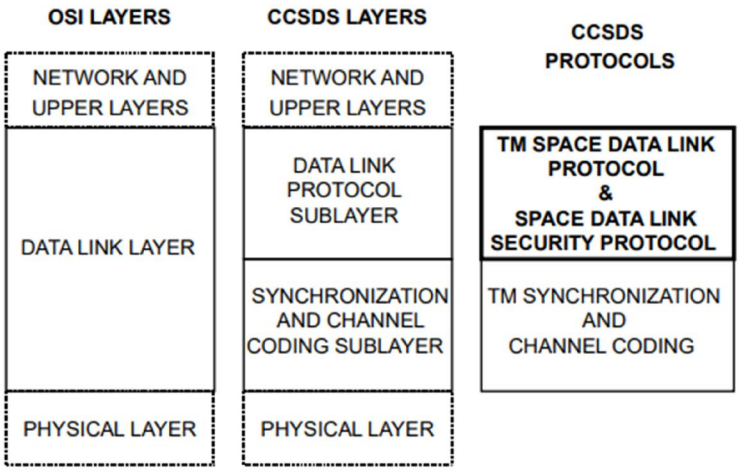
B.2. cFS and data link layer CCSDS protocols

CCSDS Space Data Link Protocol (SDLP) is a Data Link Protocol that was designed to meet the requirements of space missions for efficient transfer of space application data (CCSDS, 2015). The TM and TC SDLP corresponds to the Logical Link Sublayer, as shown in Figure B.4, and provides functions of transferring various data using a fixed-length protocol data unit called the Transfer Frame.

Immediately below Data Link Protocol Sublayer, there is the Synchronization and Channel Coding Sublayer. This is the last layer before physical layer, responsible for error-control coding/decoding, Transfer Frame delimiting/synchronizing, and bit transition generation/removal (CCSDS, 2017).

TM Space Data Link Protocol (SDLP) is defined in standard reference CCSDS 132.0-B-2 (CCSDS, 2015a). This procedure is a Data Link Layer protocol to be used over space-to-ground or space-to-space communications links by space missions. At the synchronization sublayer level, there is the CCSDS 131.0-B-3 (CCSDS, 2017), which standardizes TM synchronization and channel coding.

Figure B.4 – Equivalence between CCSDS and OSI layers.



The equivalence between CCSDS and OSI (Open System Interconnection) reference model layers.

Source: CCSDS 132.0-B-2, CCSDS (2015).

In cFS Applications Suite, there is an application called Telemetry Output (cFS TO), along with its supporting I/O Library (cFS IO_LIB), which is devoted to subscribing to other cFS applications telemetry data, multiplexing and formatting them and finally sending TM frames over user-defined physical channels.

cFS Telemetry Output (TO), along with its accompanying library IO_LIB, contains a highly parameterized CCSDS SDLP implementation. User can define several CCSDS options, telemetry routing and protocol stacks to be applied on each route.

This application, in “multi_tf” example configuration, has the following characteristics:

- It doesn’t implement Space Data Link Security (SDLS) option.
- User can define the following parameters: VC_ID, SC_ID, TM Frame Length, size of overflow buffer, etc.
- It implements only one Virtual Channel per Master Channel, to simplify the design.

- It implements Virtual Channel Packet (VCP) format.
- Operational Control Field (OCF) is implemented using CLCW Type-1-Report and associated to the Master Channel, but its use is optional.
- Error Control Field Encoding is implemented using CRC of 16 bits as per CCSDS scheme, but its use is optional.
- Secondary Frame Header, which is optional according to the standard, is not implemented.
- ASM marker (1ACFFC1D) attached in the frame for synchronization purposes.
- No TM coding technique is applied.
- (Optional) Pseudo-randomizer applied in the CADU frame.

It is important to emphasize that TO and IO_LIB applications can be customized if the user wants to add CCSDS SDLP and CCSDS TM Synchronism functionalities not already implemented.

On the telecommand chain, there is a reciprocal protocol called TC SDLP, which is defined in standard reference CCSDS 232.0-B-3 (CCSDS, 2015b). It defines the Data Link Layer protocol to be used over ground-to-space or space-to-space communications links. At the frame synchronism level, there is also the reciprocal CCSDS 231.0-B-3, which standardizes TC synchronization and channel coding.

cFS Command Ingest (CI), along with its accompanying library IO_LIB, is the reciprocal application to cFS TO. This application does all necessary format translation on received uplink command frames such that commands sent to the software bus are CCSDS Space Packet Protocol packets.

This application, in “multi_tf” example configuration, has the following characteristics:

- It presupposes BCH coding, identifying CLTU start sequence (EB90) and tail sequence (C5C5 C5C5 C5C5 C579).

- Transfer frame extraction from CLTU, extracting 7 information bytes out of 8 total bytes in each code block.
- Decoding procedures on parity bits at each code block for error control are not implemented.
- Process frame with COP-1 protocol, creating CLCW message and sending it to TO application that will transmit it back over to the ground segment.
- (Optional) De-randomize frame, if applicable.
- Extract commands from the frame and sending them to the software bus, which will then forward messages to subscriber applications.

The same comment of TO app applies here: CI application can be customized if the user wants to add CCSDS SDLP and CCSDS TC Synchronism functionalities not already implemented.

The following CCSDS protocols are thus deemed to be partially or totally implemented in cFS, through TO, CI and IO_LIB routines, as shown in Table B.1.

Table B.1 – CCSDS Data Link protocols comparability with cFS functions.

Protocol	Function	Correspondence in cFS applications
TM SDLP (CCSDS 132.0-B-2)	Transfer Frame Handling	TO (to_custom.c) Implemented
	Transfer Frame Secondary Header (Optional)	TO (to_custom.c) Implemented, but not used
	Master and Virtual Channel Handling	Only one virtual channel implemented per master channel TO (to_custom.c) Partially implemented
	VCA Services	TO (to_custom.c) Implemented, but not used
	Operational Control Field (Optional)	TO (to_custom.c) Implemented
	Frame Error Control Field (Optional)	IO Lib (tmtf.c) Implemented, but not used
TM Sync and Coding (CCSDS 131.0-B-3)	Coding Method implementation (Convolutional, Reed-Solomon, Turbo, LDCP)	IO lib (tm_sync.c) Not implemented
	ASM Marker inclusion in Transfer Frame	IO lib (tm_sync.c) Implemented
	Pseudo-Randomizer implementation	IO lib (io_lib_utils.c) Implemented
TC SDLP (CCSDS 232.0-B-3)	TC Frame Primary Header Handling	IO lib (tctf.c) Implemented
	SDLS Option (optional)	CI (ci_custom.c) Not implemented
	Master and Virtual Channel Handling	Only one virtual channel implemented per master channel CI (ci_custom.c) Partially implemented
	CLCW Generation (COP-1)	IO lib (cop1.c) Implemented
	MAP Services (along with Segment Header)	CI (ci_custom.c)

Protocol	Function	Correspondence in cFS applications
	(optional)	Not implemented
	Frame Error Control Field (optional)	CI (ci_custom.c) Not implemented
TC Sync and Coding (CCSDS 231.0-B-3)	Search for start sequence	The start sequence 0xEB90 is localized and synchronized. IO lib (tc_sync.c) Implemented
	BHC or LDCP decoding	BHC is expected, but parity check decoding not implemented IO lib (tc_sync.c) Not implemented
	Random Sequence Removal (optional)	IO lib (io_lib_utils.c) Implemented
COP-1 (CCSDS 232.0-B-2)	FARM-1 (Type A and Type B)	IO lib (cop1.c) Implemented

Source: Made by the author.

B.3. cFS and CCSDS CFDP protocol

CCSDS File Delivery Protocol (CFDP) is standardized in the document CCSDS 727.0-B-4 (2007). This recommendation defines a protocol suitable for the transmission of files to and from spacecraft data storage. In addition to the purely file delivery related functions, the protocol also includes file management services to allow control over the storage medium (CCSDS, 2007).

This protocol classes 1 and 2 in immediate mode have been implemented and used in cFS framework, through cFS CF application, at several missions, including NASA GSFC Global Precipitation Measurement (MCCOMAS, 2016). CFDP relies on the existence of a file system in the FSW, which is also mandatory as per OSAL requirements as shown in Chapter 4.2.1.

cFS CF does not support filestore requests, i.e., file management services such as create directories, copy files, etc. There is another cFS application devoted to these services (cFS FM).

Wilmot (2012) makes a balance of the advantaged of using CFDP in NASA Flight Software architecture:

- CFDP provides a standard protocol to reliably transfer files over higher bit-error links
- CFDP works well over highly asymmetric links
- CFDP has been shown to deliver more complete science data over a shorter contact time
- File systems and CFDP have reduced operations cost in NASA missions

CFDP classes 3 and 4, which aim is to support file transfer through the mediation of one or more waypoints, a use-case that can for example happen in satellite constellations, were not implemented in cFS CF app v2.2.1.

APPENDIX C – REMARKS ON cFS COMPLIANCE WITH ECSS PUS

This appendix provides additional remarks about cFS compliance with ECSS PUS services, as a complement to Section 5.2.

Service 1:

- Using cFE EVS API's in TC receipt and handling should be analog to a report of request verification. It become is a design rule proposed in Section 6.3.
- During operations, it is considered simpler to just verify command correct receipt through each application command counter and command error counter, as proposed in Section 6.3

Service 2:

- This service is analog to CCSDS SOIS Command and Data Acquisition Services, mentioned in Section 5.1.1. Same comment applies here

Service 3:

- The parameter report structures used by the housekeeping service is predefined on-board through cFS SCH tables definition. The tables telemetries groups and super groups can be changed in run-time through table upload and activation
- Super commutated parameters not implemented
- No difference between housekeeping and diagnostic parameters
- Telemetries should be enabled in cFS TO table in order to be downloaded to the ground

Service 4:

- There is no cFS built-in functionality that provides the maximum, minimum, mean and standard deviation values of on-board parameters during a specified time interval

- This service could be used to reduce the quantity of data that is systematically reported to the ground and could be a potential subject for a new cFS application or cFS DS app upgrade
- In current cFS version, cFS DS app filters the packages that will be stored by means of sub-sampling the incoming data according to user-defined rates

Service 5:

- Fully compliant

Service 6:

- Almost fully compliant, except for the scrubbing memory sub-service. Nevertheless, this is a functionality that is well known to be mission-specific, being hard to propose a generic reusable implementation in cFS. This is a topic that can be further explored by cFS developers

Service 8:

- This service states that a given application process can support one or more functions that are invoked from the ground
- ECSS (2016) says that this function remains in the current version only for backward compatibility reasons and encourages to develop mission-specific service types to supersede this capability
- For instance, some cFS applications like cFS CF need a pace message (wake_up command) to execute their functions. Originally, these messages are supposed to be issued by cFS SCH app, but nothing prevents it to be sent from ground

Service 9:

- cFS is compliant with CCSDS CUC time format as presented in Section A.5
- cFE TIME was presented in Section A.7 and has more built-in functions than Service 9 recommends, such as time correlation commands

Service 11:

- No sub-schedules (optional) implemented
- In cFS SCH app, groups are created in compilation time and cannot be created or changed in run-time
- Time-shifting all scheduled activities involves a new cFS SCH table upload
- There is no cFS SCH implemented report that provides visibility of each scheduled activity details, such as release time, group identifier, etc.

Service 12:

- cFS LC implements the minimum capabilities specified by Service 12 parameter monitoring subservice
- cFS LC has no implemented capability of detecting delta change (optional) in parameter value
- cFS HS implements the minimum capabilities specified by Service 12 functional monitoring subservice, understood as cFS applications execution monitoring

Service 13:

- This service is a generic version of the CCSDS CFDP protocol. cFS compliance to this protocol was already performed in Section B.3

Service 14:

- cFS TO application can provide real-time forward control capabilities to any cFE SB existing message, which can virtually be any on-board parameter or cFS CF PDU message for large file transfers
- cFS TO doesn’t provide the Service 14 optional subservice that define specific control conditions for housekeeping, diagnostic and event reports. There is no differentiation of cFE SB message treatment based on its origin.

Service 15:

- cFS DS constantly checks if a packet storage file maximum age or maximum size has been reached. If so, this file is closed, and a new file is opened to accommodate new incoming messages. Therefore, Service 15 bounded management policy is implemented, but circular management is not
- cFS doesn't implement a turn-key version for Service 15 "open retrieval" and "by-time-range" modes. Retrieving files is a multi-step process that starts by closing packet store files by ground or on-board requests using cFS DS functions and subsequently downloading these files to the ground segment using cFS CF services (Service 13)

Service 17:

- The capability to perform an end-to-end test under ground control in the form of an "are-you-alive" function is implemented in cFS as NOOP command. This command only effect is to increase the on-board application counter of successful received commands and also returns a successful event message
- In Section 6.3, the presence of a NOOP (No-Operations) command as the first one (function code = 0) in every cFS application is proposed as mandatory design requirement

Service 18:

- OBCP is a separate ECSS standard, identified by code ECSS-E-ST-70-01C, and is also recommended by Service 18 service
- cFS has no dedicated runtime engine interpreter of operational scripts on-board
- So far, what one operator can do is upload a new cFS application and start it in runtime or chose an easier path and upload a cFS SC time-sequenced commands table (which is implementation of Service 21). OBCPs would be a middle-term operational solution in

terms of flexibility that could be implemented in a future cFS application

Service 19:

- cFS HS application is fully compliant with this service

Service 20:

- Parameters can be changed in runtime using cFE TBL services. Although, a variable in a table shall be created in compilation time (fixed-size tables). New variables cannot be created in runtime though
- Tables in cFS are decentralized, i.e., they are tied to a certain cFS application. There is no Service 20 recommended parameter ID property that is unique within the context of the spacecraft

Service 21:

- cFS SC is fully compliant with this service

Service 22:

- There is a deterministic relation between orbit-position and time, which can be found through flight dynamics calculation. This service is therefore very much similar to Service 21, except that instead of directly relying on time as the trigger for the sequenced commands, it has an indirect dependence on time, because Service 22 counts on orbit position
- This service is currently not supported by cFS. Although, it is an interesting proposition that could be further investigated to be implemented in a cFS application. This application could start from cFS SC app implementation associated perhaps with a generic flight dynamics engine

Service 23:

- cFS FM application is fully compliant with Service 23 file handling subservice

- cFS FM application is compliant with Service 23 file copy subservice provided that it is between files systems on-board the spacecraft
- cFS CF application, as explained in Section B.3, doesn't implement filestore requests to manage files in destination peer after file transfer between ground to space. Therefore, copying a file from space to ground or vice-versa is possible but not through a turn-key service. This could be a future upgrade on cFS CF