A Software Framework for Nanosatellites based on CCSDS Mission Operations Services with Reference Implementation for ESA's OPS-SAT Mission

by

César Coelho

Submitted to

Graz University of Technology

in collaboration with the

European Space Agency

under the Networking Partnering Initiative

In partial fulfilment of the

requirements for the degree of

Doctor of Engineering Sciences (Dr. techn.)

Institute of Communication Networks and Satellite Communications

Graz, November 2017

Supervisors
Univ.-Prof. Dipl.-Ing. Dr. Otto Koudelka
Institute of Communication Networks and Satellite Communications
Graz University of Technology

Dr. Mario Merri
Mission Data Systems Division
European Space Agency

Dr. Mehran Sarkarati
Applications and Special Projects Section
European Space Agency

Referee
Prof. Dr.-Ing. Sabine Klinkner
Institute of Space Systems
University of Stuttgart

The European Space Agency supported this research through the Network Partnering Initiative programme by providing co-funding, access to the European Space Operations Centre laboratories, technical support, and networking opportunities.

Defended in Graz, Austria on the 12th of December 2017.

“Don't Call Us, We'll Call You”

- Hollywood Principle

STATUTORY DECLARATION

I, César Bichinho Whittle Coelho (César Coelho), born in Lisbon, declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly indicated all material which has been quoted either literally or by content from the sources used. The text document uploaded to TUGRAZonline is identical to the present doctoral thesis.

Date	Signature

Table of Contents

Abstract

Traditional European space missions exchange information with spacecraft using an old packet-based standard from 1994, in an era where mobile phones had the size of a brick and the internet was nearly born. Since then, many innovations appeared in the field of information and communications technologies that shaped the way one exchanges information on earth. For example, the rising market of smartphones and tablets brought new ideas into software by providing quick development of applications by taking advantage of software frameworks. In contrast, on-board computers still have simple monolithic software architectures with little reuse due to the low processing capabilities of current on-board computers. However, new and more advanced on-board computers are starting to be commercialized specifically for nanosatellites. Nanosatellites are small satellites that became increasingly more popular for the past 5 years because they are cheaper to launch without compromising a lot of functionality. The first nanosatellite mission from ESOC is OPS-SAT, a mission open to worldwide experimenters that can try new mission operation concepts and ideas. The Consultative Committee for Space Data Systems (CCSDS) recently defined a service-oriented architecture for mission operations of space assets, known as CCSDS Mission Operations services, which is intended to fly for the first time in OPS-SAT. The research defines the NanoSat MO Framework, a software framework for nanosatellites based on CCSDS Mission Operations services, including its reference implementation. Additionally, there is a Software Development Kit (SDK) in order to facilitate the development of software based on the NanoSat MO Framework. OPS-SAT experimenters can use this SDK for quick development of software capable of running on ground and/or in space. Although the reference implementation of the NanoSat MO Framework is generic to any nanosatellite mission, a dedicated mission implementation for OPS-SAT was developed. A dedicated Flatsat was built, which allows functional verification and validation of the software with the real hardware on a flat surface. In addition, its performance was analysed and improved where needed. A bright outlook is envisioned for the NanoSat MO Framework.

Acknowledgements

I am very grateful to my supervisors, Professor Otto Koudelka, Mehran Sarkarati, and Mario Merri, who provided me the great opportunity to pursue a PhD with enough room for creativity freedom. Professor Koudelka is a very thoughtful person who is always ready to solve problems, and Mehran has the ability of assembling great multidisciplinary teams that can get things done. It is important to acknowledge that this research would not be possible without Mario’s previous work that took him many years of consistent effort. Isaac Newton once stated “If I have seen further, it is by standing on the shoulders of giants”.

My family has always been able to support me whenever I needed them. My parents taught me the value and importance of education since I was young and they helped me moving in the right direction. I am very thankful for having such a fortunate family that allows me to reach my life goals without impeding my way.

Special thanks to Sam Cooper, for his help and deep expertize in the technical field of this research, I did learn a lot from him. Thanks to Professor Klinkner who dedicated some time in being the external referee of this thesis. Thanks to Valeriia Makarova for listening to my rehearsals for conference presentations. I would also like to thank Steffen Bamfaste for being a great labmate who helped with the assembly of the flatsat, and played many times as a live “rubber duck” for the most enigmatic bugs in the software. Thanks to Eric Thomas for motivating me to finish the PhD in 3 years.

The CCSDS efforts must also be acknowledged as it puts thousands of experts working together to agree on a set of standards to be used internationally. The development of standards for space data systems facilitate future cooperation all around the world and beyond.

Finally, I would like to thank the European Space Agency for the opportunity of collaborating in a friendly and international environment in the European Space Operations Centre (ESOC) facilities with a great team of Engineers in OPS-GDA.

Acronyms

A-I

ADCS	Attitude Determination and Control System
AGSA	Advanced Ground Software Applications
APEX	Advanced Processor Core for Space Exploration
APID	Application Process Identifier
APK	Android Package Kit
ARM	Advanced RISC Machine
CAN	Controller Area Network
CFP	CAN Fragmentation Protocol
CCSDS	Consultative Committee for Space Data Systems
CFDP	CCSDS File Delivery Protocol
cFS	Core Flight Software
CD	Continuous Delivery
CI	Continuous Integration
CNES	Centre National d'Études Spatiales
COM	Common Object Model
CRUD	Create, Read, Update, and Delete
CTT	Consumer Test Tool
DLR	Deutsches Zentrum für Luft- und Raumfahrt e.V.
ECSS	European Cooperation for Space Standardization
EDS	Electronic Data Sheets
EGOS	ESA Ground Operation System
EGS-CC	European Ground Systems - Common Core
ESA	European Space Agency
ESOC	European Space Operations Centre
EUD	EGOS User Desktop
GPS	Global Positioning System
GUI	Graphical User Interface
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
I^2C	Inter-Integrated Circuit (I^2C is pronounced: I-squared-C)
ICD	Interface Control Document
IP	Internet Protocol
IPC	Inter-Process Communication

J-Z

JDBC	Java Database Connectivity
JPL	Jet Propulsion Laboratory
JVM	Java Virtual Machine
JPA	Java Persistence API
M&C	Monitor and Control
MAL	Message Abstraction Layer
MCPs	Mapping Configuration Parameters
MO	Mission Operations
NASA	National Aeronautics and Space Administration
NIS	Network Interface System
NMF	NanoSat MO Framework
NPI	Network Partnering Initiative
OBSM	On Board Software Management
OBSW	On-Board Software
OMCS	OPS-SAT Mission Control System
OS	Operating System
OSRA	On-board Software Reference Architecture
PUS	Packet Utilization Standard
RISC	Reduced Instruction Set Computing
RMI	Remote Method Invocation
RHESE	Radiation Hardened Electronics for Space Environments
SAVOIR	Space Avionics Open Interface Architecture
SCOS	Satellite Control and Operation System
SDK	Software Development Kit
SDR	Software-defined Radio
SM	Software Management
SM&C	Spacecraft Monitor and Control
SOA	Service-Oriented Architecture
SOCIS	Summer of Code in Space
SPP	Space Packet Protocol
SQL	Structured Query Language
TC	Telecommands
TCP	Transmission Control Protocol
TCP/IP	TCP over IP
TM	Telemetry
URL	Uniform Resource Identifier
WG	Working Group

Chapter 1

1. INTRODUCTION

The growing market of smartphones and tablets brought new ideas into software by providing quick software development using well defined libraries from Android and iOS. Additionally, Android gave the community an extra degree of freedom by making the mobile apps portable entities of software where the same app can run on the hardware of different vendors as long as the device has the same underlying Android framework.

In the computer science field, many software frameworks exist to solve specific problems that affect a common set of software developers. They usually provide generic functionality that addresses the low-level details, whereas the developer only needs to add the missing parts that are application-specific. At the execution level, software frameworks follow the Hollywood Principle: "Don't call us, we'll call you." [1]

However, the spacecraft's on-board software is still seen as immutable software developed for a certain mission where updates are only performed in very rare instances and if there is something fundamentally wrong. Such updates are done by patching specific memory areas via Telecommands sent from ground. These activities are considered very risky as they can jeopardize the whole mission and make an expensive satellite inoperative.

The price of a space launch is mainly driven by the mass of the spacecraft carried as payload, which consequently forces universities and small institutions that are interested in doing research in space science and technology to work with smaller spacecraft designs. Hence, nanosatellites have become very popular and a new market suddenly appeared. Given the fact that these missions, most of the times, are non-critical and with educational purposes, projects are willing to take more risk in exchange of a cheaper mission price tag and so, the approach of using commercial of-the-shelf products rather than dedicated hardware is common practice.

The European Space Agency in partnership with the Graz University of Technology intends to launch OPS-SAT, a nanosatellite dedicated to testing new on-board software in space and new mission operations concepts. The payload of this "flying laboratory" comprises a very powerful system-on-chip capable of running a Linux-based operating system and also capable of running Java applications. World-wide experimenters will then submit their software to run it in OPS-SAT, a safe and reliable nanosatellite testbed. [2]

The Consultative Committee for Space Data Systems (CCSDS) is an international organization responsible for producing recommendations and standards for space data systems based on collected experience from previous space missions. Within the CCSDS, the Mission Operations and Information Management Services Area guarantees that application standards exist to facilitate the smooth transition of space mission information between the "mission operations" systems and the "mission utilization" systems. A stratified, service-oriented architecture was defined by the Spacecraft Monitor and Control Working Group in order to define services for mission operations of space assets. This is known as the CCSDS Mission Operations services and it unlocks the possibility for other Working Groups to define services that are relevant to their specific subject area while using the same underlying architecture. [3]

The objective of the research is to design a software framework for nanosatellites based on CCSDS Mission Operations services inspired by today's smartphone technologies. The design is abstracted from any specific nanosatellite platform allowing the development of software in form of "apps" that are portable entities of software that can be reused by different nanosatellite platforms. As part of the research, a reference implementation was developed, open-sourced and made available to the general public together with a Software Development Kit that facilitates the development of software. The framework is named NanoSat MO Framework and it is expected to have its first flight in the future OPS-SAT mission.

1.1. Partner's strategic interests

The research was developed by the Graz University of Technology in collaboration with the European Space Agency (ESA) under the Networking/Partnering Initiative (NPI).

ESA's NPI supports investigation done by universities on advanced technologies for space applications with the objective of promoting interactions between ESA and European universities or research institutes. [4]

The European Space Operations Centre (ESOC) is the main mission control centre for ESA and it is located in Darmstadt, Germany. It is not only responsible for controlling many European spacecraft in orbit but also developing the necessary systems on ground that support missions in space. [5]

In an attempt to advance competences for the future, ESOC is leading the specification of the Mission Operations services standardization efforts in CCSDS. At the same time, ESOC is developing OPS-SAT's ground segment to monitor and control the spacecraft and manage the experiments, by receiving, uploading and deploying new and experimental software on-board according to a scheduled plan. [2]

While ESOC is developing OPS-SAT's ground data systems, Graz University of Technology (TU Graz) is the technical lead organization for the development of the flight segment. As the OPS-SAT mission requires continuously changing the OBSW to validate new operational scenarios, a software framework has been devised that could be delivered to experimenters in order to facilitate their development of the software experiments. Additionally, TU Graz plans to fly 2 experiments on-board of OPS-SAT, respectively a "Spectrum Analyser in Space", and an "Optical Uplink experiment using a LASER Ranging Station".

TU Graz has previously acquired experience with nanosatellites from the previous involvement on the TUGSAT-1 (BRITE-Austria) mission. The Institute of Communication Networks and Satellite Communications from TU Graz includes many facilities ready to be used for the mission development, such as: Thermal Chamber, (Thermal) Vacuum Chamber, Vibration Table, Anechoic Chamber, Clean Room, and Ground Stations.

ESA intends to use the product of this research in future missions, in particular operating reconfigurable on-board software in space and preparing for future service based operations.

1.2. Nanosatellites and Space Processors

The recent miniaturization of space components and electronics has allowed the design of smaller satellites which are considerably cheaper to build and launch than conventional satellites. This decrease in the total cost of a space mission has boosted a new growing market for small satellites and, as the number of small satellites keeps increasing, there is a raising demand for reusable software across nanosatellites.

Nanosatellites (or "nanosats") are artificial satellites with a wet mass between 1 and 10 kg. M any nanosatellites are based on the “CubeSat” standard developed by California Polytechnic State University and Stanford University which consists of any number of 10 cm x 10 cm x 10 cm units. [6]

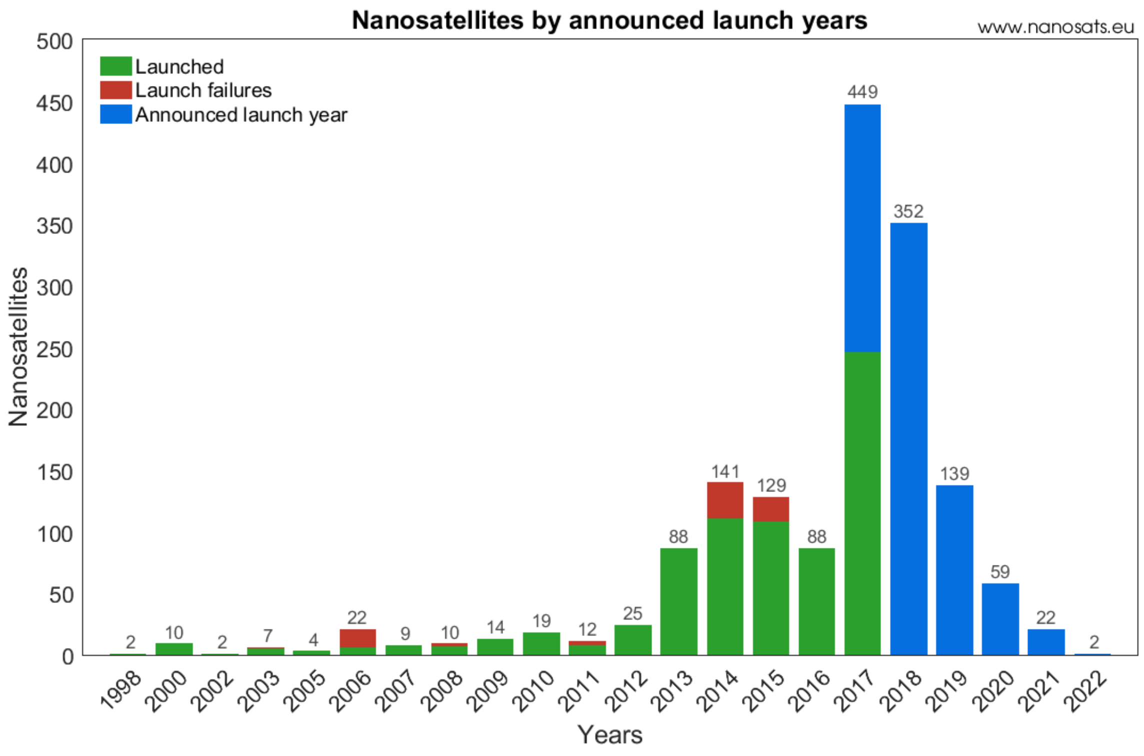


Figure 1: Number of nanosatellites announced by year. [7]

According to the Figure 1, it is possible to visualize a large increase in the number of launched nanosatellites since 2013 against previous years. It is also possible to visualize that in 2017 there is a sharp increase in the number of nanosatellites for that year. The main responsible for this major increase is a nanosatellites’ constellation by Planet (previously Planet Labs, Inc) for earth observation. [8]

In 1965, the founder of Intel Dr. Gordon Moore observed and described that the number of transistors in an integrated chip doubles approximately every 2 years; this is the so called Moore's "Law". The trend appeared to be accurate until recent times when the size of the circuits got so small that the laws of quantum physics can no longer be dismissed. A similar prediction to Moore's "Law" was done by Intel's executive David House, who has estimated that the chip performance would double every 18 months. [9] [10]

Spacecraft on-board computers are not as powerful as the cutting-edge commercial off-the-shelf computers available on retail stores for computer enthusiasts. This happens because the former need to be simultaneously low-power, radiation-hardened, high-throughput, and flight-validated. [11]

The Radiation Hardened Electronics for Space Environments (RHESE) project is advancing the current technology in radiation-hardened electronics by developing high performance devices capable of withstanding the radiation requirements in space. Through the project, it was observed that radiation-hardened processors lag commercial devices by several technology generations. Figure 2 shows the 2 different processor types plotted together and it is possible to visualize a 10 years lag between them. [12]

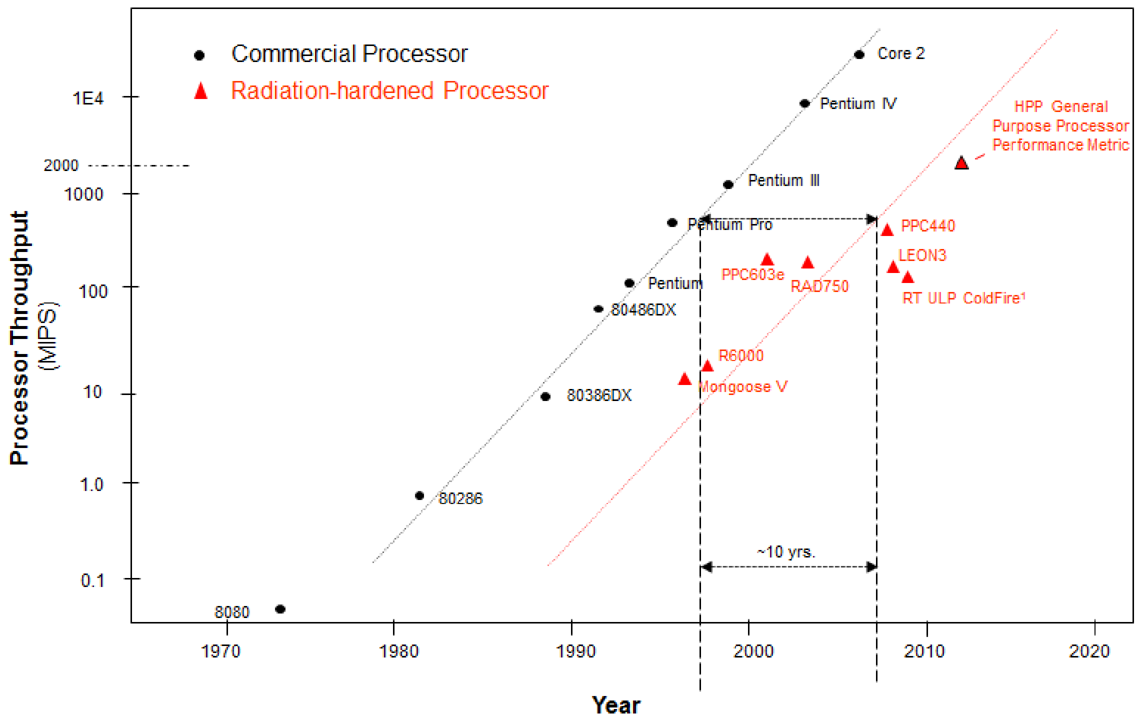


Figure 2: Processor throughput of Commercial Processors and Radiation-hardened Processors in function of their emergence

To tackle the problem above, the European's Seventh Framework Programme (FP7) has funded the Advanced Processor Core for Space Exploration (APEX) project from 2014 to 2016, to conduct pre-commercial research to lay the foundations and establish the mechanisms for the development of a fault and radiation tolerant ARM processor to be used in the computing systems of future space missions. The system on chip of the APEX is a Xilinx Zynq-based which includes an ARM Cortex-A9 processor capable of running a Linux-based operating system. [13] [14]

As of 2017, not many nanosatellites have flown powerful processors but the landscape is expected to change as the nanosatellite industry is starting to take the first steps in selling space-qualified on-board computers. One concrete example is the product NanoMind Z7000 from GOMSpace which also contains a Xilinx Zynq-based system on chip. [15]

The European Space Agency in partnership with the Graz University of Technology intends to launch OPS-SAT, a nanosatellite dedicated to testing new on-board software in space and new mission operations concepts. Since a powerful experimental platform needs to be available, a MitySOM-5CSx system on module was selected which includes an Altera Cyclone V featuring a dual-core ARM Cortex A9 processor however the system on module is not radiation-hardened and so, two units will be present for redundancy. [16]

1.3. ESA OPS-SAT mission

The criticality of space assets has made many project managers willing to trade innovation for the sake of reliability. As a direct consequence, we now witness the continued reuse of old software while new ideas do not fly. A catch-22 situation is experienced as new concepts won't fly because they have never flown before.

ESA and its European industry partners generate many new and innovative ideas for advancing European space technology regarding mission operations every year, but the majority of these innovations never make it to orbit. OPS-SAT emerged, providing a low cost in-orbit laboratory available for authorized experimenters to test, demonstrate and validate their software experiments. OPS-SAT is the first CubeSat designed by ESOC and is a safe experimental platform that shall fly in a LEO dawn-dusk orbit. OPS-SAT makes available a reconfigurable platform, at every layer from channel coding upwards, and it will be available for experimenters wishing to test and demonstrate new software and mission operation concepts. [17]

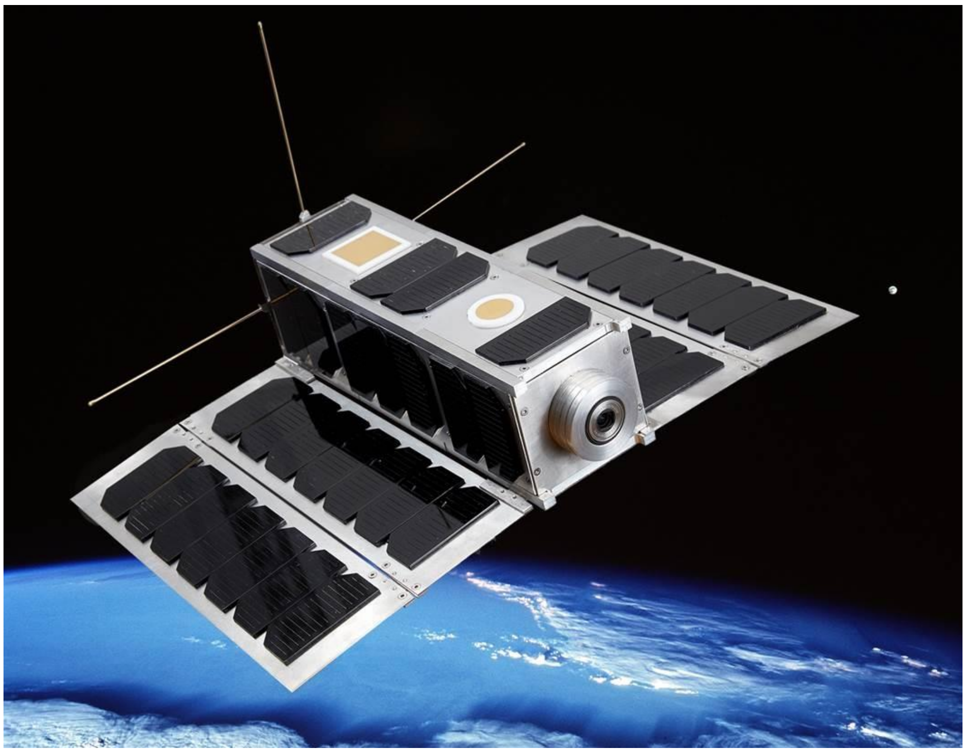


Figure 3: Artistic rendering of OPS-SAT flying around the Earth [18]

The mission is devoted to the demonstration of the arising improved capabilities in mission operations when spacecraft are capable of running faster on-board computers. It contains an experimental platform computer that is ten times more powerful than any current ESA spacecraft. OPS-SAT mission has the very clear objective of breaking the cycle: “has never flown, will never fly” mentioned above, in the area of satellite control. [19]

Graz University of Technology is the technical lead organization for ESA's OPS-SAT mission, a robust and low-cost triple-cube satellite capable of running experimental software on a very powerful Processing Platform composed of two MitySOM-5CSx system on module which include an Altera Cyclone V featuring a dual-core ARM Cortex A9 processor, a 1 GB DDR3 RAM and 8 GB of mass memory. [20] [16]

Due to the risk-free mission design, ESA OPS-SAT satellite provides an excellent opportunity to test the latest standards from the CCSDS such as CCSDS File Delivery Protocol (CFDP) for space file transfer and CCSDS Mission Operations (MO) services for mission operations of space assets. [21]

In 2014, a Proof of Concept demonstrated that it was possible to run a java implementation prototype of two CCSDS Mission Operations services (Parameter service and Aggregation service) on a MitySOM-5CSx device. [22]

1.4. CCSDS Mission Operations

Spacecraft not only tend to produce massive amounts of telemetry data every single day but also need to be remotely controlled in order to accomplish the mission goals. Currently in ESA missions, this exchange of information is done by using a decades old packet-based standard known as Packet Utilisation Standard (PUS). The PUS was developed in 1994, in an era where mobile phones had the size of a brick and the internet was nearly born. [23] [24]

To tackle this problem, the Spacecraft Monitoring & Control (SM&C) Working Group of the Consultative Committee for Space Data Systems (CCSDS) defined a service-oriented architecture for space mission operations, in a set of standardised, interoperable mission operation services, which allow rapid and efficient construction of co-operating space systems. [25]

The CCSDS MO architecture was defined in a stratified manner (Figure 4) to allow mission operations services to be specified in an implementation and communication agnostic manner. The MO services are specified in compliance to a reference service model, using an abstract service description language, the MAL. This is similar to how Web Services are specified in terms of Web Services Description Language (WSDL). For each concrete deployment, the abstract service interface must be bound to the selected software implementation and communication technology. [26]

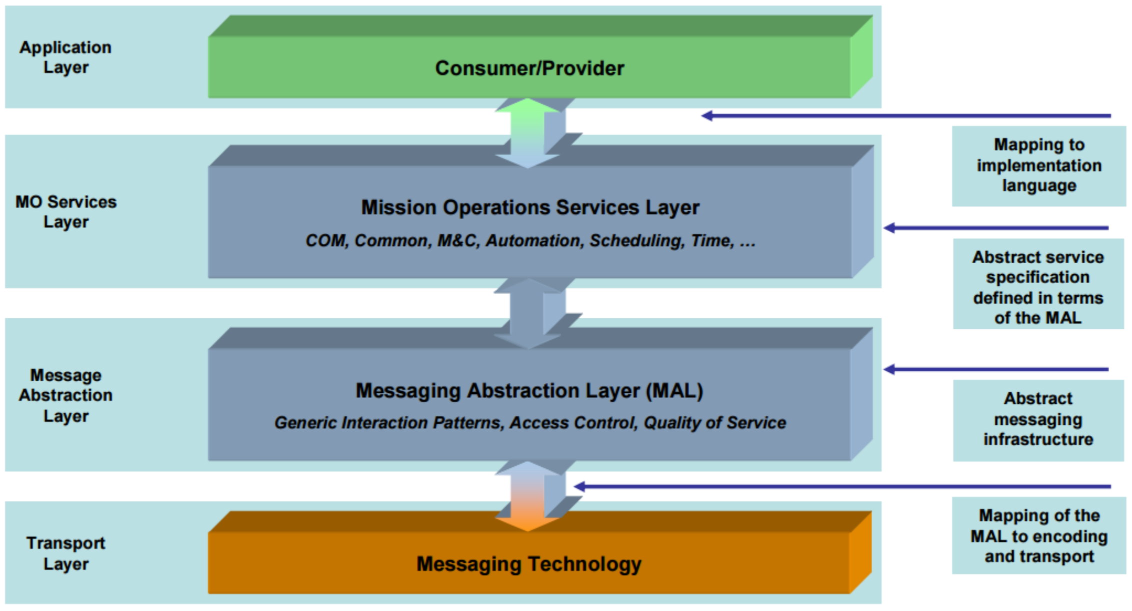

Application Layer
Consumer/Provider
Mapping to implementation language
MO Services Layer
Mission Operations Services Layer
COM, Common, M&C, Automation, Scheduling, Time, ...
Abstract service specification defined in terms of the MAL
Message Abstraction Layer
Messaging Abstraction Layer (MAL)
Generic Interaction Patterns, Access Control, Quality of Service
Abstract messaging infrastructure
Mapping of the MAL to encoding and transport
Transport Layer
Messaging Technology

Figure 4: CCSDS MO services Architecture

Standardization of a Mission Operations service framework offers a number of potential benefits for the development, deployment and maintenance of mission operations infrastructure: [27]

- Increased interoperability between agencies;
- Re-usage between missions;
- Reduced costs;
- Greater flexibility in deployment boundaries;
- Increased competition and vendor independence;
- Improved long-term maintainability.

The deployment of standardized interoperable interfaces between operating Agencies, the spacecraft and internally on-board would in itself bring a number of benefits. Each organization would be able to develop or integrate their own multi-mission systems that can then be rapidly made compliant with the spacecraft. It does not preclude the re-use of legacy spacecraft, simply requiring an adaptation layer on the ground to support it, rather than many mission-specific bespoke interfaces. In the on-board environment, where software development costs are considerably higher due to platform constraints and reliability requirements, software reuse can bring immense savings. [28]

The MO services layer is composed of sets of services that can be either CCSDS-standardized services (for example, COM, Common, M&C, Automation, etc) or bespoke services. The term “MO Core” is used throughout the document to represent collectively the MAL, COM and the Common services. [27]

1.5. Mobile Apps and Spacecraft Software Management

An application program is a computer program designated to perform a set of defined functions, tasks, or activities in a computer system. In recent times, the word "app" became the popular term to refer to an application program that runs on mobile devices.

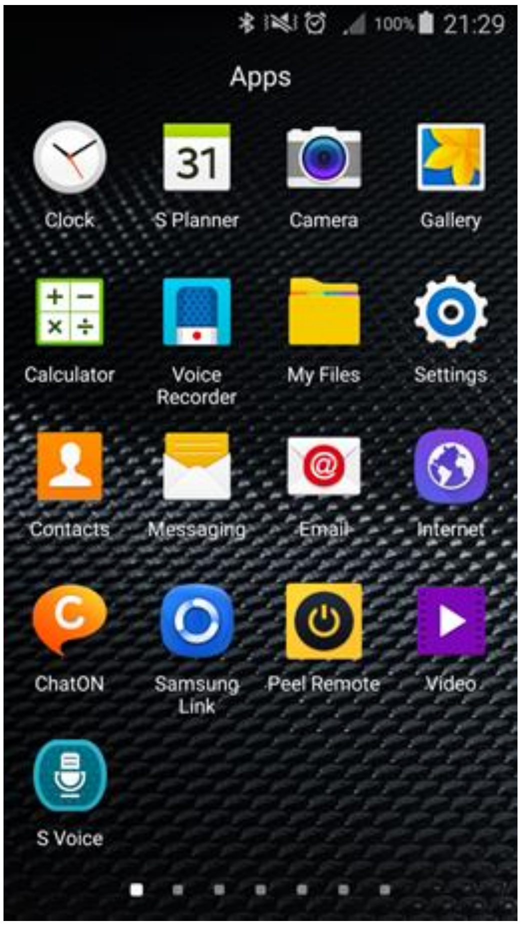


Figure 5: Screenshot captured showing apps

Andy Rubin was the co-founder of Android Inc, a company focused on developing a mobile operating system for smartphones which was acquired by Google in July 2005. Android gave the community an extra degree of freedom by making the mobile apps portable entities of software where the same app can run on the hardware of different vendors as long as the device has the same underlying Android framework. [29]

Mobile apps are usually installed, uninstalled or updated using an app marketplace such as "App Store" for iOS or "Google Play" for Android.

Mobile devices usually expose the icons of the installed apps in a launcher such as the one presented in Figure 5. This simplistic interface allows the user to browse through the list of apps and tap on the one wished to be started.

It is important to notice that app marketplaces do two different sequential tasks, first, they provide digital distribution by allowing the user to transfer the app to the phone and second, they provide package management system functionality by installing, updating and uninstalling the app according to the user desires. Apps are wrapped by file packages that contain additional metadata necessary to distribute and manage them. For example, when the user installs an app from Google Play, an Android Package Kit (APK) file is downloaded and then installed on the mobile device in a close to transparent way.

In Linux-based systems, the installation of software has been long done through package management systems which facilitate the process by fetching from a remote repository the desired package containing the pre-compiled binary code to be installed and additionally resolving dependencies on external libraries. [30]

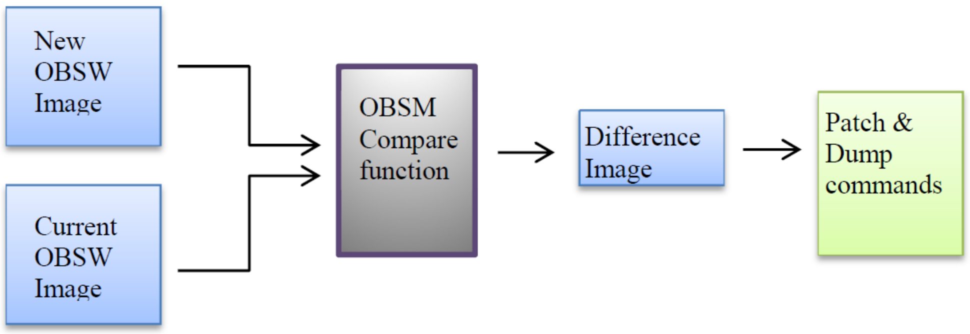


Figure 6: Software patches generation process using Galileo's OBSM tool. [31]

Despite the fact that package management systems are the preferred way to manage software in information technology, the space domain still does not use them and spacecraft software still relies on software patches that must be sent via telecommands. A concrete example of this situation is verified with the Galileo satellite constellation, a relatively recent mission which uses the "On Board Software Management" (OBSW) tool to compare the current on-board software image with the new one and generate the difference between them. Afterwards, a stack of commands addressing specific memory locations is produced in order to patch the on-board software. A diagram of the process can be visualized in Figure 6. [31]

1.6. Research Approach

As in any doctoral research, the outcome of the research was not completely defined from the beginning and it was open to the researcher within certain boundaries. A research proposal was outlined containing the area of research, motivation, objectives, a draft work programme, a high-level schedule, the description of the partners and their respective interests.

The research was started in October 2014 and was carried out until end of 2017. The initial objectives were to get familiar with the existing codebase, to start the first prototypes and then improve and learn from the experience obtained on each iteration. A concise set of ideas were defined, implemented and tested into what today is known as the NanoSat MO Framework.

Chapter 2 is the core of the research and presents the new concepts in order to produce a software framework for nanosatellites based on CCSDS MO services. Chapter 3 describes a concrete implementation in Java programming language for these new concepts as reference implementation. A Software Development Kit was put together to facilitate the development of applications based on the NanoSat MO Framework and it is described in chapter 4.

The NanoSat MO Framework is not tied to any particular mission therefore the mission-specific parts must be implemented separately. Chapter 5 explains the specific implementation for OPS-SAT and its operation in the overall system. In chapter 6, the validation of the framework for OPS-SAT is presented and some performance metrics are available.

The last chapter presents the outlook of the NanoSat MO Framework including many ideas on how to leverage it further.

A short activity report was produced every month giving a brief explanation of the goals achieved on that month, a selection of tasks to be completed for the upcoming month and finally, an overall view of the progress.

The research was focused on an end-to-end approach in software development which has been proposed before in one of the ESA studies. [131]

Chapter 2

2. NANOSAT MO FRAMEWORK

In this chapter, the NanoSat MO Framework (NMF) is introduced and described in detail. This chapter is the core of the research and all the upcoming chapters derive from the theoretical part presented here.

As the title of the dissertation indicates, the NanoSat MO Framework is a software framework for nanosatellites based on CCSDS Mission Operations services. It facilitates not only the monitoring and control of the nanosatellite software applications, but also the interaction with the nanosatellite platform. This is achieved by using the latest CCSDS standards for monitoring and control, and by exposing services for common peripherals among nanosatellite platforms. Furthermore, it is capable of managing the software on-board by exposing a set of services for software management.

The NanoSat MO Framework is built upon the MO Architecture and thus inherits its benefits, for example, the NMF is not bound to a single programming language and it is not limited to a specific domain. Additionally, it is independent from any specific nanosatellite platform.

The framework sets of services are assembled together into the NMF Generic Model that provides a generic model for extension by the NMF Composites. The components extending the NMF Generic Model are named NMF Composites and they are specialized for a specific segment and particular purpose.

New dedicated concepts are defined for the NMF such as: NMF App, NMF Package, NMF Mission, NMF Library and NMF SDK. These are important terms that are used in the next chapters.

The NMF Composites must be connected to each other in order to create a complete end-to-end scenario. Although each NMF Mission is responsible for defining its own scenario, some generic scenarios are presented and they can be combined to form more complex ones.

2.1. Principles

This section presents the core functionality, the problem domains, and how portability is achieved on-board.

2.1.1. Core Functionality

The NanoSat MO Framework is a software framework with the main objective of facilitating the development of software for nanosatellites including both the ground and space segments, and also to simplify the orchestration of software with other tools. The core functionalities of the framework are:

- Monitoring and control of on-board status and activities
- Monitoring and control of the platform peripherals
- On-board software management

The framework can monitor on-board statuses such as alerts, parameters and aggregations of parameters, and also control on-board activities by supporting execution of actions and by allowing parameters to be set on-board. The on-board software can spontaneously generate parameters and alerts nevertheless periodic reporting is possible for parameters and aggregations. Additionally, individual parameters and/or aggregations can also be retrieved on request.

Nanosatellites usually include a common set of peripherals that are also available on other nanosatellite platforms. It is possible to examine these peripherals and find what are the common functionalities provided by them, and then generate a generalized interface that allows the monitoring and control to most of them. For example, any GPS unit allows the possibility to retrieve the current position and/or time information. The framework provides this meaningful information in an abstract way regardless of the GPS unit's vendor.

The framework supports the management of on-board software in a spacecraft from a remote entity. This includes the traditional memory patches and new mechanisms involving management of packages and launching applications. For example, software packages with applications can be installed, uninstalled, and updated. These applications can then be initialized and terminated at request on a later instance of time. The focus is however on the new techniques.

Other important aspects of the research were to:

- Follow and contribute to the latest CCSDS standards on MO Services

- Investigate relevant and innovative software technologies in other fields, for example smartphone technologies
- Use the ESA's OPS-SAT mission as proof of concept

The Spacecraft Monitor and Control Working Group from the CCSDS defined a service-oriented architecture that allows specifying services for mission operations. This architecture provides a solid support for the design of a framework dedicated to mission operations of nanosatellites. [27]

Smartphones became very popular in the beginning of the 21st century by quickly replacing the old phones that were restricted to a few set of tasks such as calling and texting. The smartphones brought a completely new set of possibilities as it allows the user to dynamically install "apps" that provide dedicated data and can accomplish specific tasks pertinent only to a smaller set of people. In 2017, the number of apps available in leading app stores is approximately 2.8 million for Google Play and 2.2 million for Apple's Apps Store. The technology behind this success is well established and can provide good inspiration for the improvement of today's space software. [32]

Lastly, ESA's OPS-SAT mission will include a powerful processing platform that provides the perfect environment for the validation of the framework in space. Although the first flight is expected to take place in OPS-SAT, the framework shall not be restricted to a single mission and it is generic enough to be used in future missions. The core functionality is intended to be validated in space and at the same time to provide a reliable solution to facilitate the development of software for other OPS-SAT experimenters. [19]

With that in mind, it is important to distinguish between what is generic for all missions and what is dedicated to OPS-SAT. The dissertation clearly separates both into different chapters. The reference implementation presented in chapter 3 is generic and can be used by other missions. By contrast, chapter 5 is specific to the NanoSat MO Framework's implementation for the OPS-SAT mission.

A Proof of Concept prototype with two CCSDS MO services, the Aggregation and Parameter service, was previously developed at ESA in 2014, which could be partially reused. Thus, some expertise already existed internally in ESOC premises about CCSDS MO. [22]

By having a clear set of core functionality defined as final goals, it becomes easier to start from the prototype mentioned above and extend it into a full-blown software framework for nanosatellites which includes an open-source reference implementation available to the general public.

2.1.2. Problem domains

A space mission after being developed, launched and commissioned successfully is ready to be operated and to achieve its objectives. The system of a typical single spacecraft mission for commercial uses is usually composed of 3 parts, the Space Segment, the Ground Segment and the User Segment just like Figure 7 illustrates. Other system configurations can be possible however, they would have some resemblance to the one presented in figure below.

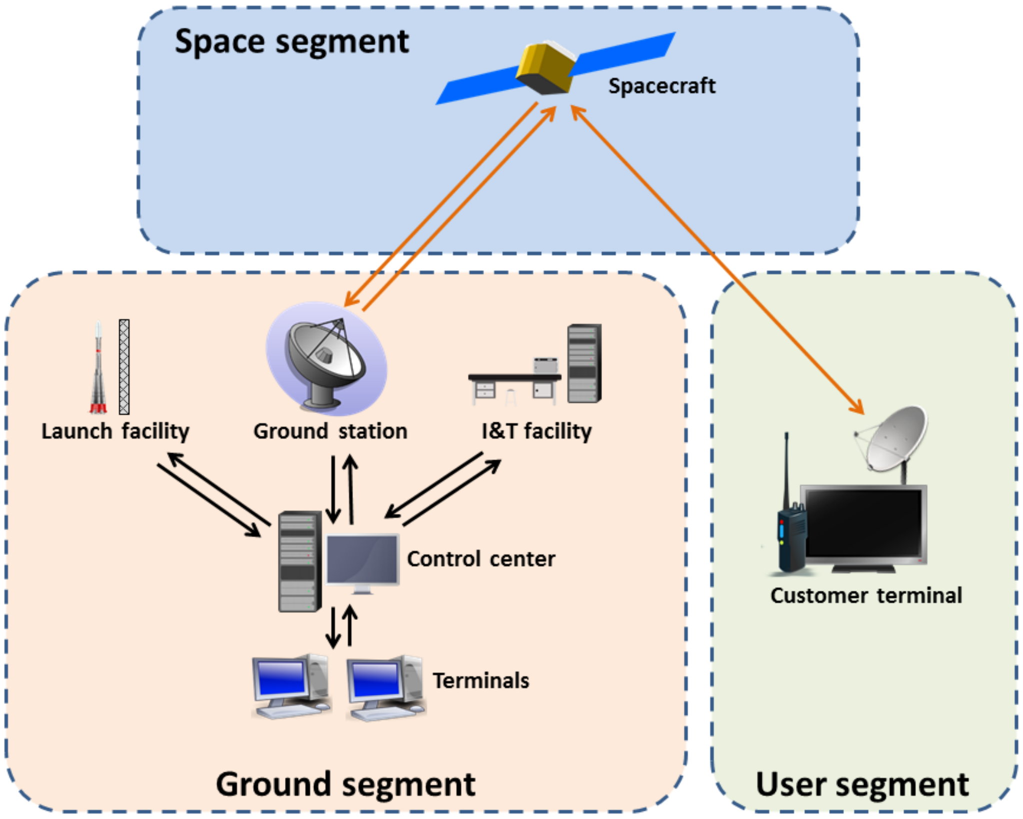


Figure 7: Example of a spacecraft system [33]

The set of activities necessary to operate a spacecraft and its payload is commonly known as mission operations. These activities include monitoring and control of the spacecraft, mission planning and scheduling, automation, flight dynamics, management of on-board software, distribution of mission data products. They are typically regarded as the functions of the Mission Control Centre (MCC) and are performed by a team of people. The different activities of mission operations do not necessarily need to be all in the same centre as collaborating agencies and ground-segment sites can distribute them across different locations. [27]

If different facilities are operated by separate agencies, it is necessary to use interoperable interfaces in the interactions that cross distribution boundaries. Figure 8 contains an example where the connecting lines show end-to-end interactions between activities. [27]

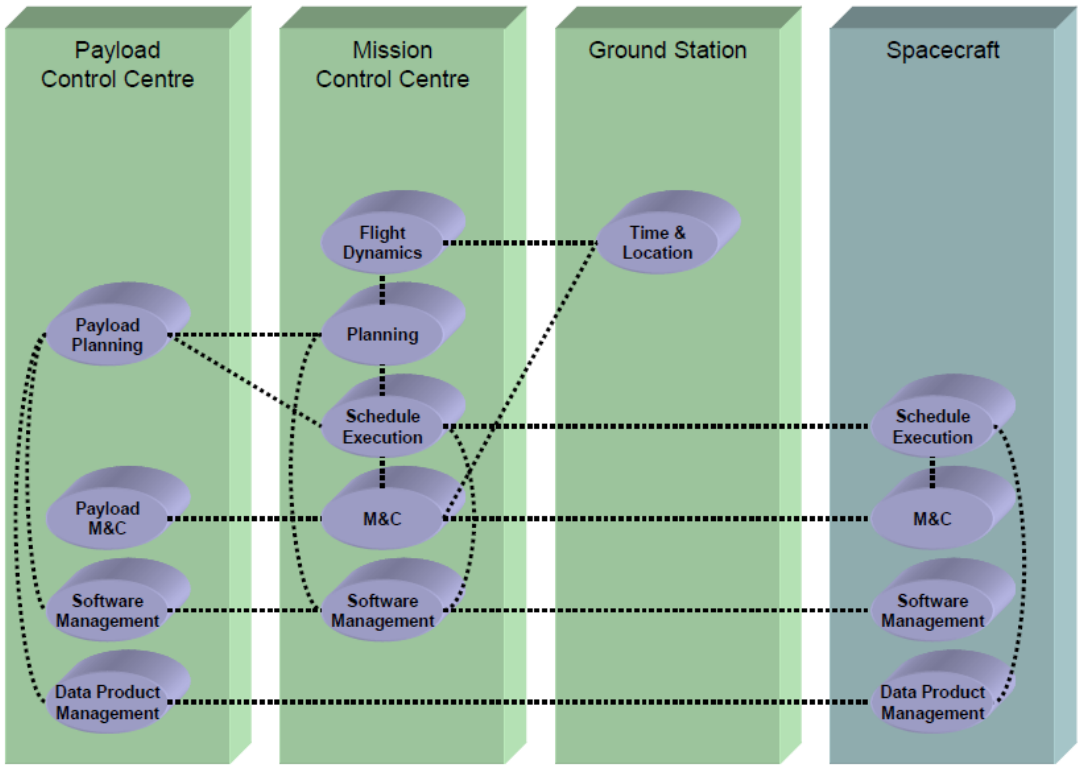


Figure 8: Mission operations activities distributed between a Spacecraft and three separate Ground Segment facilities. [27]

The CCSDS Mission Operations Service Framework is concerned with end-to-end interactions between mission operations application software, wherever it may reside within the space system. The underlying transport services over which Mission Operations services are carried can be different depending on the nature of the communications path. [27]

By specifying standardized services abstracted from the transport, rapid construction and deployment of new systems and configurations can be achieved. One of the expected deployments of the MO components is on-board the spacecraft where there will exist the need to communicate not only with ground based components, but also between on-board components. [28]

From the software development point of view, the developers can be completely focused on the mission operations while neglecting the underlying layers and also, by having a uniform way of representing the interfaces, it is possible to auto-generate documentation and source code in different programming languages, thus significantly reducing human errors. These are some of the many advantages of using MO services.

The NanoSat MO Framework is built upon the CCSDS Mission Operations Service Framework and thus inherits the same agnosticism towards the transport layers used in the space system. This allows the conceptualization of a "multi-segment" software framework dedicated to nanosatellites that is neither limited to the space segment nor the ground segment, and instead it covers both segments simultaneously.

From a spacecraft system point of view, the NanoSat MO Framework is split in two segments. First, the "Ground Segment" just like in any traditional spacecraft system. Second, the "NanoSat Segment" which is the equivalent of the space segment but because the target deployment in space are nanosatellites, it contains a more specialized name.

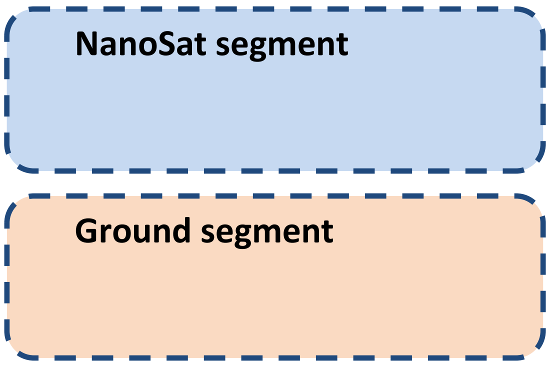


Figure 9: Spacecraft system of the NMF.

Once again, the underlying transport of how the messages are exchanged on the layers below the MAL transport binding is not relevant. For example, Space Link Extension (SLE) is a very popular mechanism of exchanging data within the Ground segment but this is transparent to the MO services layer. [34]

In the NanoSat segment, there are two envisaged ways of deploying software:

- Monolithic
- Multiple applications

The monolithic deployment is the traditional way of deploying on-board software at present however, it is possible to split functionality into multiple applications and start/stop them as needed. The latter takes inspiration from the popular concept of "apps" existent in today's mobile devices. It includes a supervisor application to start/stop applications, to provide a mechanism for finding and connecting to the NMF Apps, and exposes high-level services to monitor and control the nanosatellite platform.

No specific Operating System is needed for defining the NanoSat MO Framework, as this is a concrete implementation problem. As an example, chapter 3 presents the Java implementation of the NMF which the minimum system requirement is to have a Java Runtime Environment capable of running Java Virtual Machines.

In a traditional European mission, the separation between the stakeholders is commonly done in the Ground-Space link by defining an interface that is usually a tailored version of the ECSS Packet Utilization Standard (PUS). This creates a clear separation between the ground software development and the space software development. The latter is usually developed in a monolithic way covering all the functionalities of the spacecraft. [24]

Comparatively, when a mobile app is developed, it is never intended to cover all the possible functionalities of a normal phone but instead they are software entities dedicated to executing particular tasks in order to reach the user goals. Extending this idea to nanosatellites can potentially generate a new set of space software developers focused on particular satellite operations and functionalities, just as the Uber™ app and Snapchat™ app have very different purposes to the user.

Following this line of thinking, one can split the problem domain in three parts: NMF Ground Development, NMF Mission Development, and NMF App Development.

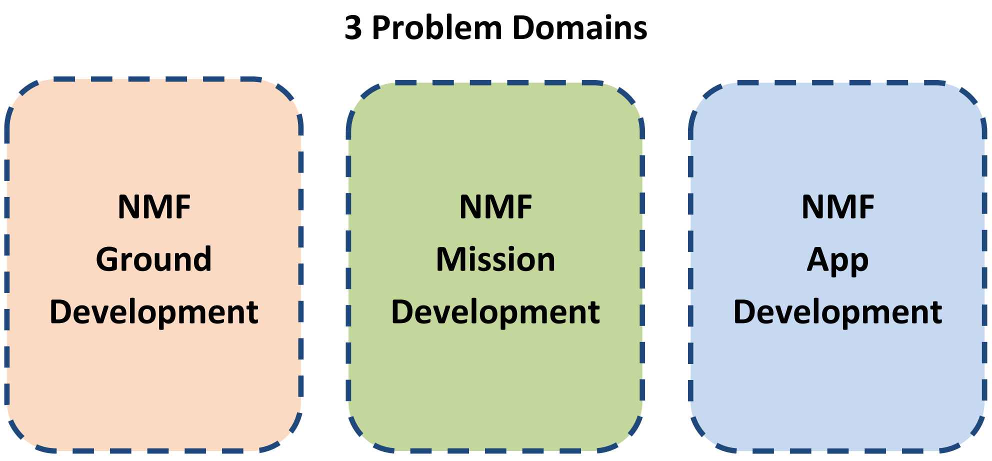


Figure 10: The 3 problem domains of the NMF.

In the NMF, applications on-board share a common library, which contains high-level components that let the integration of the software to be done in a seamless way by extending them with the application logic or with mission-specific logic. These components are described in section 2.6, specifically, the NanoSat MO Connector and the NanoSat MO Supervisor. On Ground, the same idea was applied and a high-level component was derived for ground that allows access to any application that uses the previous components. The stubs and skeletons of these components are automatically generated from the services definition file and so, they always perfectly match each other. The main advantage is that developers does not need to worry about the specific service interfaces and can shift their focus to the functional part of the software.

By using well-defined services and by selecting the transport binding between software entities, it is possible to decouple all the three problem domains.

This means that a ground developer working in the NMF Ground Development problem domain can, for example, develop a Mission Control System (MCS) that is capable of connecting to any on-board NMF App independently of the mission where it is running.

A mission developer working on the NMF Mission Development problem domain does not need to know the specific details of the other two problem domains as the focus is on developing the missing mission-specific bits. For example, the transport binding between ground and space in case the mission uses a dedicated transport and/or developing the adapters for the Platform services in order to be able to monitor and control the peripherals on-board. In this sense, the NMF Mission Development domain belongs to both Ground and NanoSat segments.

The development of applications becomes then abstracted from the underlying layers because high-level components are used that remain abstract enough from any mission in particular but specific enough to develop software against it and still get a functional application that can be executed and tested.

2.1.3. On-board software portability

Portability is the ability of using the same software in different environments. The pre-requirement for portability is the generalized abstraction between the application logic and system interfaces. [35]

Software portability has been very popular on mobile devices where the two main players, Android and iOS, expose interfaces that abstract from the low-level details of the device in order to facilitate the development of apps for different phones. Figure 11 shows the Android Framework stack and its different abstraction layers that enable portability. [36]

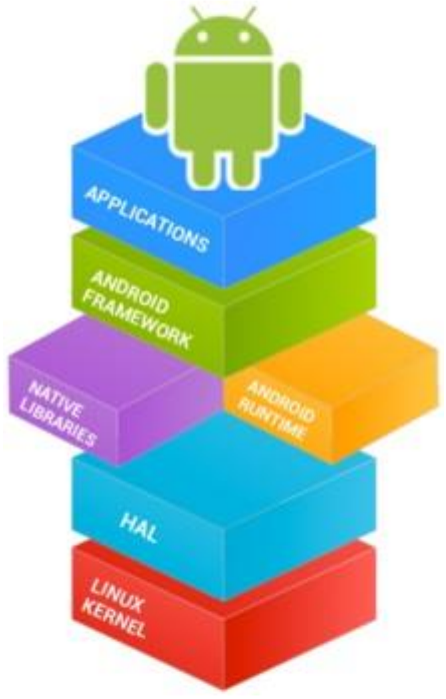


Figure 11: Android Framework stack.

The idea of creating portable on-board software has been introduced in the literature for the first time by A. Koltashev in "A Practical Approach to Software Portability" back in 2003 and their approach relied on architectural stratification and interface standardization both for the on-board software and the development environment. [37]

Core Flight System (cFS) is an onboard software framework developed by NASA that also uses a layered architecture and compile-time configuration parameters, which make it portable and scalable for different platforms. Their goal is to sustain an open-source application ecosystem. Although cFS follows the concepts of stratification, portability and code reuse, it is developed with an emphasis on embedded software systems while the NanoSat MO Framework is aiming at taking full advantage of systems capable of running on complete/ground-used Operating Systems and that are not restrained by ancient hardware technology. [38]

The Space Avionics Open Interface Architecture (SAVOIR) is a European initiative to federate the space avionics community in order to improve the way that space avionics sub-systems are built. The SAVOIR On-board Software Reference Architecture (OSRA) is a reference architecture for software on-board spacecraft platforms that uses a component-based approach executed by an execution platform adapter to space. The standardization process of the interfaces is the focus of SAVOIR OSRA. [39] [40]

Neither cFS nor SAVOIR OSRA is completely focused on nanosatellites and instead they are intended to be used by spacecraft in general. Additionally, they are not based on the CCSDS Mission Operations standards at the moment of writing.

There are nanosatellite missions that have used Android in space such as STRaND-1 and NASA AMES's PhoneSat. The former used "STRAND Application Framework" to allow Android application to run on the phone and downloading their data on ground. This is accomplished by using a "STRaND Application Wrapper" that transforms a generic Android App into a 'STRaND' App. Thus the developed apps are tied to STRaND-1 platform and are not portable. Additionally, the apps would necessarily expect to have an Android system underneath. [41]

The NanoSat MO Framework consists of a stratified architecture where the application logic is exposed to an abstraction interface that allows:

- Generic monitoring and control functionality
- Nanosatellite's platform monitoring and control

For generic monitoring and control, the application logic can register on the provider side, sets of Parameters, Aggregations, Actions and Alerts, and then monitor and control them from a consumer side. The NMF takes care of storing and managing their respective definitions in the CCSDS M&C services and ensures that it makes the right call back when necessary, either periodically or by a request from a consumer. This abstracts the application logic from the layers below including the transport mechanism to reach the consumers that can be a ground application, an on-board device, or another process running in the same device. Figure 12 presents a visualization of the different layers involved for generic monitoring and control of an application's logic, with a consumer stack on the left side and a provider stack on the right side.

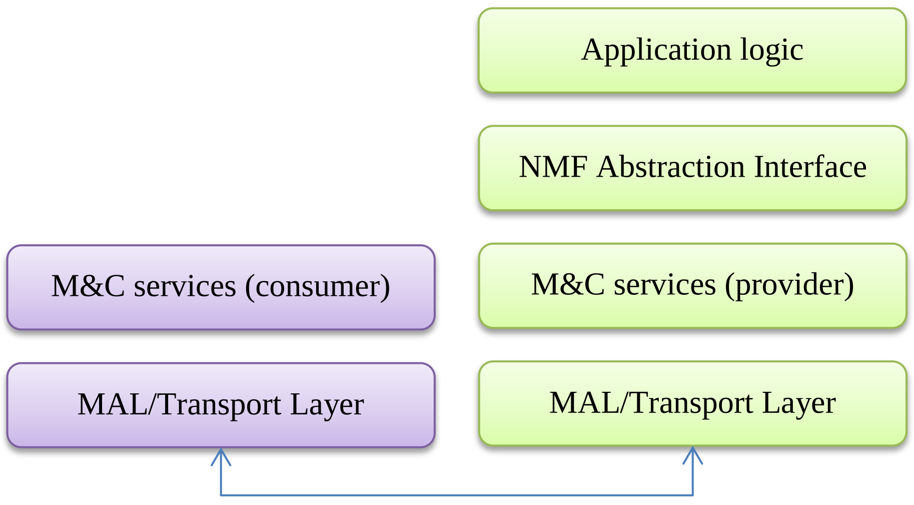


Figure 12: Consumer and Provider side for generic monitoring and control of an application's logic.

The CCSDS M&C services standardize the interface between the consumer and provider, and do not prescribe details for concrete implementation of the service and its respective orchestration with the layers above if any. When implementing the NMF, an

NMF Abstraction Layer is expected to be defined in order to always expose the same interface towards the application logic.

For the nanosatellite's platform monitoring and control, the application logic is exposed to a set of services well-defined in terms of MO known as "Platform services". These services have been specified in a way that allows the monitoring and control of common nanosatellite's platform devices, such as, GPS, ADCS, Camera and others. It is important to notice that on the first case, the provider of the data was the application's logic but now the application's logic is the consumer of the Platform services. Section 2.4 explains these services in further detail.

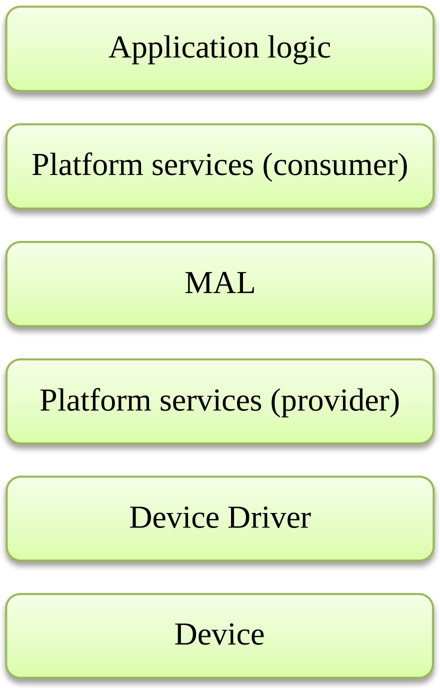


Figure 13: Monitoring and control of a nanosatellite platform's device using the Platform services with a single process.

The stack in Figure 13 includes both a consumer and provider of Platform services. When running in the same process, the stack introduces an overhead because between the consumer and provider, the MAL implementation has to handle the exchange of messages however this induced overhead is expected to be small, as the MAL only has to do internal routing without actually having to encode the message.

The stack allows the consumer and provider sides to be deployed in different processes and split the stack in two independent parts as depicted in Figure 14. The main advantage of doing that is that multiple consumers can connect to the Platform provider side and so, they can share the nanosatellite's resources.

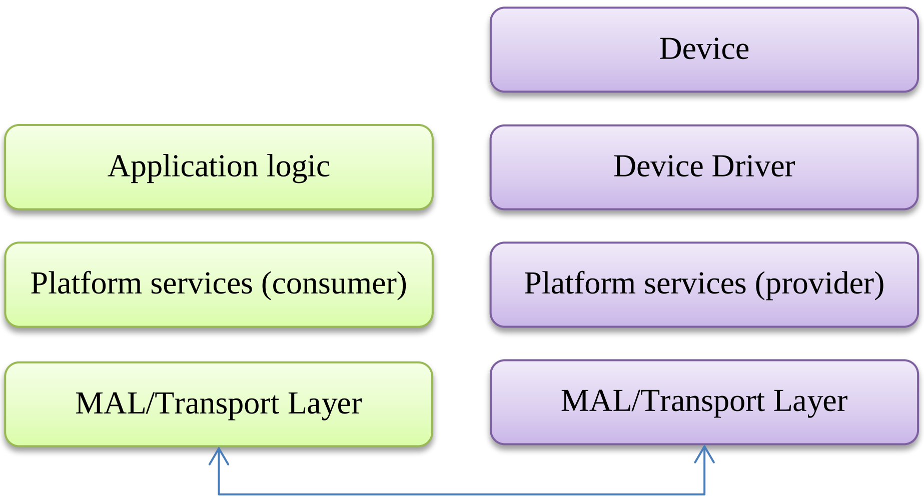


Figure 14: Monitoring and control of a nanosatellite platform's device using the Platform services with two processes.

These two ways of deploying the Platform service are important during the design time of the mission. The NMF includes high-level components for both ways where for the first deployment, the NanoSat MO Monolithic component is used while for the multiple processes, the NanoSat MO Supervisor takes the responsibility of deploying the Platform services provider side and the NanoSat MO Connector to deploy the consumer side. Section 2.6 explains these high-level components in further detail.

The second way inherits from MO the advantage of being transport-agnostic and, the consumer and provider sides can be developed in different programming languages just like the example on Figure 15.

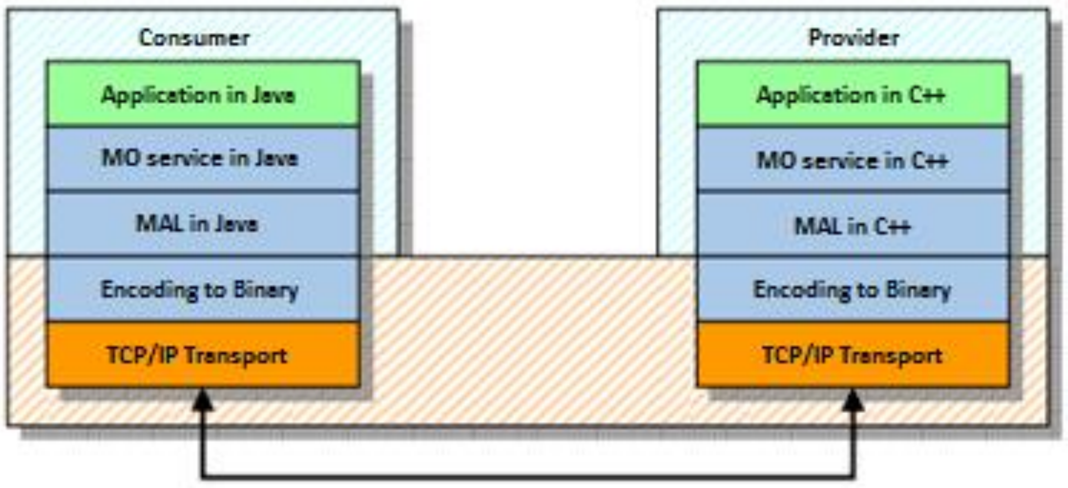


Figure 15: A consumer and a provider developed in different programming languages.

The consumer of the services is not limited to applications running on-board as these can also be consumed from different deployment locations, such as, an application running on ground or an on-board device on the same bus. The most appropriate transport layer binding should be selected depending on the location of the consumer/provider for example, between the Ground segment and the NanoSat segment the most likely candidates are SPP or TCP/IP, but for Inter-Process Communication (IPC), one can use TCP/IP, ZeroMQ or RMI.

The same underlying transport layer must be present both on the consumer and provider sides in order to exchange data between both entities.

For a concrete example, please check the selected transport bindings for OPS-SAT mission on chapter 5.

2.2. CCSDS MO Heritage

This section describes the CCSDS MO Heritage that is present in the NanoSat MO Framework and it was split in 3 sub-sections: MO Core, Transport Binding, and Monitor and Control services.

As described in section 2.5, the NanoSat MO Framework takes advantage of the MO architecture and uses 3 sets of services that have already been standardized by the SM&C Working Group of the CCSDS organization: the COM, Common, and M&C services. The first 2 sets are described in sub-section 2.2.1 while the M&C services are described in sub-section 2.2.3. Figure 16 presents an exploded view of the MO architecture including its sub-layers. [27] [42] [43] [44]

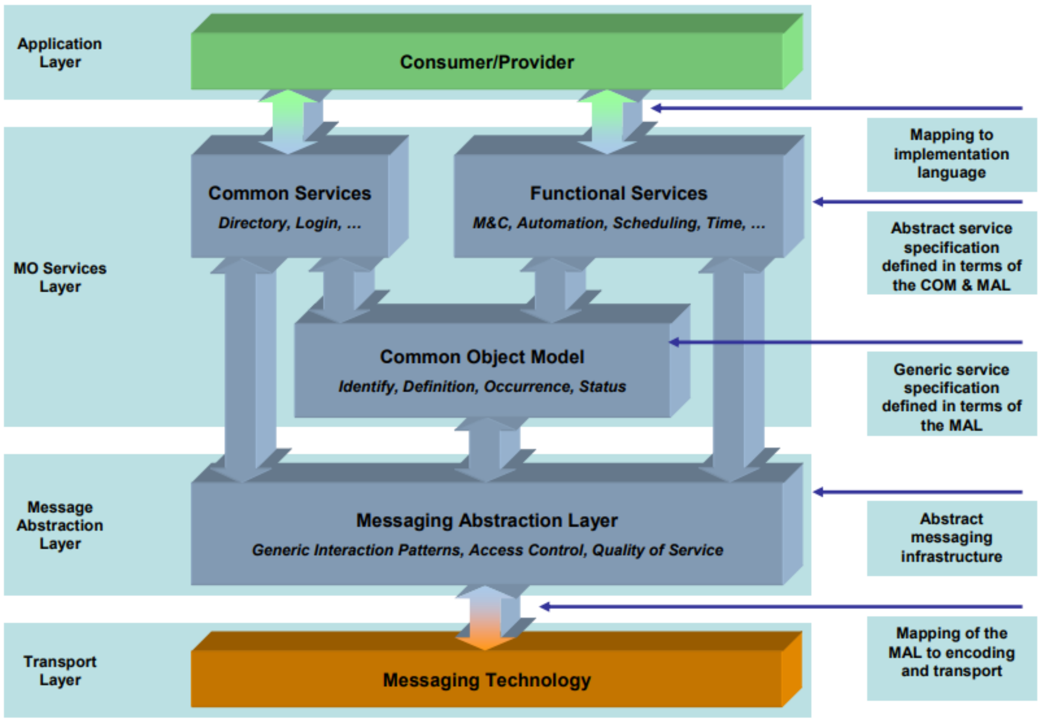
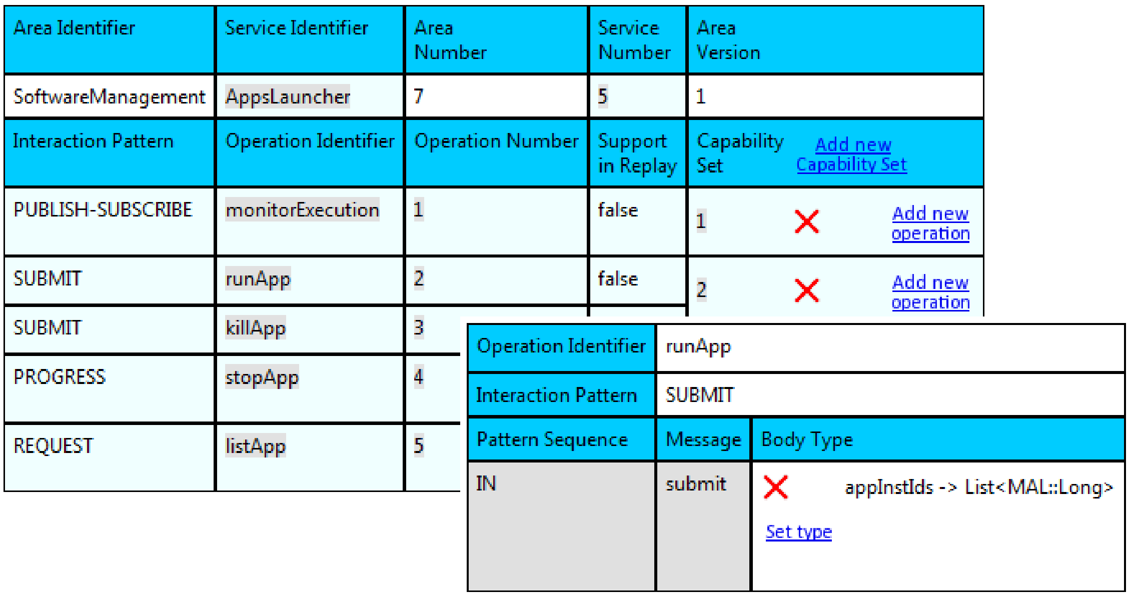

Application Layer
Consumer/Provider
MO Services Layer
Common Services
Directory, Login, ...
Functional Services
M&C, Automation, Scheduling, Time, ...
Common Object Model
Identify, Definition, Occurrence, Status
Message Abstraction Layer
Messaging Abstraction Layer
Generic Interaction Patterns, Access Control, Quality of Service
Transport Layer
Messaging Technology
Mapping to implementation language
Abstract service specification defined in terms of the COM & MAL
Generic service specification defined in terms of the MAL
Abstract messaging infrastructure
Mapping of the MAL to encoding and transport

Figure 16: Exploded view of the MO architecture. [27]

Section 2.2.1 includes a high-level description of the MAL, COM services and Common services. Although they are presented together, they are not at the same level in the MO architecture stack as observed in Figure 16.

Since the MO architecture is used, all the services are specified in terms of the MAL, which provides an abstract service specification and MAL Data Types. Each service contains a set of operations that must specify the interaction pattern in use between the consumer and provider, and the data exchanged in terms of MAL Data Types. Figure 17 includes an example of a concrete service and a concrete operation defined in terms of the MAL. [130]



Area Identifier	Service Identifier	Area Number	Service Number	Area Version		
SoftwareManagement	AppsLauncher	7	5	1		
Interaction Pattern	Operation Identifier	Operation Number	Support in Replay	Capability Set	Add new Capability Set	
PUBLISH-SUBSCRIBE	monitorExecution	1	false	1	✗	Add new operation
SUBMIT	runApp	2	false	2	✗	Add new operation
SUBMIT	killApp	3				
PROGRESS	stopApp	4				
REQUEST	listApp	5				

Operation Identifier	runApp	
Interaction Pattern	SUBMIT	
Pattern Sequence	Message	Body Type
IN	submit	✗ appInstIds -> List<MAL::Long> Set type

Figure 17: MO Graphical Editor tool displaying the runApp operation of the Apps Launcher service. [45]

The Software Management services and Platform services are not part of this section. They have their own dedicated sections because they are bespoke interfaces defined for the NanoSat MO Framework. Although these have not been defined by the CCSDS, they are both very good candidates to become future CCSDS standards.

2.2.1. MO Core

The MO Core is the collection of the elements, MAL, COM services and Common services. These elements were standardized by the CCSDS. The MO Core is not part of the MO terminology but it was defined for the NanoSat MO Framework for simplification reasons.

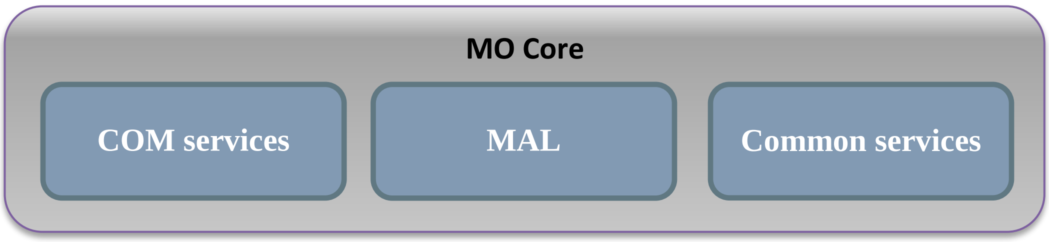
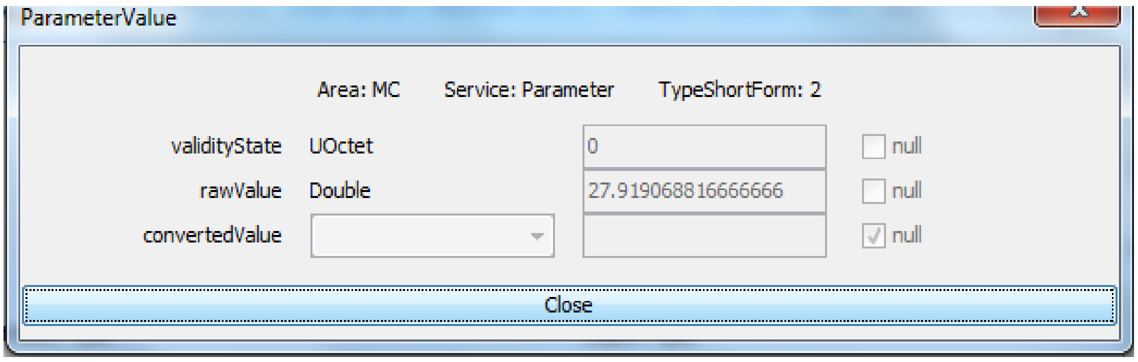


Figure 18: The MO Core with COM services, MAL, and Common service.

Although the individual elements of the MO Core are presented side-by-side in Figure 18, they are not at the same level in the MO architecture stack as presented in Figure 16.

The Message Abstraction Layer (MAL) abstracts the services layer from the transport layer by defining a set of MAL Interaction Patterns and a set of MAL Data Types. The MAL Interaction Patterns are SEND, SUBMIT, REQUEST, INVOKE PROGRESS, PUBLISH-SUBSCRIBE, and they allow a service operation to specify the interaction behavior between the consumer and the provider.

The MAL Data Types allow the creation of Composite types that contain inside MAL Attributes and/or other Composite types. MO services can use these structures to exchange information. At runtime, one can start instances of the structures defined in terms of MAL Data Types. For example, the ParameterValue structure is defined by the Parameter service and it contains 3 fields: validityState, rawValue, and convertedValue. An example of a concrete instance of a ParameterValue composite structure is presented in Figure 19.


ParameterValue

Area: MC Service: Parameter TypeShortForm: 2

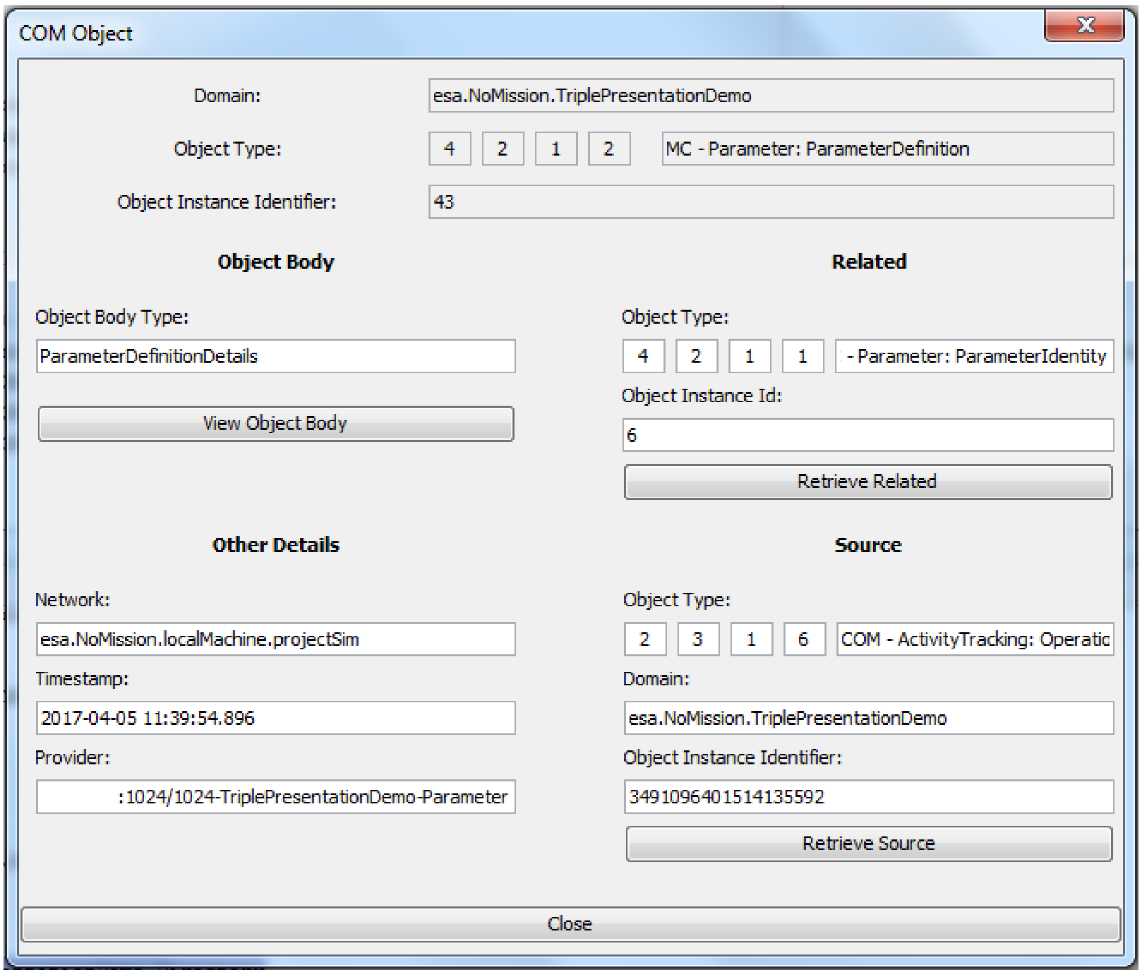
validityState	UOctet	0	null
rawValue	Double	27.919068816666666	null
convertedValue			[✓ null]

Figure 19: An instance of the ParameterValue data structure displayed on the Consumer Test Tool.

The COM provides a standard data object model for MO Services to utilize. Whereas the MAL provides the building blocks that can be used to define the operations of a MO service, the COM provides the building blocks for the specification of the data objects of a service. This builds upon the MAL to define a standard data model for an MO service. Each service that utilizes the COM must define the object, or set of objects, that form the data model of the service. [42]

A COM object is uniquely identified by the domain of the object, together with the type of the object, and with the object instance identifier. A COM object can link to 2 other objects via a related field and a source field. It is service specific how these links are to be used. [42]

An instance of a ParameterDefinition COM Object can be seen in Figure 20 which includes a related link to the ParameterIdentity COM Object and a source link to the OperationActivity COM Object that generated the creation of the object. The instance includes an object body of type ParameterDefinitionDetails.


COM Object

Domain: esa.NoMission.TriplePresentationDemo

Object Type: 4 2 1 2 MC - Parameter: ParameterDefinition

Object Instance Identifier: 43

Object Body

Object Body Type: ParameterDefinitionDetails

Related

Object Type: 4 2 1 1 - Parameter: ParameterIdentity

Object Instance Id: 6

Other Details

Network: esa.NoMission.localMachine.projectSim

Timestamp: 2017-04-05 11:39:54.896

Provider: :1024/1024-TriplePresentationDemo-Parameter

Source

Object Type: 2 3 1 6 COM - ActivityTracking: Operatic

Domain: esa.NoMission.TriplePresentationDemo

Object Instance Identifier: 3491096401514135592

Figure 20: An instance of a ParameterDefinition COM Object displayed on the Consumer Test Tool.

The COM specification provides 3 support services that build upon the basic object model, these services shall be used by other services: [42]

- Event service: A generic service to publish COM events
- Archive service: Provides a generic way to archive COM objects following the CRUD principles
- Activity Tracking service: Provides the ability to track the progress of activities

The Common services' set is composed of three services: Directory service, Configuration service, and Login service.

The Directory service offers service discoverability, the means for implementing dynamic bindings by providing publish and lookup facilities to providers and consumers, and retrieval of the service specifications. It allows providers to publish their location in the form of a URI (Universal Resource Indicator) so that consumers can locate them without having to know in advance the location. An additional functionality of the Directory service is the capability of providing the specification of the available services in the provider. Any consumer requesting this information receives an XML file containing all the services' operations, Interaction Patterns and Data Types, thus allowing automatic configuration of the consumer to match the service interface offered by the provider. [43] [46]

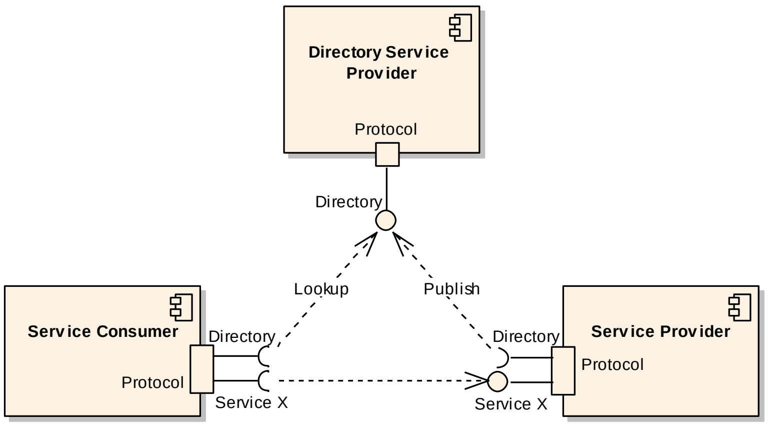


Figure 21: Directory service concept.

The Configuration service provides the ability to configure a service and/or a provider of services. It provides facilities for the management of configurations held by a provider if applicable to that implementation. The COM Objects defined by the Configuration service can be persisted in the COM Archive and reloaded on a later instance of time. Restoring a provider's state upon start up is an envisioned use case. [44]

The Login service allows an operator to provide authentication information to the system. It takes the operator's credentials and uses a deployment-specific mechanism to authenticate the operator. [44]

2.2.2. Transport Binding and Encoding

The transport binding binds the MAL to a specific transport technology. Additionally, the encoding determines how the MAL data types are encoded into concrete data.

At the time of writing, the following transports are CCSDS standards or expected to become CCSDS standards in the near future:

- Space Packet Transport Binding [47]
- TCP/IP Transport Binding [48]
- HTTP Transport Binding
- ZeroMQ Transport Binding

At the time of writing, the following encodings are CCSDS standards or expected to become CCSDS standards in the near future:

- Binary Encoding [47]
- Split Binary Binding [48]
- XML Encoding

A transport binding to Java RMI developed in Java programming language is available online and it is used for testing applications however it is not part of any official CCSDS standard. [49]

The Space Packet Protocol Transport Binding does not specify the subnetwork to be used, leaving it open as an implementation-specific detail. The TCP/IP Transport Binding is more appropriate to be used on terrestrial networks and inter-process communication (IPC) however it may be also used for the space link. The HTTP Transport Binding can be used on terrestrial networks and because it uses a well-known protocol, it can be used to bypass firewalls and facilitate the deployment. The ZeroMQ Transport Binding can be used for terrestrial and IPC.

It is also possible to use non-standardized Transports Bindings or tailored versions, however this is not recommended because it breaks the interoperability goal from the CCSDS. For example, in OPS-SAT the Binary Encoding had to be tailored because SCOS is not able to handle the decoding of the standardized Binary encoding. This is further explained in chapter 5.

2.2.3. Monitor and Control services

The Monitor and Control services provide generic monitoring and control functionality of a remote entity and they are composed of 6 main services and 2 auxiliary services. The 6 main services are: Parameter service, Aggregation service, Alert service, Action service, Check service, and Statistic service. The 2 auxiliary services are: Conversion service and Group service. [43]

The main services are: [43]

- The Parameter service allows a consumer to acquire parameter values from a remote entity, to manage the parameters and to enable/disable their generation.
- The Aggregation service allows a consumer to acquire sets of parameters values from a remote entity, to define aggregations with parameter, to manage the aggregations, to enable/disable their generation, and to periodically report on them.
- The Alert service allows a provider to publish alerts to consumers. A consumer can manage the alert definitions including enabling and disabling them.
- The Action service allows a consumer to invoke actions on a remote entity. These actions can be tracked through the different stages starting in release of the action until the completion of the action.
- The Check service allows a consumer to define checks for certain parameters and get reports in case the check is triggered.
- The Statistic service allows a consumer to acquire statistics for certain parameters and to manage the statistic definitions.

The auxiliary services are: [43]

- The Conversion service provides conversion mechanisms to be used by the main services.
- The Group service allows a consumer to group sets of COM Objects to facilitate the management of the other services. For instance, instead of enabling a big set of aggregations, one can directly enable a group containing those aggregations.

For more information on the specific service interfaces, please check the reference to the CCSDS M&C standard book. [43]

2.3. Software Management services

The Software Management set of services contains services for on-board software management of a spacecraft. The following services are contained in the Software Management set:

- The Memory Management service is a low-level service that allows a consumer to load, dump and check memory addresses.
- The Software Image service allows a consumer to manage Software Images and Software Patches by allowing the generation of new Software Images from a patch.
- The Package Management service allows a consumer to manage software packages on modern operating systems using the concept of packages.
- The Apps Launcher service allows a consumer to launch applications.
- The Heartbeat service publishes a periodic beat to the listening consumers.

This set of services were defined during the research and are not part of the CCSDS standardized services at the time of writing however they are good candidates for future standardization.

The Memory Management service provides the capability for loading an on-board's memory device; the capability for dumping data from an on-board memory device and aborting it; the capability for checking an on-board's memory device.

Area Identifier	Service Identifier	Area Number	Service Number	Area Version
SoftwareManagement	MemoryManagement	7	1	1
Interaction Pattern	Operation Identifier	Operation Number	Support in Replay	Capability Set
SUBMIT	loadMemory	1	No	1
PROGRESS	dumpMemory	2	No	2
SUBMIT	abortMemoryDump	3	No	
INVOKE	checkMemory	4	No	3

Table 1: Memory Management service.

The Software Image service provides the ability to deploy, list, generate, delete, clone and check the integrity of Software Images. This service can be used to manage Software Images on-board of a spacecraft or to manage Software Images in virtual machines that might run on-board of a spacecraft. The service supports the generation of a new Image from a previous baseline Image using an additional delta. This allows patching software Images without the need to transfer the complete new Image.

Area Identifier	Service Identifier	Area Number	Service Number	Area Version
SoftwareManagement	SoftwareImage	7	2	1
Interaction Pattern	**Operation Identifier**	**Operation Number**	**Support in Replay**	**Capability Set**
SUBMIT	deployImage	1	No	1
INVOKE	checkImageIntegrity	2	No	2
INVOKE	cloneImage	3	No	3
REQUEST	listImage	4	Yes	4
INVOKE	patchImage	5	No	5
REQUEST	addImage	6	No	
SUBMIT	deleteImage	7	No	
REQUEST	listPatch	8	Yes	6
REQUEST	addPatch	9	No	7
SUBMIT	deletePatch	10	No	

Table 2: Software Image service.

The Package Management service provides the ability to install, uninstall, upgrade, list, check packages. The service shall optionally support verification of packages, verification of digital signatures, upgrade software, manage dependencies.

Area Identifier	Service Identifier	Area Number	Service Number	Area Version
SoftwareManagement	PackageManagement	7	3	1
Interaction Pattern	**Operation Identifier**	**Operation Number**	**Support in Replay**	**Capability Set**
REQUEST	findPackage	1	Yes	1
INVOKE	install	2	No	
INVOKE	uninstall	3	No	
INVOKE	upgrade	4	No	2
REQUEST	checkPackageIntegrity	5	No	3

Table 3: Package Management service.

The Processes service provides the ability to monitor and manage processes running on-board.

Area Identifier	Service Identifier	Area Number	Service Number	Area Version
SoftwareManagement	ProcessManagement	7	4	1
Interaction Pattern	**Operation Identifier**	**Operation Number**	**Support in Replay**	**Capability Set**
PUBLISH-SUBSCRIBE	monitorProcess	1	No	1
SUBMIT	setRate	2	No	
SUBMIT	startProcess	3	No	2
SUBMIT	endProcess	4	No	3
REQUEST	getProcessSummary	5	No	4

Table 4: Process Management service.

The Apps Launcher service provides the ability to monitor the execution, run, stop, kill and list applications on-board of a spacecraft. The applications can be organized in categories. The service is independent from any particular Operating System or platform.

Area Identifier	Service Identifier	Area Number	Service Number	Area Version
SoftwareManagement	AppsLauncher	7	5	1
Interaction Pattern	Operation Identifier	Operation Number	Support in Replay	Capability Set
PUBLISH-SUBSCRIBE	monitorExecution	1	No	1
SUBMIT	runApp	2	No	2
SUBMIT	killApp	3	No	
PROGRESS	stopApp	4	No	3
REQUEST	listApp	5	Yes	4

Table 5: Apps Launcher service.

The Heartbeat service provides the ability to periodically publish a heartbeat message. Additionally it is possible to get the period of the beat.

Area Identifier	Service Identifier	Area Number	Service Number	Area Version
SoftwareManagement	Heartbeat	7	6	1
Interaction Pattern	Operation Identifier	Operation Number	Support in Replay	Capability Set
PUBLISH-SUBSCRIBE	beat	1	No	1
REQUEST	getPeriod	2	Yes	2

Table 6: Heartbeat service.

The services presented in this section can become the precursor to the standardized CCSDS Software Management services that are known to be part of the CCSDS Agency roadmap for future standardization. [27]

2.4. Platform services

The Platform set of services contains services for monitoring and control of devices on-board of a spacecraft platform. The following services are contained in the Platform services set:

- The Camera service allows a consumer to take and stream pictures.
- The GPS service allows a consumer to retrieve satellite navigation data.
- The Autonomous ADCS service allows a consumer to determine the attitude, and additionally to control the attitude by selecting the desired definition.
- The Software-defined Radio service allows a consumer to configure and receive a stream of data from a Software-defined Radio.
- The Optical Data Receiver service allows a consumer to stream data from the Optical Data Receiver.
- The Magnetometer service allows a consumer to acquire the magnetic field.
- The Power Control service allows a consumer to control the power of the different subsystems.

The Camera service allows a consumer to acquire pictures and control a camera in the spacecraft platform. The service can perform format conversions in case the consumer selects a specific format other than raw. The service can also stream pictures periodically.

Area Identifier	Service Identifier	Area Number	Service Number	Area Version
Platform	Camera	105	1	1
Interaction Pattern	**Operation Identifier**	**Operation Number**	**Support in Replay**	**Capability Set**
PUBLISH-SUBSCRIBE	streamPictures	1	No	1
SUBMIT	setStreaming	2	No	
SUBMIT	unsetStreaming	3	No	
REQUEST	previewPicture	4	No	2
INVOKE	takePicture	5	No	3
REQUEST	getProperties	6	Yes	4

Table 7: Camera service.

The GPS service provides the ability to retrieve satellite navigation data from a Global Navigation Satellite System (GNSS) device receiver in the spacecraft platform. The GPS service provides the capability for streaming NMEA messages; the capability for enabling/disabling the streaming of NMEA messages; the capability for getting the last known position from the receiver; the capability for getting the satellites GNSS information; the capability for maintaining the list of nearby position events.

The nearbyPosition operation allows a consumer to receive a message from the service when the spacecraft enters or exists a certain position. These can be set using the addNearbyPosition and removed using the removeNearbyPosition.

The GPS service operation getLastKnownPosition has been inspired by Android's getLastKnownLocation method from the LocationManager API. [50]

Area Identifier	Service Identifier	Area Number	Service Number	Area Version
Platform	GPS	105	2	1
Interaction Pattern	**Operation Identifier**	**Operation Number**	**Support in Replay**	**Capability Set**
INVOKE	getNMEASentence	1	No	1
REQUEST	getLastKnownPosition	2	No	2
INVOKE	getPosition	3	No	3
INVOKE	getSatellitesInfo	4	No	4
REQUEST	listNearbyPosition	5	No	5
REQUEST	addNearbyPosition	6	No	6
SUBMIT	removeNearbyPosition	7	No	
PUBLISH-SUBSCRIBE	nearbyPosition	8	No	7

Table 8: GPS service.

The ADCS service allows a consumer to monitor the attitude from an ADCS device in the spacecraft platform and to set/unset the desired attitude from a list of attitude definitions.

Area Identifier	Service Identifier	Area Number	Service Number	Area Version
Platform	AutonomousADCS	105	3	1
Interaction Pattern	**Operation Identifier**	**Operation Number**	**Support in Replay**	**Capability Set**
PUBLISH-SUBSCRIBE	monitorAttitude	1	No	1
SUBMIT	setDesiredAttitude	2	No	2
SUBMIT	unsetAttitude	3	No	
REQUEST	listAttitudeDefinition	4	No	3
REQUEST	addAttitudeDefinition	5	No	4
SUBMIT	removeAttitudeDefinition	6	No	

Table 9: Autonomous ADCS service.

The Software-defined Radio provides a generic mechanism to set, configure and receive data from a Software-defined Radio device.

Area Identifier	Service Identifier	Area Number	Service Number	Area Version
Platform	SoftwareDefinedRadio	105	4	1
Interaction Pattern	**Operation Identifier**	**Operation Number**	**Support in Replay**	**Capability Set**
PUBLISH-SUBSCRIBE	streamRadio	1	No	1
SUBMIT	enableSDR	2	No	2
SUBMIT	updateConfiguration	3	No	3

Table 10: Software-defined Radio service.

The Optical Data Receiver service provides a mechanism to receive messages from an Optical Data Receiver device.

Area Identifier	Service Identifier	Area Number	Service Number	Area Version
Platform	OpticalDataReceiver	105	5	1
Interaction Pattern	**Operation Identifier**	**Operation Number**	**Support in Replay**	**Capability Set**
PUBLISH-SUBSCRIBE	streamData	1	No	1
REQUEST	setPublishingFrequency	2	No	

Table 11: Optical Data Receiver service.

The Magnetometer service provides a generic mechanism to retrieve the magnetic field from a magnetometer in the spacecraft platform.

Area Identifier	Service Identifier	Area Number	Service Number	Area Version
Platform	Magnetometer	105	6	1
Interaction Pattern	**Operation Identifier**	**Operation Number**	**Support in Replay**	**Capability Set**
REQUEST	getMagneticField	1	No	1

Table 12: Magnetometer service.

The Power Control service provides a generic mechanism to list the available power units in a spacecraft platform and to enable/disable them.

Area Identifier	Service Identifier	Area Number	Service Number	Area Version
Platform	PowerControl	105	7	1
Interaction Pattern	**Operation Identifier**	**Operation Number**	**Support in Replay**	**Capability Set**
REQUEST	listUnitsAvailable	1	No	1
REQUEST	enableUnit	2	No	2

Table 13: Power Control service.

2.5. NMF Generic Model

The NMF Generic Model defines the set of services and building blocks to be used together and its respective orchestration. This is the foundation to the NMF Composites, a set of specialized components that when connected together create end-to-end scenarios. The MO architecture is the underlying architecture of the NMF Generic Model.

As part of the MO architecture, the MAL is present in order to decouple the Services Layer from the Transport Layer. This allows the use of the same service over different transports. [130]

In order to cover the core functionalities of the framework, the NMF Generic Model takes advantage of five sets of services:

- COM services
- M&C services
- Common services
- Platform services
- Software Management services

The COM services are part of the framework because they include a set of three support services that can be used by all the other services. In the list of support services, the Archive service provides persistence storage including the four CRUD basic functions: create, read, update, delete; then, the Event service provides a mechanism for the distribution of events; and finally, the Activity Tracking service provides the ability to track the progress of activities. [42]

The M&C services provide a set of generic services for monitoring and control of a remote entity and some are part of the framework in order to do monitoring and control of on-board status and activities. This set includes the Parameter service that allows a consumer to set and retrieve parameter values; the Aggregation service for retrieving a collection of parameters on request or periodically; the Alert service for reporting alert status to a consumer; and the Action service for executing activities on the provider. [43]

The Common services provide a set of infrastructure services and are useful in order to support the system. This set includes the Configuration service for service configuration management, which is important for reestablishing the last state of a service upon initialization; the Directory service includes a registry of the providers and their respective services in the system; and the Login service that allows a consumer to provide authentication information to the system. [44]

The CCSDS specified (or is in the process of specifying) the first three sets, COM, M&C and Common services as international standards. However, the CCSDS does not provide dedicated services neither for the monitoring and control of the platform peripherals nor for on-board software management at the time of writing. Thus, two bespoke services sets were defined in order to cover for these two missing core functionalities during the research. These are respectively the Platform services and the Software Management services.

The Platform services provide dedicated services for the monitoring and control of nanosatellite platform peripherals. This set includes services for peripherals such as GPS, Camera, Magnetometer, Software-defined Radio and Optical Data Receiver. In addition, it includes the AutonomousADCS that allows a consumer to control an ADCS unit using high-level definitions with different possible modes such as sun pointing, target pointing mode or nadir pointing mode. Section 3.3.5 explains these services in further detail.

The Software Management services provide a set of services for on-board software management. This set includes the Heartbeat service for broadcasting a periodic heartbeat to all the consumers connected to an application running on-board; the Apps Launcher service for launching applications on-board; and the Package Management service for installing, uninstalling and updating packages on-board. Both the Apps Launcher and the Package Management services are specified without being tied to any particular OS or package management system. Section 2.3 explains these services in further detail.

The Package Management service allows the installation, uninstallation and update of packages on an Operating System using a package management system. The service is not specific to any particular package management system however it is expected to be able to be integrated with the most common ones (rpm, apt, dpkg). During the research, a dedicated package was defined and implemented for the NMF. This is defined as "NMF Package" and more information is available in section 2.7.3.

The same underlying transport layer must be present on the consumer and provider sides in order to exchange data between the two entities. At least one transport layer binding needs to be available in all the NMF Composites in order to guarantee that the components are able to exchange data. Therefore, the NMF Generic Model has TCP/IP transport layer binding as default due its general availability in today's computers, and for being already in the process of becoming a standardized CCSDS transport binding. The NMF Generic Model has the Binary encoding as default due to its simplicity. [47] [48]

If a mission needs to use a mission-specific transport, a protocol bridge can be used to accommodate this need. Section 2.6.5 includes more technical details.

Figure 22 presents a visualization of the NMF Generic Model including the selected sets of services and their respective orchestration in the MO architecture. The figure is abstract and does not represent any specific component nor specifies the deployment location. The transport binding remains undefined because the NMF Composites will select the appropriate transports to be used depending on the intended deployment location of the component and configuration although TCP/IP is available. The MO Core is the collection of MAL, COM services and Common services, and it is further explained in section 2.2.1.

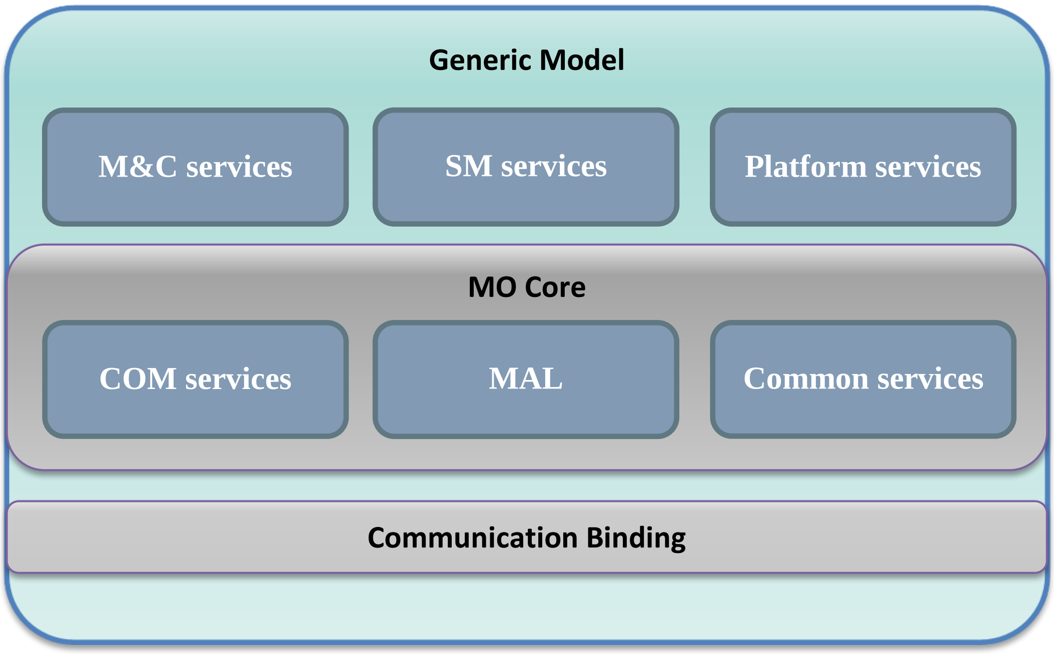


Figure 22: The NMF Generic Model based on the MO architecture.

As previously mentioned the NMF Generic Model uses the MO architecture and specifies the set of services to be orchestrated with it that allow the design of specialized components that include dedicated functionality depending on its use case. These specialized components are the “NMF Composites” and are further described in section 2.6. They can produce complete applications that are able to operate in different system scenarios such as the ones described in section 2.8.

2.6. NMF Composites

An NMF Composite is a software component that consists of interconnected services based on the NMF Generic Model specialized for a certain purpose and to be deployed on the NanoSat segment or Ground segment. The NMF Composites are based on SOA's service composability design principle that encourages reusing existing services and combine them together to build an advanced solution. [51]

While section 2.5 presents the generic model and set of services common to all the NMF Composites, this section presents the specialized components that can be generated by extending the NMF Generic Model. The NMF Composites can be further specialized depending on the mission needs. An example is presented in chapter 5.

The objective of the NMF Composites is to provide prebuilt components that allow quick development of new software solutions that are interoperable in end-to-end scenarios. Some generic scenarios are present in section 2.8.

The design of the NMF Composites is done in a modular and flexible manner which allows them to be reconfigured or adapted depending on the needs in the overall design of the system. This is similar to a Lego® type approach where the bricks can be recombined to form something different.

The names for the NMF Composites follow the convention: <Segment> MO <Purpose>

The family of NMF Composites is the following: [52]

- NanoSat MO Monolithic
- NanoSat MO Supervisor
- NanoSat MO Connector
- Ground MO Adapter
- Ground MO Proxy

The first 3 NMF Composites are supposed to run on the NanoSat segment while the last 2 are meant to run on the Ground segment. The Ground MO Adapter and Ground MO Proxy are components which are intended to be deployed on the Ground segment and when combined with the NanoSat segment part, they are capable of providing end-to-end deployment solutions.

For some deployments, a simple monolithic architecture is sufficient just like it is done on today's OBSW where there isn't a clear separation of concerns into different applications. The NMF still supports this type of approach by using the NanoSat MO Monolithic component however this is not recommended approach for NMF software. Chapter 4 presents the "Monolithic Provider Demo", a provider application that uses the NanoSat MO Monolithic component plugged to a software simulator. This is particularly useful for debugging ground applications by allowing the retrieval of representative data.

For running multiple applications, using the monolithic architecture would imply that each application would load the full stack of services, using resources that could be shared between them. In order to allow the use of the same platform peripheral by different applications, the introduction of two new components was necessary. First, the NanoSat MO Supervisor, responsible for managing the applications (start/stop) and providing an implementation of the Platform services that can be utilized by different applications. And second, the NanoSat MO Connector, the component responsible for connecting to the Platform services to allows the interaction with the peripherals and additionally exposing interfaces to monitor and control the application from a remote consumer.

As a result, it is possible to share hardware resources by different software entities, such as the same GPS unit being accessed by 2 different applications running simultaneously. However, it is important to mention that this might not be possible for every single platform peripheral, for example, an ADCS unit should not be controlled by two different applications simultaneously.

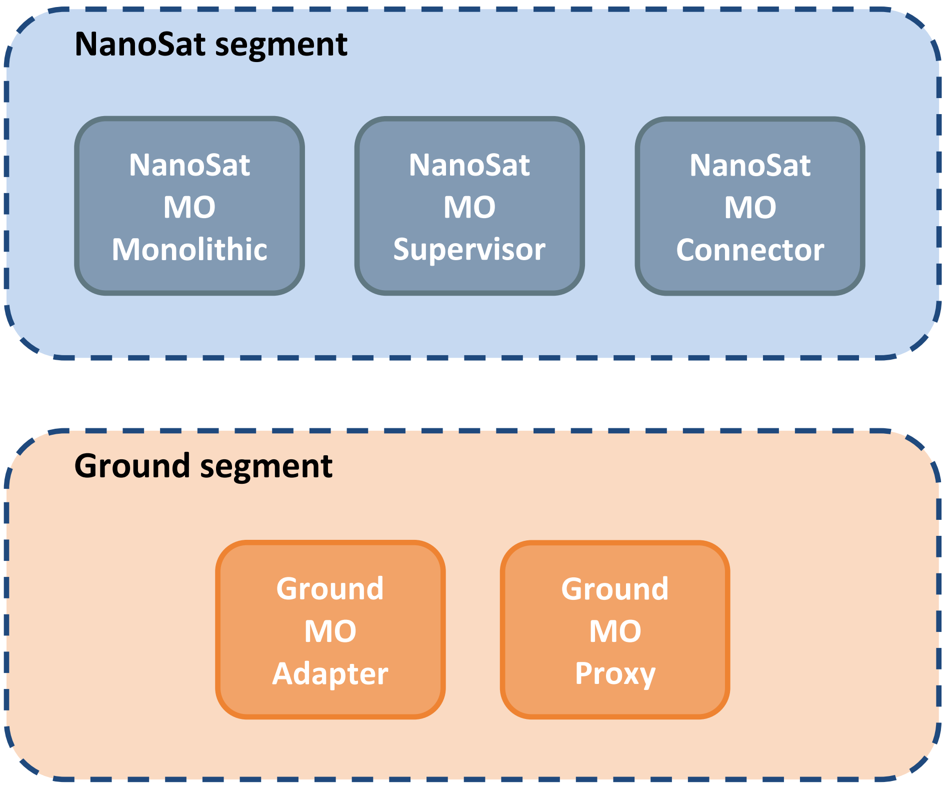


Figure 23: NMF Composites on their respective deployment segment.

2.6.1. NanoSat MO Monolithic

The NanoSat MO Monolithic is an NMF Composite intended to be deployed on the NanoSat segment for an on-board monolithic architecture. It consists of a set of services that facilitate not only the monitoring and control of the nanosatellite, but also the interaction with the platform peripherals. This component must be extended to the specific platform and particular mission use case.

There are three mission-specific parts to be integrated with this component. First, the communication binding to an appropriate transport needs to be selected for communication with ground. Second, the Platform services implementation for the specific platform. Third, the mission-specific logic for unique features dedicated to a specific mission scenario.

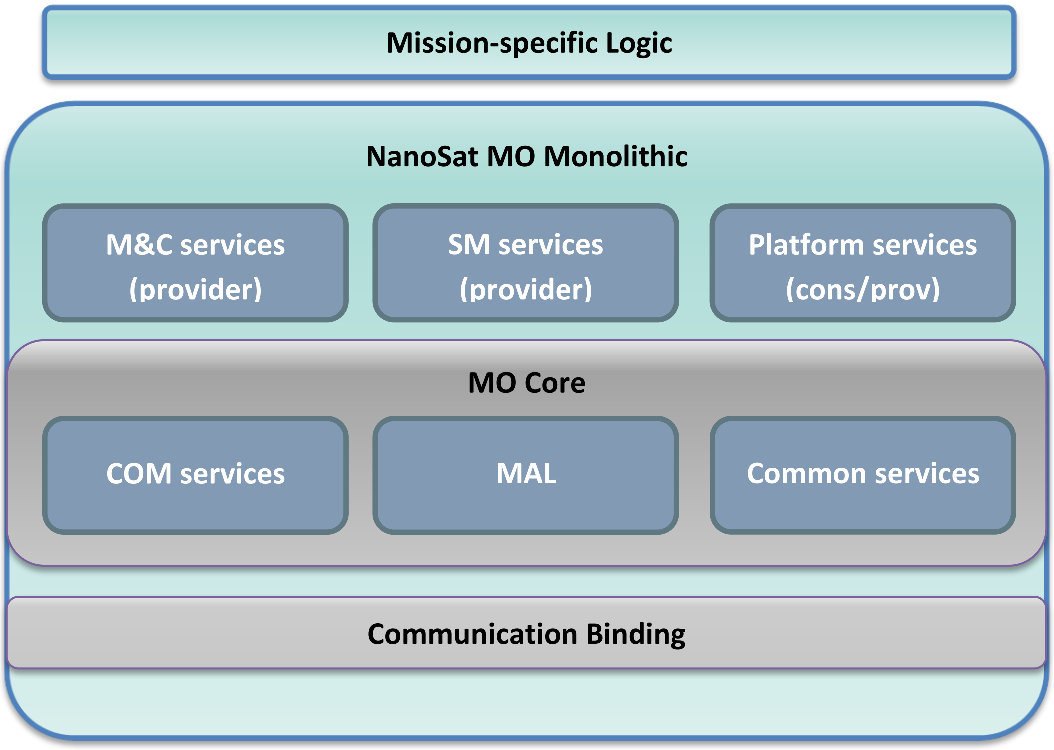


Figure 24: NanoSat MO Monolithic component representation.

The Platform services includes an instance of both the service consumer and provider side. The provider side is intended to be consumed by both ground and the mission-specific logic of the application therefore a service consumer is needed to be present. Section 2.1.3 includes more details about the deployment of the Platform services in a single process.

In chapter 4, the NanoSat MO Monolithic was integrated with an implementation of the Platform services connected to an OPS-SAT Software Simulator. This allows debugging the logic of an NMF App using meaningful data coming from the Platform services.

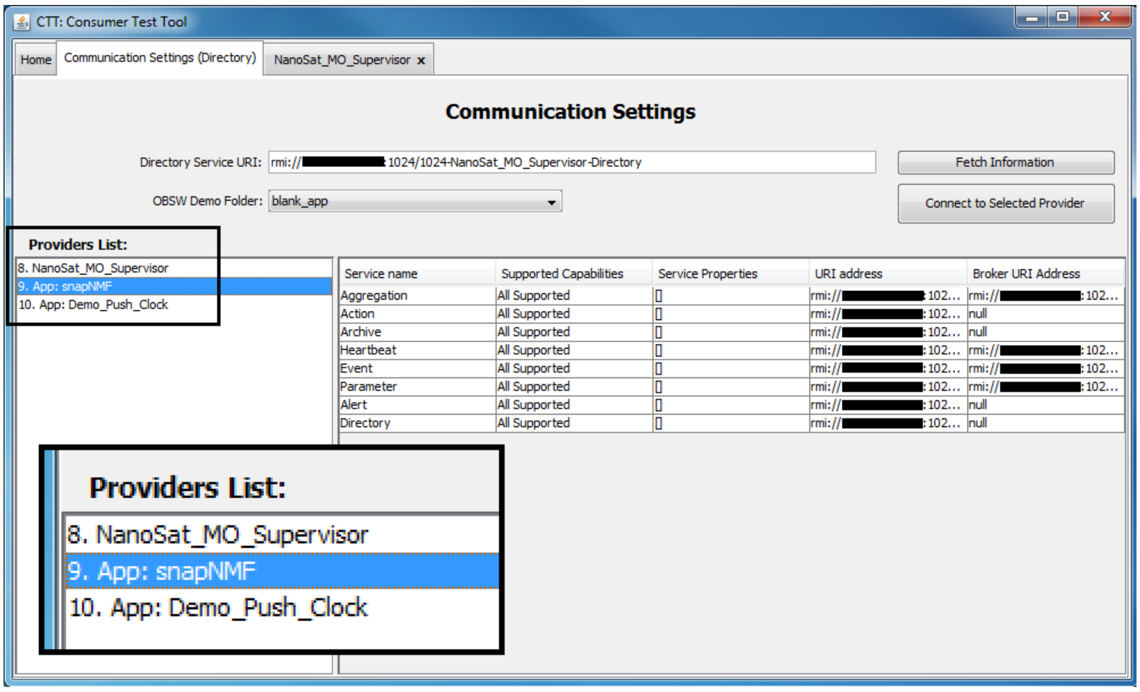
For a deployment that consists of a single application running on-board or for a constrained device incapable of running a full operating system, this simplistic monolithic approach is a possibility. However this is not the recommended approach because it does not take full advantage of the new NanoSat MO Framework concepts.

In order to have multiple applications running on the same machine with the NMF, two components were defined: the NanoSat MO Supervisor and the NanoSat MO Connector. These are described on the next sub-sections.

2.6.2. NanoSat MO Supervisor

The NanoSat MO Supervisor is an NMF Composite that acts as a supervisor and it is intended to be deployed on the NanoSat segment for an on-board deployment with multiple applications. It supports the discoverability of the providers available on-board, and it allows different applications to interact with the peripherals on-board via a set of Platform services. Additionally, it includes software management capabilities such as: starting, stopping, installing, updating and uninstalling applications on-board of the spacecraft. This component must be extended to the specific platform and particular mission use case.

This component includes the Central Directory service that provides discoverability functionality by holding the connections information about all the providers running on-board. Specifically, the NanoSat MO Supervisor provider and the set of registered running provider applications with their respective services. Figure 25 presents an example of a set of registered providers running on the Central Directory service.


CTT: Consumer Test Tool

Home | Communication Settings (Directory) | NanoSat_MO_Supervisor x

Communication Settings

Directory Service URI: rmi://[redacted]:1024/1024-NanoSat_MO_Supervisor-Directory

OBSW Demo Folder: blank_app

Fetch Information

Connect to Selected Provider

Providers List:

8. NanoSat_MO_Supervisor
9. App: snapNMF
10. App: Demo_Push_Clock

Service name	Supported Capabilities	Service Properties	URI address	Broker URI Address
Aggregation	All Supported	[]	rmi://[redacted]:102...	rmi://[redacted]:102...
Action	All Supported	[]	rmi://[redacted]:102...	null
Archive	All Supported	[]	rmi://[redacted]:102...	null
Heartbeat	All Supported	[]	rmi://[redacted]:102...	rmi://[redacted]:102...
Event	All Supported	[]	rmi://[redacted]:102...	rmi://[redacted]:102...
Parameter	All Supported	[]	rmi://[redacted]:102...	rmi://[redacted]:102...
Alert	All Supported	[]	rmi://[redacted]:102...	null
Directory	All Supported	[]	rmi://[redacted]:102...	null

Providers List:

8. NanoSat_MO_Supervisor
9. App: snapNMF
10. App: Demo_Push_Clock

Figure 25: A zoom-in visualization of the 3 providers available on a Central Directory service presented on the Consumer Test Tool.

The implementation of the Platform services to the platform peripherals on the NanoSat MO Supervisor allows multiple consumers to interact with the peripherals available on-board. The connection between a consumer and a provider of a Platform service is expected to happen via Inter-Process Communication (IPC). Figure 26 presents an NMF App connected to the NanoSat MO Supervisor.

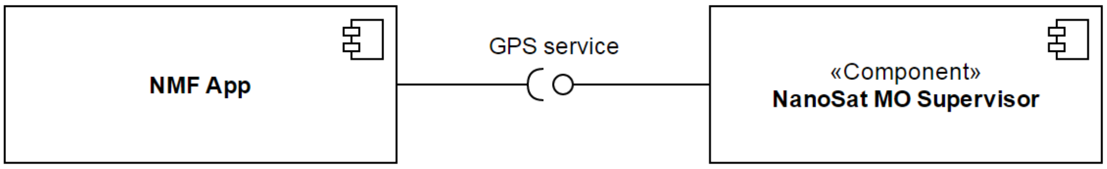


Figure 26: Component diagram showing an NMF App connected to the GPS service of the NanoSat MO Supervisor.

The NanoSat MO Supervisor is able to install packages carrying applications or libraries via the Package Management service. After an application is installed, the Apps Launcher service can be used to start the application and afterwards stop it. The Package Management is able to update a certain package which allows, for example, new features or bug fixes to be ported into an application. Lastly, a package can be uninstalled by the NanoSat MO Supervisor. Figure 27 presents a sequence diagram with the lifecycle of an application.

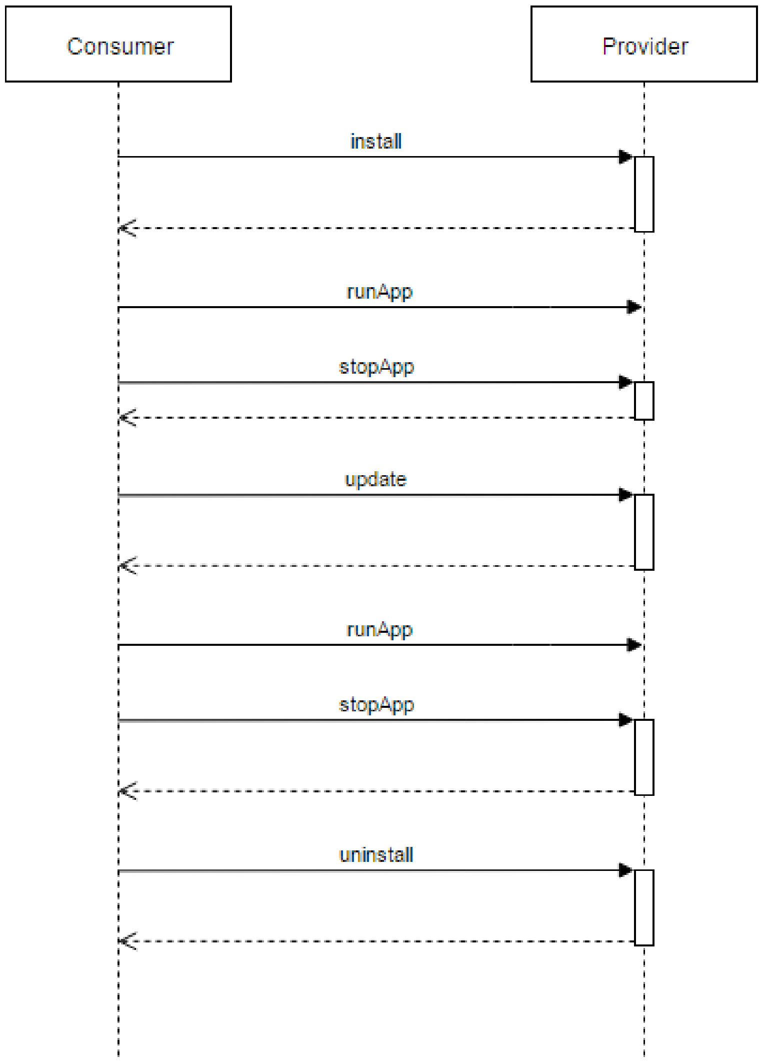


Figure 27: Sequence Diagram of the expected lifecycle of an application.

2.6.3. NanoSat MO Connector

The NanoSat MO Connector is an NMF Composite that acts as a connector to the NanoSat MO Supervisor and it is intended to be deployed on the NanoSat segment for an on-board deployment with multiple applications. It is able to connect to the Directory service and register its services, connect to the Platform services to interact with the on-board peripherals, receive CloseApp event requests from the supervisor, and unregister from the Directory service. Additionally, it can provide monitoring and control capabilities to the layers above this component. This component doesn't need to be extended but it is expected to be integrated with other software to form an NMF App.

The interactions between the NanoSat MO Connector and the NanoSat MO Supervisor are expected to occur via IPC when they both run on the same operating system. This means that 2 different transports might coexist simultaneously, one for ground communications and another for IPC.

Upon startup, it shall make available the service provider connections to ground and then connects to the NanoSat MO Supervisor services and register the exposed services on the Central Directory service for visibility.

It is possible to find the service URIs of the Platform services in the Central Directory service and then connect to them using the consumer stub of the desired service. Figure 28 displays an example for the GPS service.

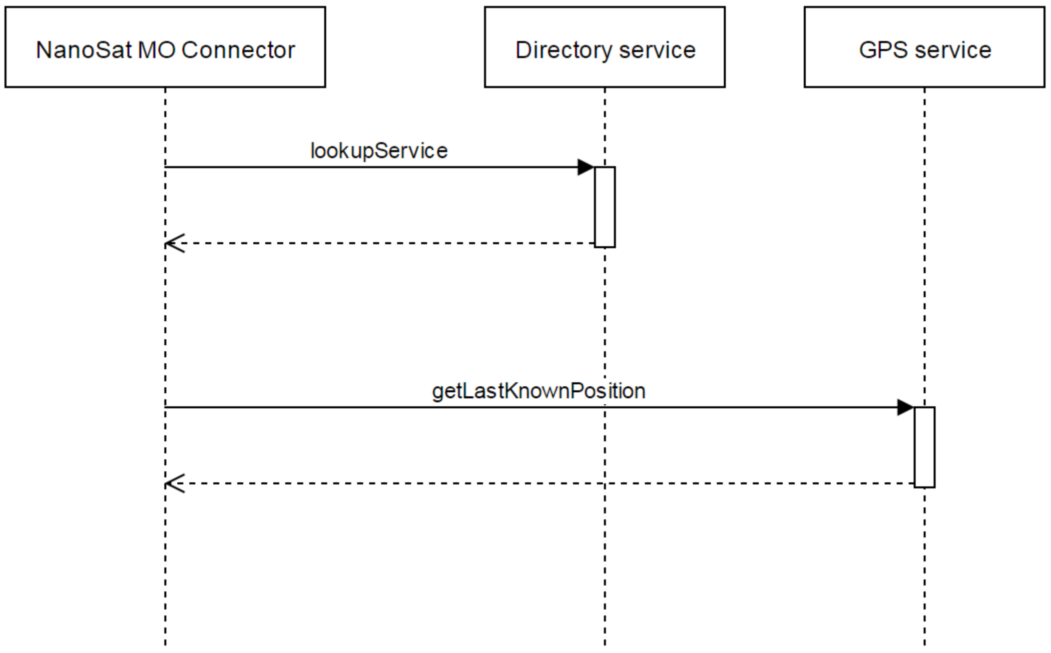


Figure 28: Sequence Diagram of an example showing the NanoSat MO Connector finding the GPS service address and then calling an operation.

After the stopApp operation is called from ground on the NanoSat MO Supervisor, it will send an event to instruct the NanoSat MO Connector to close. After receiving the event to close, the component will unregister the application from the Central Directory service and gracefully close the application. Additionally, an adapter can be set which will execute application-specific logic upon the reception of such event for the gracefully termination of the application logic.

Optionally, the NanoSat MO Connector can provide monitoring and control capabilities that can be integrated by the developer via the registration of Parameters definitions, Aggregations definitions, Alerts definitions and Actions definitions. The M&C services' logic doesn't need to be reimplemented multiple times and instead, it can be implemented one single time and used by the developers by simply taking advantage of the adapter pattern for the retrieval of parameters and execution of actions.

The NanoSat MO Connector allows the development of NMF Apps. This is further described in the NMF App section and how they are related.

2.6.4. Ground MO Adapter

The Ground MO Adapter is an NMF Composite that enables ground systems to connect to MO providers. It is intended to be deployed on ground and it is able to connect to all the other NMF Composites. Service interfaces are exposed to the ground system for interactions with MO providers. The services' communication configurations can be automatically discovered from the Directory service by using the provider's URI of the Directory service.

This component is foreseen to be used for the development of Monitor and Control Systems (MCS) or ground applications capable of interfacing with an NMF App however it is not limited NMF software. This means that a ground consumer using this component shall be able to connect to other different providers, such as a simple provider with only a Parameter service.

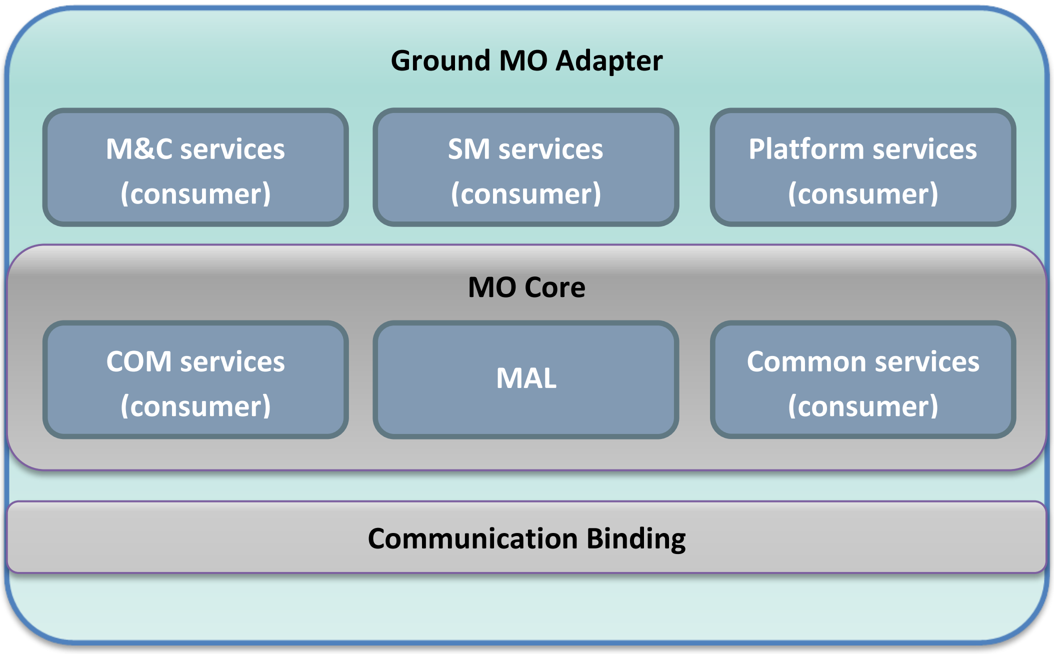


Figure 29: Ground MO Adapter component representation.

The integration of this component with another ground system application is reduced to linking the services that are intended to be used to the rest of the application. This integration can be done for an MCS, a mission automation system, or a dedicated application designed to connect to a specific NMF App.

An example of the integration with an MCS is presented in in section 5.2.3.EUD4MO is a generic monitoring and control system that connects to MO providers which a user can interact with using a simple web browser.

An example of the integration with a mission automation system is presented in chapter 7. AutoMATISMO is a conceptual idea of an SMF driver that would allow MATIS, a mission automation system developed by ESA, to connect to MO providers. [53]

An example of the integration with a dedicated application designed to connect to a specific NMF App is presented in section 4.7.2. This ground demo publishes a post to the Facebook social network when a certain parameter value from the space counterpart is pushed to ground.

The Consumer Test Tool (CTT) is a tool to test newly developed NMF Apps and it is part of the NMF SDK. This tool integrates the Ground MO Adapter and provides an easy GUI to interact with some of the services available on the NMF App. Section 4.3 presents CTT in further detail. Additionally, a set of Ground demos exemplifying how to use the Ground MO Adapter are also presented in chapter 4.

The Ground MO Adapter opens many possibilities for the integration with other ground systems. Some of these are explained in section 7.

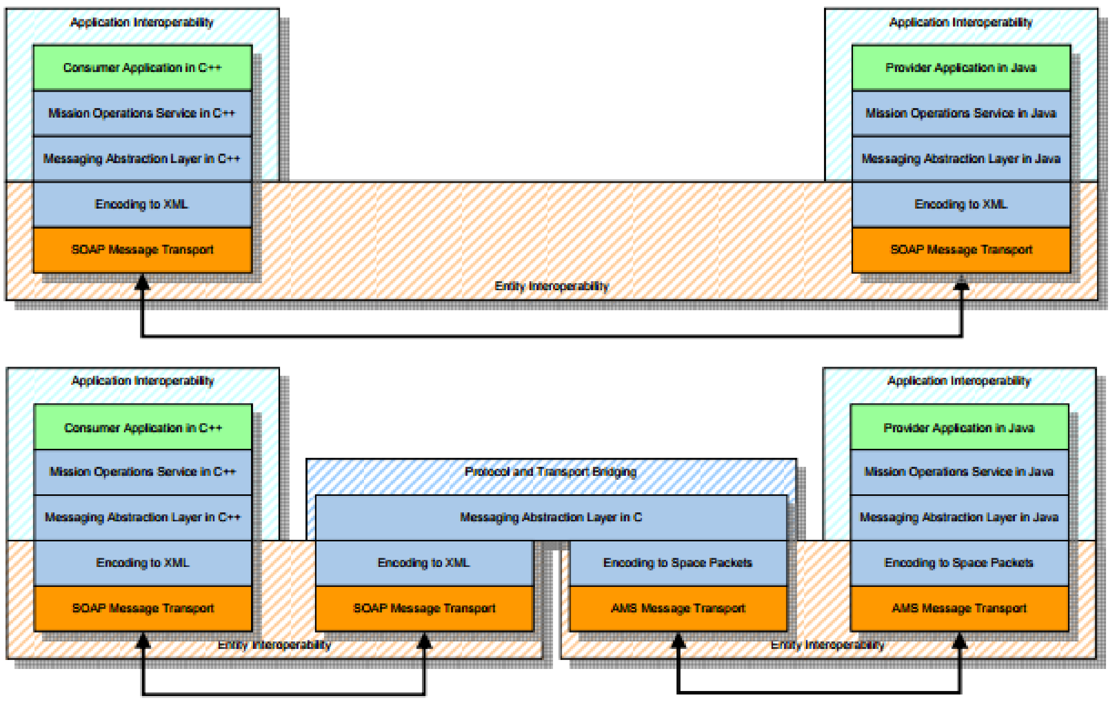
2.6.5. Ground MO Proxy

The Ground MO Proxy is an NMF Composite that acts as a proxy for ground systems that use the Ground MO Adapter. The Ground MO Proxy has 3 main functionalities, to act as a protocol bridge, as a COM Archive mirror, and it allows the queuing of actions. As a result, multiple consumers can share the same ground-to-space connection to the spacecraft, and connect to independent NMF Apps simultaneously.

This component has been included as part of the NMF Composites in order to solve the problem of using mission-specific protocols in the space-to-ground link while still remaining agnostic from any mission.

The MO Architecture allows the creation of a protocol-matching bridge or Gateway that allows translation from one encoding/transport choice to another because it supports the separation of information interoperability (MAL and higher layers) and protocol interoperability (encoding and transport). Figure 30 shows an example of a protocol bridge with different transport/encodings. [54]

The Ground MO Proxy shall connect to the NanoSat MO Supervisor, typically via SPP (but IP connections are not uncommon in nanosatellites), and expose on ground all the providers available on the Central Directory service. The connections details are edited to include the URI of the protocol bridge to route correctly forthcoming ground consumers. Examples of suitable protocol transports that can be exposed to ground consumers are HTTP, TCP/IP, ActiveMQ and/or RMI.



Application Interoperability
Consumer Application in C++
Mission Operations Service in C++
Messaging Abstraction Layer in C++
Encoding to XML
SOAP Message Transport
Application Interoperability
Provider Application in Java
Mission Operations Service in Java
Messaging Abstraction Layer in Java
Encoding to XML
SOAP Message Transport
Entity Interoperability
Application Interoperability
Consumer Application in C++
Mission Operations Service in C++
Messaging Abstraction Layer in C++
Encoding to XML
SOAP Message Transport
Protocol and Transport Bridging
Messaging Abstraction Layer in C
Encoding to XML
SOAP Message Transport
Encoding to Space Packets
AMS Message Transport
Application Interoperability
Provider Application in Java
Mission Operations Service in Java
Messaging Abstraction Layer in Java
Encoding to Space Packets
AMS Message Transport
Entity Interoperability
Entity Interoperability

Figure 30: Protocol Bridge Example [55]

When the connection to the spacecraft is done using the MAL-SPP transport, address mapping has to be defined. This is because the MAL-SPP transport binding does not support routing as part of the binding protocol. A solution is to use a range of APIDs dedicated to the creation of virtual URIs that are dynamically mapped to the source URI upon the connection of a new consumer. [47]

This solution was implemented for OPS-SAT's mission and further information is available in chapter 5.

Figure 31 shows a Proxy component acting as a consumer of Service A from the Provider component and as a provider of Service C to the Consumer component. A Proxy provides a way of exposing a service to external consumers where direct visibility would not be desirable. In the example the Proxy component is also acting as a protocol bridge; however, this is not required. [54]

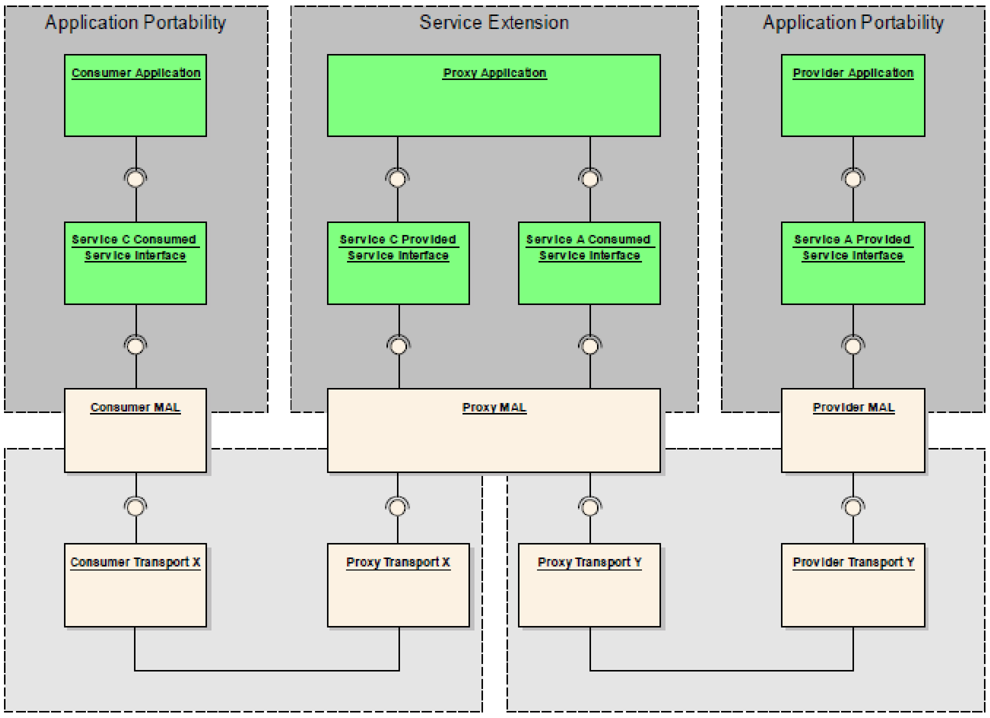


Figure 31: Service Extension Example [54]

For some services, the Ground MO Proxy follows the concept of proxy service extension defined in the MO Reference Model book. This allows, for example, queuing actions, exposing a Directory service with re-routed URIs to the correct location, having an Archive service replica on ground in order to minimize the access to the on-board Archive services of each NMF App and activity tracking for the forwarding of actions.

2.7. NMF Concepts

This section defines the NMF Concepts of the NanoSat MO Framework. These concepts are defined in order to explain the research throughout the dissertation. Some of them are interconnected.

2.7.1. NMF App

An NMF App is an on-board software application based on the NanoSat MO Framework. Upon start-up, it registers itself in the Central Directory service for visibility from ground and unregisters before terminating gracefully. Additionally, it can connect and interact with the Platform services implementation available on the nanosatellite. An NMF App can be monitored and controlled from ground by taking advantage of the CCSDS-standardized M&C services if the services are available.

An NMF App is developed by integrating the NanoSat MO Connector component into the software application and they are expected to be started, monitored, stopped, or killed by the NanoSat MO Supervisor component.

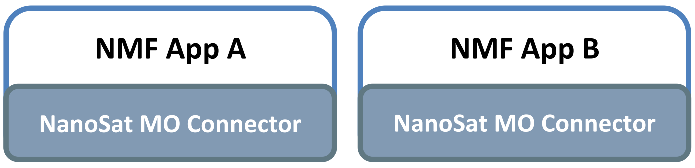


Figure 32: Conceptual visualization of two NMF Apps.

On-board software portability is a key principle of the NMF design that allows the use of the same NMF App in different nanosatellites. Section 2.1.3 presents in more detail how this is achieved.

The simplest NMF App needs as minimum requirements to:

- Have a Directory service provider and consumer
- Have an Event service provider and consumer

The Directory service provider is necessary to hold the list of available services by the NMF App. The Directory service consumer is necessary for finding the Platform services in the system and to register the services of the NMF App to be exposed to ground and other consumers. The Event service consumer is necessary in order to listen for “Close App” events on the Event service from the NanoSat MO Supervisor. The Event service provider is necessary to release an event to the NanoSat MO Supervisor explicitly informing that the NMF App is closing.

More complex NMF Apps can have the CCCSDS MO Monitor and Control services in order to be monitored and controlled from a consumer. However this is not mandatory for an NMF App. This is not enforced because there are use cases where having monitor and control services might not be necessary, for example, if a simple NMF App just needs to read periodically a value from the GPS service and store it in a file. For more complex applications, the use of the M&C services is of course advised.

The NanoSat MO Supervisor is also expected to install, uninstall and update NMF Apps via packages. A dedicated package format for the NMF was defined and it is presented in section 2.7.3.

It is important to mention that throughout the dissertation, the word "application" is not used interchangeably with "NMF App". An "application" is defined as a computer program designated to perform a set of defined functions, tasks, or activities in a computer system. In this sense, an application might or might not be an NMF App.

Chapter 4 presents an NMF Playground that lets developers try out the developed NMF App, the lifecycle of an NMF App, and also presents some NMF App demos. Although these demos are developed based on the Java implementation, an NMF App is independent of the programming language. Section 7.2.1 presents how a connector in C could be developed.

Nowadays, there's a high demand for mobile app software developers in the market mainly because of the new generation of phones with advanced processing capabilities that can run full Operating Systems that have internet access most of the time. These mobile app developers do not know the implementation details of every single smartphone and instead they develop their apps against well-defined interfaces that let them do the operations they intend on most of the smartphones. [56]

When a mobile app is developed, it is never intended to cover all the possible functionalities of a normal phone but instead they are software entities dedicated to executing particular tasks in order to reach the user goals. Transposing this idea to the space domain could potentially create a new set of space software developers focused on particular satellite operations and functionalities. Just as the Uber™ app and Snapchat™ app have very different purposes to the user. [56]

By shifting the software development approach to a low risk activity focused on nanosatellite-specific functionality rather than the development of complete systems, smaller companies are be able to specialize on particular spacecraft functionalities and easily enter the space software arena with specialized software functionalities that could be reused by different spacecraft. This on-board software development fits into the NMF App Development problem domain mentioned in section 2.1.2. [56]

2.7.2. NMF Ground application

An NMF Ground application is a ground software application based on the NanoSat MO Framework. An NMF Ground application is developed by integrating the Ground MO Adapter component into the software application. This makes it able to connect to any NMF App and also to connect to the Ground MO Proxy.

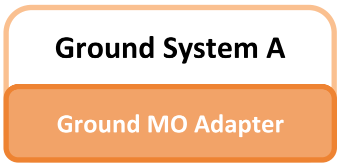


Figure 33: Conceptual visualization of a NMF Ground application.

2.7.3. NMF Package

A NMF Package is a digital content distribution package intended to be used to install, uninstall and update applications on a spacecraft system that uses the NanoSat MO Framework. The package shall support versioning, digital signature and file checksums in order to facilitate updates, prevent content tainting and checking for possible data corruption.

An application inside a NMF Package doesn't necessarily need to be a NMF App and instead it can also be a dedicated or spacecraft-specific application.

The Package Management service does not handle digital distribution of software (transfer of files). It is responsible solely for installing, uninstalling and updating the applications contained inside the packages. In order to transfer the packages to the spacecraft, third-party software for file transfer is necessary.

The NMF SDK includes a tool with a simple GUI capable of generating NMF Packages. This is presented in chapter 4.

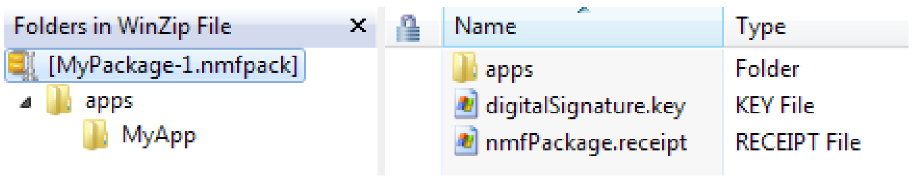
An NMF Package is a simple zip file with the extension ".nmfpack" that follows the following conditions:

- The package shall contain a 'nmfPackage.receipt' file.
- The package shall contain a 'digitalSignature.key' file.
- The package may contain an 'apps' folder.
- The package may contain a 'libs' folder.

Additional conditions are:

- The package may contain both an ‘apps’ folder and a ‘libs’ folder simultaneously.
- The package may contain either an ‘apps’ folder or a ‘libs’ folder.
- The package shall not be valid if the ‘apps’ folder and the ‘libs’ folder simultaneously do not exist.

In Figure 34, it is possible to visualize an NMF Package based on the conditions mentioned above.


Folders in WinZip File
[MyPackage-1.nmfpack]
apps
MyApp

Name	Type
apps	Folder
digitalSignature.key	KEY File
nmfPackage.receipt	RECEIPT File

Figure 34: Content of an NMF Package.

For the “apps” folder the conditions are the following:

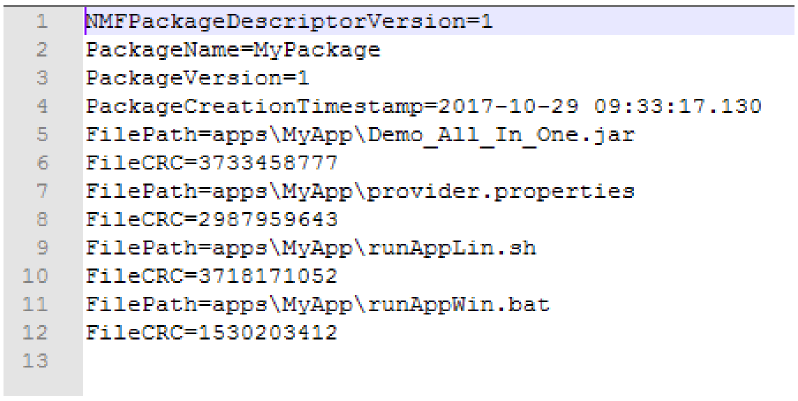
- The folder apps shall contain the apps to be digitally distributed.
- A package is not limited to a single app and therefore can have 2 or more different apps.
- The folder apps shall not exist in case the package is only distributing libraries.

For the “libs” folder the conditions are the following:

- The folder libs shall contain the libraries to be digitally distributed.
- A package is not limited to a single library and therefore can have 2 or more different libraries.

The file nmfPackage.receipt is a NMF Package statement that contains metadata information about the package. Version 1 of the receipt file specifies that the nmfPackage.receipt contains a list of the files and folders contained in the package and their respective checksums. It also contains the version and additional information about the package. The NMF Package concept allows further evolution by supporting newer version of the receipt file. These new versions can hold more metadata.

In Figure 35, it is possible to visualize an NMF Package receipt file.



```
NMFPackageDescriptorVersion=1
PackageName=MyPackage
PackageVersion=1
PackageCreationTimestamp=2017-10-29 09:33:17.130
FilePath=apps\MyApp\Demo_All_In_One.jar
FileCRC=3733458777
FilePath=apps\MyApp\provider.properties
FileCRC=2987959643
FilePath=apps\MyApp\runAppLin.sh
FileCRC=3718171052
FilePath=apps\MyApp\runAppWin.bat
FileCRC=1530203412

```

Figure 35: Example of an NMF Package receipt file.

As mentioned in section 1.5, the idea of packages is not new and in Linux-based systems, the installation of software has been long done through package management systems. However, creating a dedicated package for nanosatellites and apply the same concept in space has never been done before. [30]

The possibility for an open repository of NMF Packages online would increase the reuse of software among nanosatellite developers. Chapter 7 presents this idea.

2.7.4. NMF Mission

An NMF Mission is a concrete implementation of the NanoSat MO Framework for a specific mission. The NMF Mission development includes activities such as implementing the Platform services and the NanoSat MO Supervisor for the specific platform. If a custom or tailored transport is used for the mission, then the transport binding must be implemented and additionally, integrated with the Ground MO Proxy for protocol bridging.

An important part of an NMF Mission is to define the end-to-end scenario to describe how the system operates. Section 2.8 presents some generic scenarios that can be used to build something more complex such as the end-to-end scenario for OPS-SAT mission which is presented in section 5.2.

By reusing parts of an NMF implementation, it is possible to quickly develop a mission-specific service however an application connecting to this service would no longer be considered as an NMF App because it is no longer mission-agnostic.

In order to develop an NMF Mission, an NMF implementation is necessary. Chapter 3 presents a Java implementation of the NMF and chapter 5 presents the NMF Mission for OPS-SAT developed on top of this implementation. This approach promotes reusability and minimizes the development effort per mission.

There is also an NMF Mission: Software Simulator implementation as part of the SDK. This allows an NMF App developer to get some simple data from the Platform services without needing to have a full and heavy simulator deployed. Chapter 4 includes a section dedicated to the lightweight software simulator.

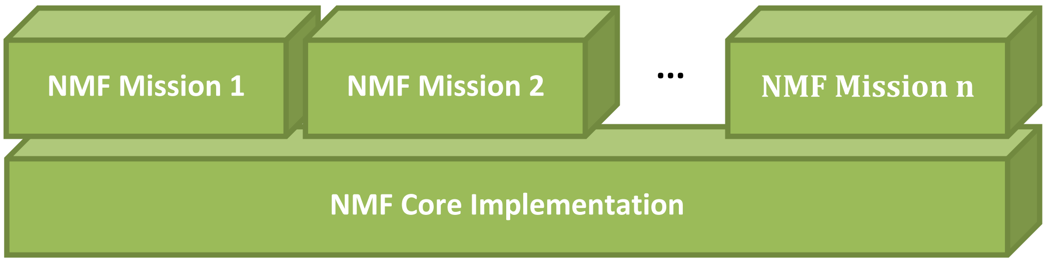


Figure 36: NMF Missions developed on top of an NMF Implementation

As mentioned in section 2.1.2, the NMF Mission development is separate from the NMF Ground development and also from the NMF App development. This means that it is possible to have two different implementations of the NMF in different languages and still be able to connect the NMF Composites to each other as section 2.8 describes.

2.7.5. NMF Library

An NMF Library is a shared library made available on the on-board computer that includes the NMF Mission implementation software. It includes all the necessary dependencies to execute the implementation of the NanoSat MO Supervisor for that mission and also NMF Apps.

By definition, shared libraries are intended to be shared by different applications and they include resources, pre-written code, subroutines, classes, and others, that are loaded at runtime. In contrast, a static library is added with the executable file by the compiler which replicates the same logic multiple times. Figure 37 depicts both types.

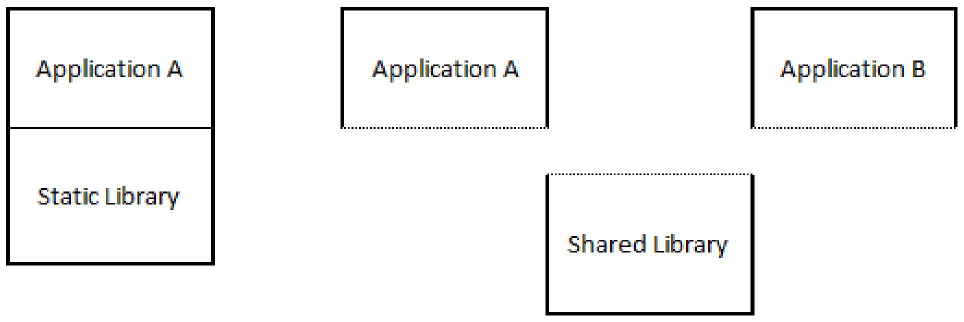


Figure 37: Difference between a static library and a shared library in relation to the application.

The use of shared libraries reduces the file size of the applications because it holds just the logic part of the application while the common code is loaded from the shared library when needed. An NMF Package can take advantage of this reduction in the application's file size in order to reduce the necessary time for the uplink of the package to the nanosatellite. For example, an NMF App would expect to have the NanoSat MO Connector available on-board bundled with the NMF Library and thus it does not need to be inside the package.

On the ground segment, it is not necessary to have shared libraries for the applications because the amount of memory storage on ground computers is not as constrained as in space and the amount of resources is greater.

An NMF Library for OPS-SAT was developed based on the NMF Java Implementation. NMF Apps developed in Java running on-board can link to this library that contains the NanoSat MO Connector component and therefore the compiled jar file doesn't need to integrate it. This technique minimizes the content needed to be transferred to the spacecraft, to just the logic of the application. Chapter 5 includes a dedicated section demonstrating the deployment with an NMF Library for OPS-SAT.

2.7.6. NMF SDK

The NanoSat MO Framework Software Development Kit (NMF SDK) is a set of development tools and software source code that facilitate the creation of applications with the NanoSat MO Framework.

It is composed of:

- Demos for NMF Ground software development
- Demos of NMF Apps
- Consumer Test Tool (CTT)
- NMF Package Assembler
- NMF Playground
- Documentation

The NMF SDK is the starting point for a software developer willing to develop applications with the NMF. Chapter 4 is fully dedicated to the description of the SDK.

2.8. Generic scenarios

This section includes common scenarios that can be put together with the NMF Composites. It is possible to compose complex scenarios by combining multiple of the generic scenarios. One example of a complex scenario is presented in section 2.8.6.

Some of these scenarios are only possible because of the flexibility gained from the idea of separating the on-board software into multiple applications.

The scenarios represent the applications down to the NMF Composites and do not go to the layers below that. The transport binding is only color coded in order to differentiate between multiple possibilities in any specific scenario. The scenarios also do not represent the system where they are deployed nor any system's components in between.

2.8.1. Simple Scenario

The Simple Scenario is the most basic end-to-end scenario that can be assembled and it consists of an application developed using the Ground MO Adapter connected to an NMF App. Figure 38 shows this scenario.

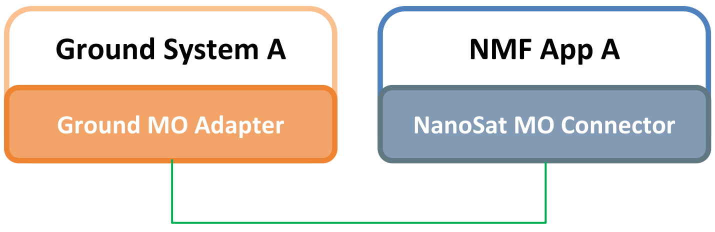


Figure 38: Simple Scenario example.

Both the NanoSat MO Connector and the Ground MO Adapter extend the NMF Generic Model, and thus the applications are able to exchange data using a MAL-TCP/IP Binding with Binary encoding as this transport binding is expected to be in all NMF Composites. Moreover, the software development of the 2 applications is separate in two different problem domains so, as long as both rely on the NMF Composites to exchange data, they can be used for any NMF Mission.

The Ground MO Adapter is not limited to the NanoSat MO Connector component and can also be used for the NanoSat MO Supervisor and for the NanoSat MO Monolithic therefore the scenario could be further generalized, although this would be more complex to describe.

This scenario is expected to be used for both the NMF Ground development and NMF App development problem domains. On the NMF Ground development problem domain, a developer can use the NMF SDK to start developing the ground application by taking as example one of the ground demos and use one of the demo NMF Apps to test the counterpart. On the NMF App problem domain, a developer can use CTT to test the developed NMF App. Chapter 4 presents more details about the demos and CTT.

2.8.2. Multiple Consumers Scenario

The Multiple Consumers Scenario consists of multiple consumers connected to a provider either via a protocol bridge or directly to a NMF App. Figure 39 presents an example with 2 consumers connected to a single NMF App.

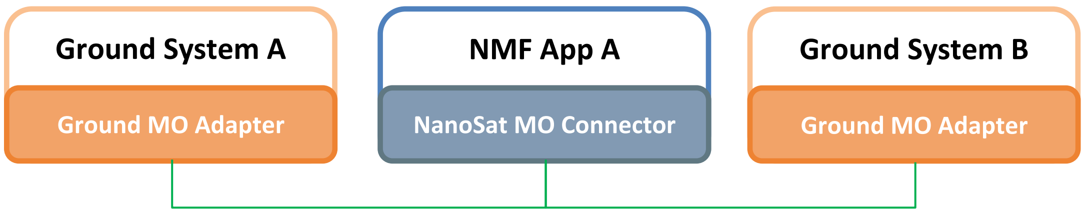


Figure 39: Multiple Consumers Scenario example.

2.8.3. Multiple NMF Apps Scenario

The Multiple NMF Apps Scenario consists of multiple NMF Apps running on the NanoSat segment that are connected to the Platform services of the NanoSat MO Supervisor using the NanoSat MO Connector. Figure 40 shows this scenario.

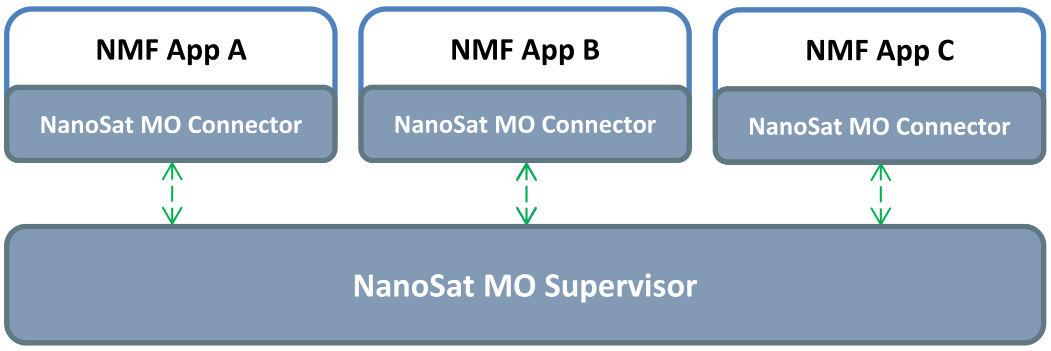


Figure 40: Multiple NMF Apps Scenario example.

The NanoSat MO Supervisor includes the Central Directory service that plays an important role in this scenario as it allows NMF Apps to be found. A ground application can connect to the Central Directory service and find the right NMF App to connect to. NMF Apps can also use the Central Directory service to find each other.

This scenario is expected to be deployed in the NanoSat segment therefore the communications between the NMF Apps and the NanoSat MO Supervisor are expected to occur via IPC.

2.8.4. Proxy Scenario

The Proxy Scenario consists of an application developed using the Ground MO Adapter connected to an NMF App via a Ground MO Proxy implemented for a specific mission. Figure 41shows an example.

The purpose and added functionality of having a Ground MO Proxy is presented in section 2.6.5. Summarizing, it acts as a protocol bridge, a COM Archive replica, a Directory service rerouting the connections for the ground consumers, and queueing of actions if necessary.

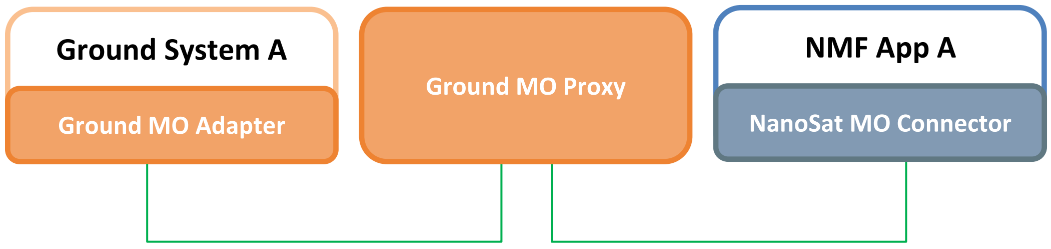


Figure 41: Ground MO Proxy connected to an NMF App.

Ground consumers can connect to NMF Apps through the proxy. The scenario is initiated by connecting a Ground MO Proxy to a NanoSat MO Supervisor. After the connection is established, it will look up for the available providers on the Central Directory service of the NanoSat MO Supervisor and then edit the URIs of the services to have the URI including the protocol bridge URI. This allows ground consumers to be able to find the providers even if there is no link to the spacecraft available.

2.8.5. Other Envisaged Scenarios

This section presents other envisaged scenarios that can be created. These have not been implemented during the research work however they are technically feasible.

The Cascading NMF Apps Scenario consists of multiple NMF Apps connected to each other in a cascading way. It is possible to create new solutions where an NMF App exposes services and connects to the services exposed by another NMF App at the same time. This is based on the service composability design principle of SOA. Figure 42 shows this scenario. [51] [56]

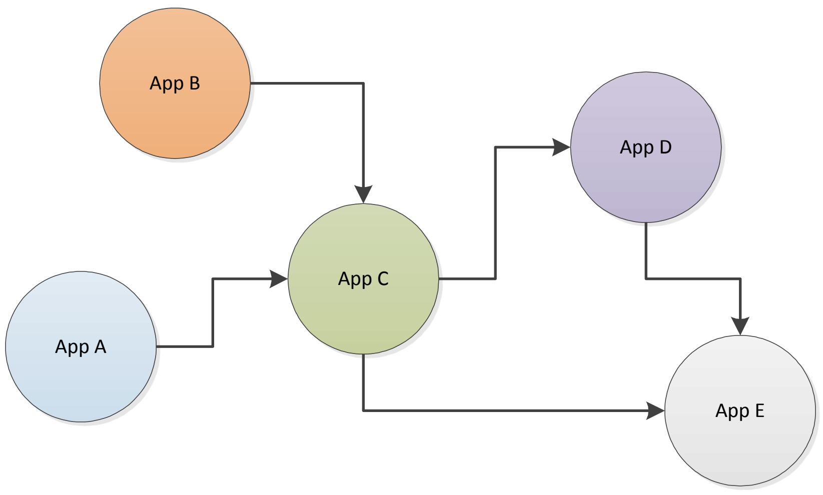


Figure 42: Cascading NMF Apps Scenario.

The Multiple Ground MO Proxies Scenario consists of connected Ground MO Proxy instances deployed in different locations. For example, one Ground MO Proxy could be connected directly to the NanoSat segment and then a second one on a remote location connected to the first. This would allow the first one to open a single TCP/IP port to the second Ground MO Proxy in the remote location and then the remote proxy would allow the access of multiple consumers internally such as it is presented in Figure 43.

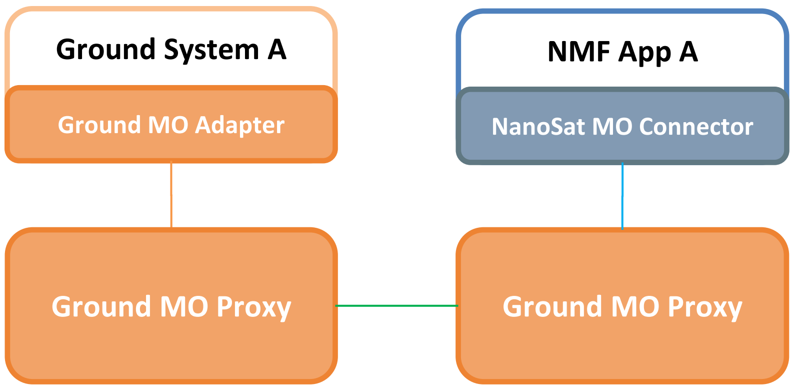


Figure 43: Multiple Ground MO Proxies Scenario.

Another envisaged scenario could be for constellations of nanosatellites. These could deploy an instance of the same NMF App in all the spacecraft in order to achieve a certain goal. The different instances could be aware of each other for better performance of the task to be accomplished by the constellation.

2.8.6. Complex Scenario Example

This section presents an example of a possible scenario that can be created by combining some of the generic scenarios presented before. Figure 44 presents a complex scenario example composed of the generic scenarios: Multiple Consumers, Multiple NMF Apps, and Proxy.

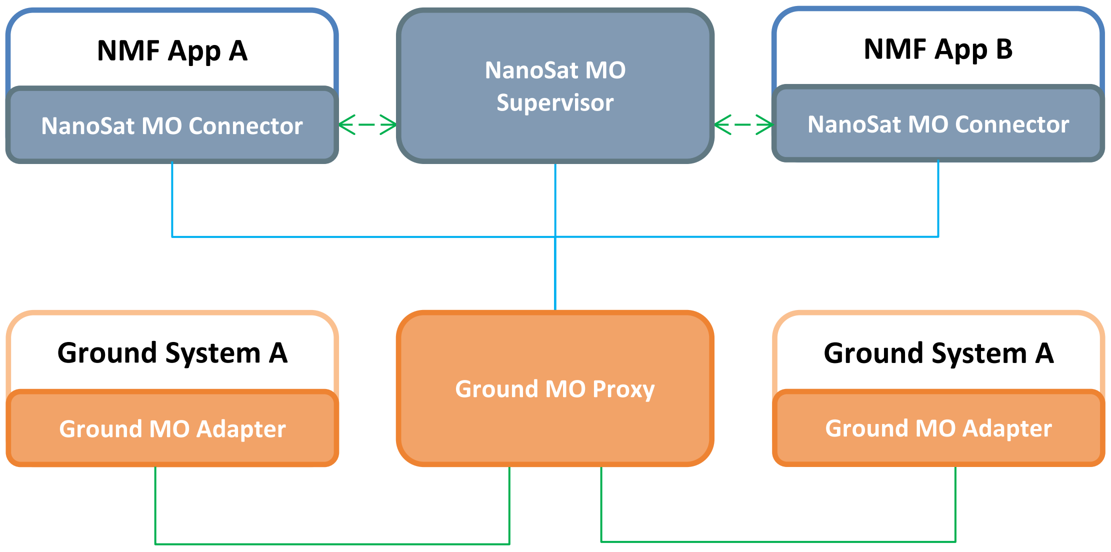


Figure 44: Complex Scenario example.

Chapter 3

3. NANOSAT MO FRAMEWORK: JAVA IMPLEMENTATION

In this chapter, the Java implementation of the NanoSat MO Framework is presented and the most important information is described. This chapter is based on the theory of the NanoSat MO Framework presented in chapter 2.

It can be considered as a reference implementation because it provides a concrete interpretation for the "specifications" presented in chapter 2 and additionally it allowed the discovery of problems, errors and ambiguities in the interfaces.

Software reusability has been taken into account since the first lines were written therefore the implementation is not limited to a specific mission and it can be extended to many missions. The "NMF Core" project in the repository includes generic classes that can be extended by the specific missions while the projects starting in "NMF Mission" are specific to a certain mission. This chapter is focused on the "NMF Core" which includes the implementation of all the services and also of the NMF Composites.

The code of the implementation mentioned in this chapter is not directly present in the dissertation because it is very extensive. However it is available online under ESA's open-source licence. The paragraph reference includes the link to the repository online where the code can be accessed. [57]

3.1. Rationale

The selection of Java as the programming language for the reference implementation of the NanoSat MO Framework might not appear to be the best choice by some individuals. This section presents the reasoning behind this choice.

The first version of Java was released in 1996 and many major releases were released since then. It is a general-purpose computer programming language that is mature, concurrent, class-based, object-oriented, and designed to have as few implementation dependencies as possible. It is intended to let application developers "write once, run anywhere" by using a Java Virtual Machine (JVM) that runs the compiled bytecode of the application. This process can be visualized in Figure 45. [58] [59]

The JVM is an abstract machine for which Java programming language compilers can generate code. Its specification is implemented for specific hardware and software platforms in order to provide the concrete realization of the virtual machine. [59]

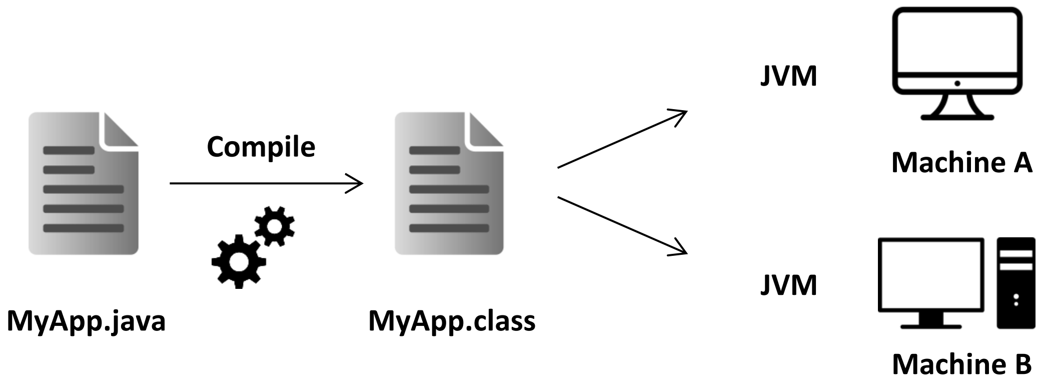


Figure 45: The process of running a Java application in JVMs for two different machines.

The Java portability characteristic mentioned above allows a developer to try and test applications without the need to have the same operating system or system hardware of the spacecraft. Therefore, an NMF App in Java can be developed and tested on the developer’s local machine, then be deployed on an advanced flatsat with the real hardware, and finally be transferred and executed in the actual mission. Section 4.6 presents a local playground environment to test NMF Apps that can be used in different operating systems by taking advantage of Java portability.

In terms of robustness, Java’s memory management model relies on the “new” operator to create objects and there are no explicit programmer-defined pointer data types, no pointer arithmetic, and automatic garbage collection takes care of reclaiming the memory taken by objects that are no longer in use. This memory management model eliminates entire classes of programming errors that usually cause many troubles to C and C++ programmers. [59]

When this research was started in 2014, only the “MO MAL - Java API” book was released and available. The “MO MAL – C++ API” was in the process of being standardized at the same time of this research. There is also a C implementation of the MAL API online developed by CNES however it is not following an official CCSDS standard. [60] [61] [62]

Prior to this research, ESA made available online MO software packages implemented in Java under ESA’s open-source licence. The Java implementation of the NMF uses some of these to maximize code reuse and avoid “reinventing the wheel”. In return, many contributions for improvement of the existing MO software packages were submitted. [63]

The current trend in ESOC’s Ground Data Systems software is to move towards Java. Concrete examples are EGOS User Desktop (EUD), which is ESOC’s next generation of Generic User Interface Framework for Ground Segment Software, and also EGS-CC which selected Java as its preferred programming language for its components. [64] [65]

Java is also popular in the Android developer’s community. It is possible to convert Java bytecode into dex bytecode and then execute it in the Android Runtime (ART) Environment. [66]

The Java programming language seems to be a good choice for the NMF implementation considering the maturity of the technology, timeline of the research, availability of developed software, and its popularity.

3.2. Capitalizing on existing tools, technologies and software

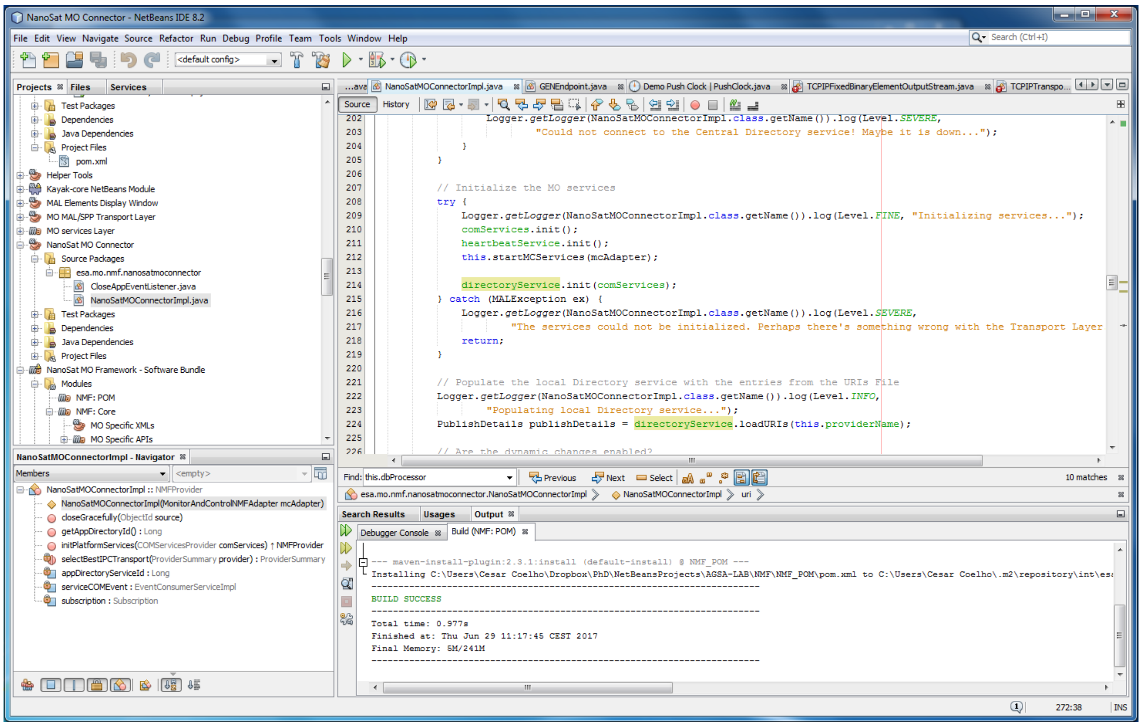
There are tools and systems available that facilitate the orchestration of multiple software components and aid the process of software development. Some of these are presented and put in contrast on how they were used during the implementation process.

ESA's existing MO software was reused in order to avoid developing the same software once again. The NMF's Java implementation takes advantage of multithreading therefore the concept is briefly introduced in a dedicated subsection.

3.2.1. Netbeans

The development of the NMF's Java implementation software was done using Netbeans IDE. An Integrated Development Environment (IDE) normally provides to software developers a software application capable of manipulating source code and usually numerous features that aid the development process, for example, a compiler, a debugger, a profiler, possibility to add external plugins, and others. [67]

The integrated profiling tool allows profiling the CPU, memory, threads, locks and SQL queries as well as basic JVM monitoring, allowing developers to be more productive in solving performance and memory issues. [68]



NanoSat MO Connector - NetBeans IDE 8.2
File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
<default config>
Search (Ctrl+I)
Projects Files Services
Test Packages
Dependencies
Java Dependencies
Project Files
pom.xml
Helper Tools
Kayak-core NetBeans Module
MAL Elements Display Window
MO MAL/SPP Transport Layer
MO services Layer
NanoSat MO Connector
Source Packages
esa.mo.nmf.nanosatmoconnector
CloseAppEventListener.java
NanoSatMOConnectorImpl.java
Test Packages
Dependencies
Java Dependencies
Project Files
NanoSat MO Framework - Software Bundle
Modules
NMF: POM
NMF: Core
MO Specific XMLs
MO Specific APIs
NanoSatMOConnectorImpl - Navigator
Members
<empty>
NanoSatMOConnectorImpl :: NMFProvider
NanoSatMOConnectorImpl(MonitorAndControlNMFAdapter mcAdapter)
closeGracefully(ObjectId source)
getAppDirectoryId() : Long
initPlatformServices(COMServicesProvider comServices) ↑ NMFProvider
selectBestIPCTransport(ProviderSummary provider) : ProviderSummary
appDirectoryServiceId : Long
serviceCOMEvent : EventConsumerServiceImpl
subscription : Subscription
...ava NanoSatMOConnectorImpl.java GENEndpoint.java Demo Push Clock | PushClock.java TCPIPFixedBinaryElementOutputStream.java TCPIPTranspo...
Source History
Find: this.dbProcessor
Previous Next Select
10 matches
esa.mo.nmf.nanosatmoconnector.NanoSatMOConnectorImpl NanoSatMOConnectorImpl uri
Search Results Usages Output
Debugger Console Build (NMF: POM)
272:38 INS

```
                Logger.getLogger(NanoSatMOConnectorImpl.class.getName()).log(Level.SEVERE,
                        "Could not connect to the Central Directory service! Maybe it is down...");
            }
        }

        // Initialize the MO services
        try {
            Logger.getLogger(NanoSatMOConnectorImpl.class.getName()).log(Level.FINE, "Initializing services...");
            comServices.init();
            heartbeatService.init();
            this.startMCServices(mcAdapter);

            directoryService.init(comServices);
        } catch (MALException ex) {
            Logger.getLogger(NanoSatMOConnectorImpl.class.getName()).log(Level.SEVERE,
                    "The services could not be initialized. Perhaps there's something wrong with the Transport Layer
            return;
        }

        // Populate the local Directory service with the entries from the URIs File
        Logger.getLogger(NanoSatMOConnectorImpl.class.getName()).log(Level.INFO,
                "Populating local Directory service...");
        PublishDetails publishDetails = directoryService.loadURIs(this.providerName);

        // Are the dynamic changes enabled?
```

```
--- maven-install-plugin:2.3.1:install (default-install) @ NMF_POM ---
Installing C:\Users\Cesar Coelho\Dropbox\PhD\NetBeansProjects\AGSA-LAB\NMF\NMF_POM\pom.xml to C:\Users\Cesar Coelho\.m2\repository\int\esa
------------------------------------------------------------------------
BUILD SUCCESS
------------------------------------------------------------------------
Total time: 0.977s
Finished at: Thu Jun 29 11:17:45 CEST 2017
Final Memory: 5M/241M
------------------------------------------------------------------------
```

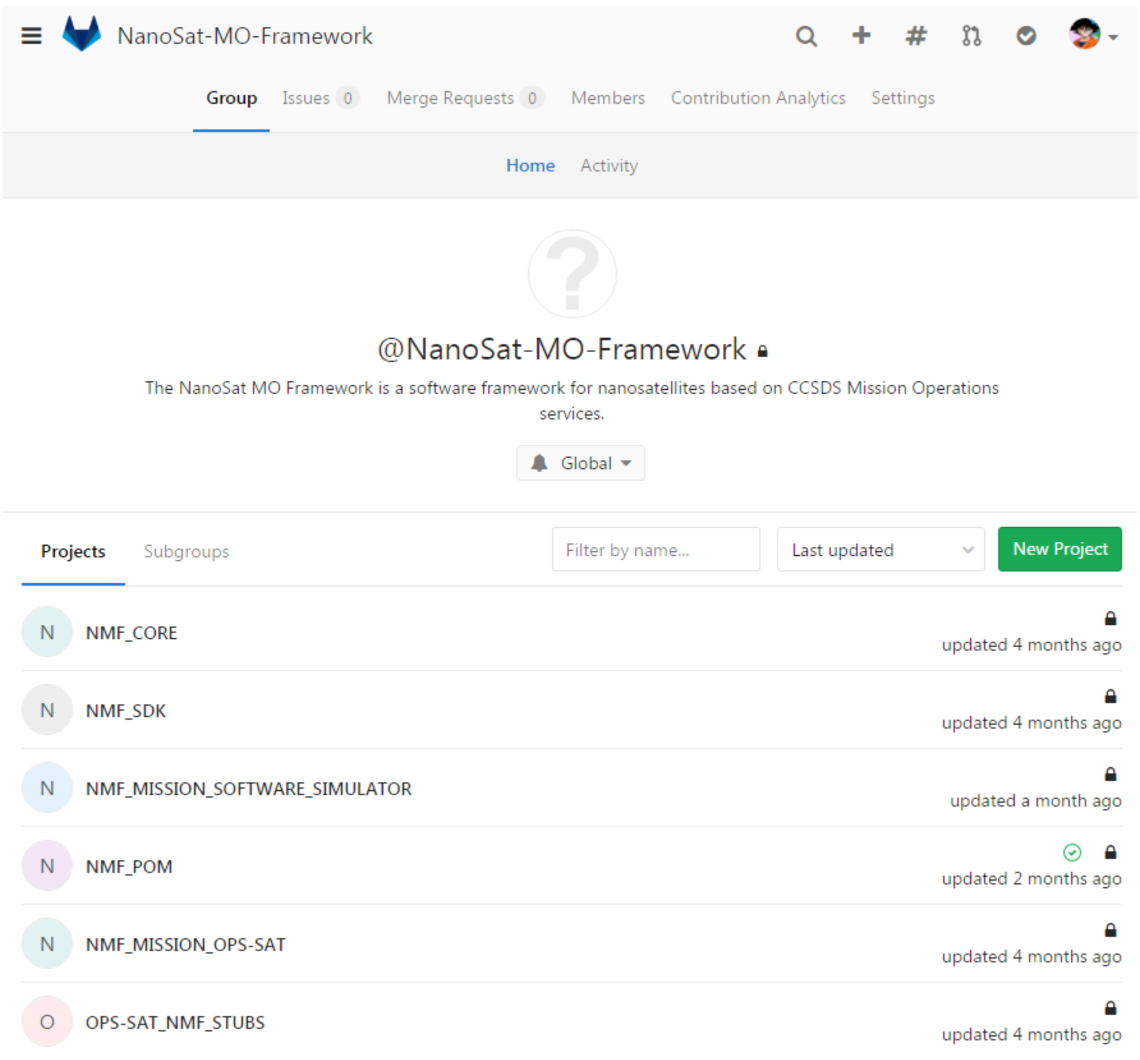
Figure 46: Netbeans IDE screenshot.

3.2.2. Version Control System

Throughout the software development lifecycle, the code grows, evolves, and suffer changes because of bug fixes or new features that are introduced into the codebase. In order to keep track of these changes, a version control system is recommended to be used during the software development process.

Git is a free and open source distributed version control system designed to handle everything from small to very large projects with speed and efficiency. GitLab provides Git repository management via a web-based interface with fine grained access controls, code reviews, issue tracking, activity feeds, wikis, and continuous integration. GitLab was used during NMF's Java implementation on an local server as it is presented in Figure 47. [69] [70]

The first official release of the NMF's Java implementation code was made available online in GitHub together with the NMF Software Development Kit. [57]



@NanoSat-MO-Framework

The NanoSat MO Framework is a software framework for nanosatellites based on CCSDS Mission Operations services.

NMF_CORE	updated 4 months ago
NMF_SDK	updated 4 months ago
NMF_MISSION_SOFTWARE_SIMULATOR	updated a month ago
NMF_POM	updated 2 months ago
NMF_MISSION_OPS-SAT	updated 4 months ago
OPS-SAT_NMF_STUBS	updated 4 months ago

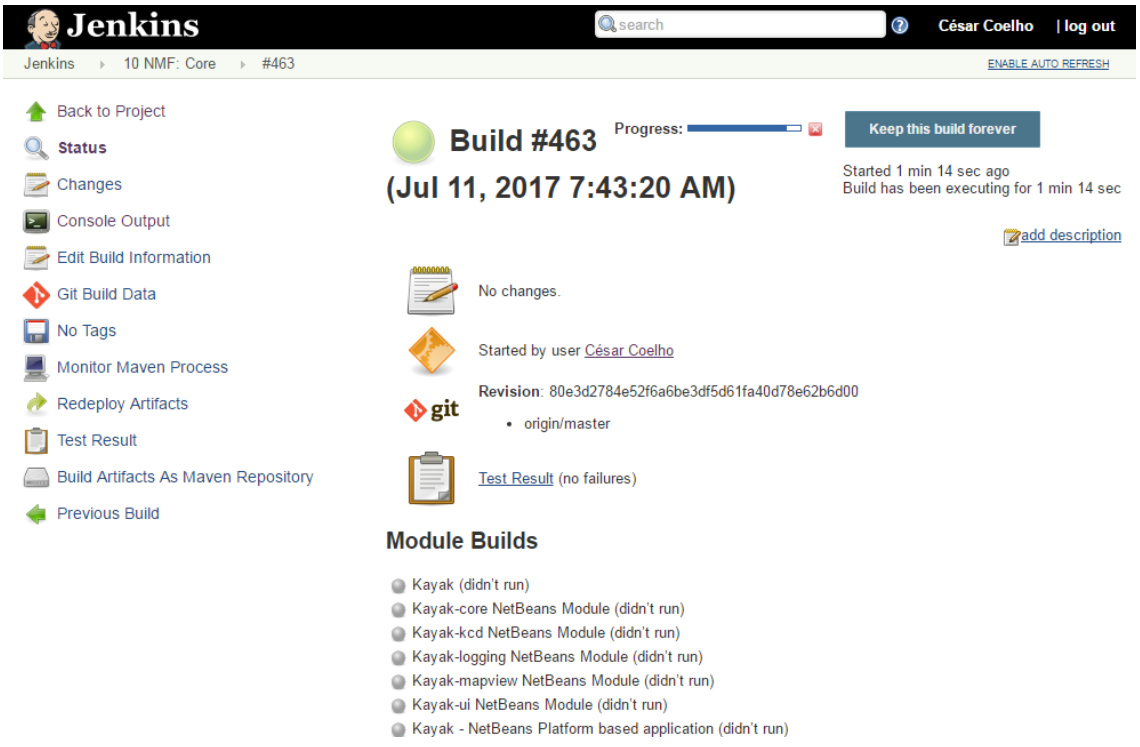
Figure 47: Web-based GitLab displaying the NMF group.

3.2.3. Continuous Integration

Continuous Integration (CI) is a development practice that requires developers to integrate code into a shared repository several times a day. Each check-in is then verified by an automated build, allowing teams to detect problems early. By integrating regularly, you can detect errors quickly, and locate them more easily. [71]

CI was done during the development of the NMF's Java implementation using Jenkins, an open source automation server with hundreds of plugins to support building, deploying and automating any project. [72]

Whenever a commit is pushed into GitLab's NMF Core repository, Jenkins is triggered and then the compilation process happens in Jenkins. If an error occurs, Jenkins can send an email to a predefined set of people.



Build #463 (Jul 11, 2017 7:43:20 AM)

Started 1 min 14 sec ago
Build has been executing for 1 min 14 sec

No changes.

Started by user César Coelho

Revision: 80e3d2784e52f6a6be3df5d61fa40d78e62b6d00

- origin/master

Test Result (no failures)

Module Builds

- Kayak (didn't run)
- Kayak-core NetBeans Module (didn't run)
- Kayak-kcd NetBeans Module (didn't run)
- Kayak-logging NetBeans Module (didn't run)
- Kayak-mapview NetBeans Module (didn't run)
- Kayak-ui NetBeans Module (didn't run)
- Kayak - NetBeans Platform based application (didn't run)

Figure 48: Jenkins during the compilation process of the NMF Core.

3.2.4. Dependencies management

The NMF’s Java implementation was developed in a modular and flexible way. Modularity creates problems during the evolution of software because the resolution of dependencies at compilation time becomes increasingly harder.

Apache Maven is a software project management and comprehension tool. It can manage a project’s build, reporting and documentation from a central piece of information, the Project Object Model (POM). Maven was used in the implementation in order to facilitate the resolution of project dependencies. [73]

The POM is an XML representation of a Maven project held in a file named pom.xml. A project contains configuration files, as well as the developers involved and the roles they play, the defect tracking system, the organization and licenses, the URL of where the project lives, the project's dependencies, and all of the other details that come into play to give code life. [74]

In a multi-module project, a parent POM can be defined by referencing one or more submodules. The parent POM can also define the versions of the different submodules which allows the resolution of dependencies. The NMF’s Java implementation includes a NMF POM project that is extended by all other projects. [75]

3.2.5. MO software reuse

This implementation uses some of the packages previously developed by ESA and made available online on the GitHub platform. There is online, an implementation of the MAL, many transport bindings, the service XML files, the Stub Generator, testbeds, and the MO Graphical Editor.

The MO parent POM project contains the versions of all the projects mentioned above and therefore external projects can extend the MO parent POM to easily select different versions of the MO software. This allows easy migration of the projects from an old version to a new one with a single change of the MO parent POM’s version. When the research was started, the MO parent POM was still at version 1 (released on the 22nd of August 2014) and at the end of 2017, version 7 was being produced and getting ready for release. This implementation is partially responsible for upping the MO parent POM versions because during its development many bugs were discovered, followed by an issue raise and/or a bug fix with for the code to be patched. [74]

The Stub Generator project is developed in form of a maven plugin that can be hooked by other projects for the generation of the code stubs and skeletons of the service from an XML file. Additionally, it can also generate the Interface Control Documentation (ICD) of the service. [75]

The MAL implementation available online is included in the NMF Generic Model which is extended by all other NMF Composites. Thus, the MAL implementation is present in all NMF Composites of the NMF's Java implementation. The MAL implementation allows the selection of the transport binding at runtime. [76]

There is available a generic transport project that includes classes for the handling of data in the transports that are generic to all transports. The RMI transport implementation was originally used at the beginning of the research because it is very simple and it always worked without problems. However, the RMI transport is not an official CCSDS standard and to comply with the NMF Generic Model, the TCP/IP implementation became the default selection.

3.2.6. Multithreading

Multithreading is a widespread programming and execution model that allows multiple threads to exist within the context of a single process. These threads share the process's resources but are able to execute independently. Multithreading is mainly found in multitasking operating systems. [77]

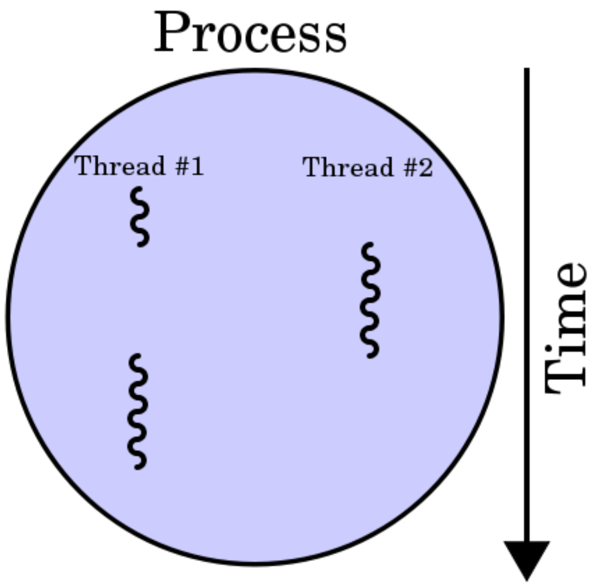


Figure 49: Conceptual visualization of multithreading: A process with two threads of execution, running on a single processor. [78]

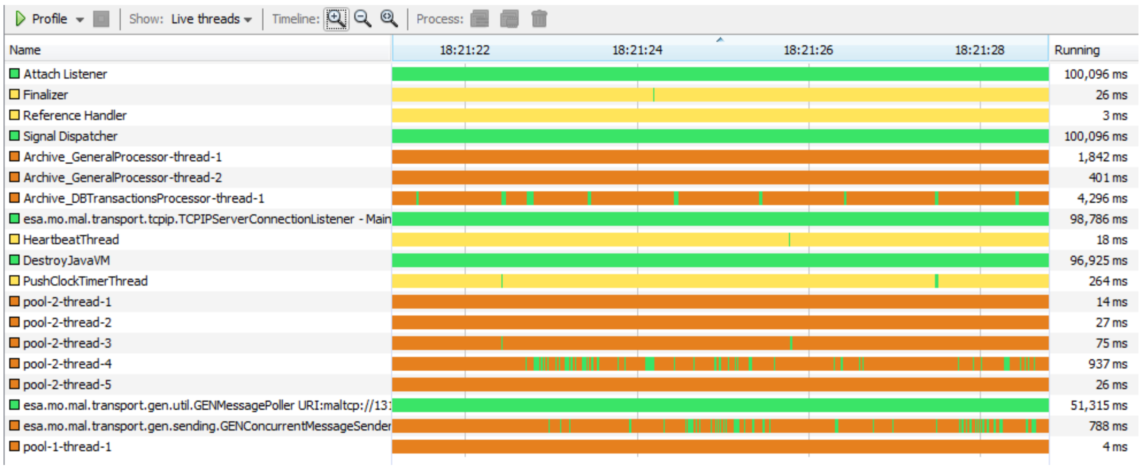
A multi-core processor is a single computing component with two or more independent processing units (called "cores"), which can read and execute program instructions at the same time. This improves the performance of the software because there are fractions of it that can run simultaneously on multiple cores without affecting the execution of the other fractions. [79] [80]

The main challenge in designing concurrent applications is concurrency control because ensuring the correct sequence of interactions or communications between different computational executions, and coordinating access to resources that are shared among executions is not so simple. Frequently, problems such as race conditions, deadlocks, starvation, and livelock arise from developing concurrent applications. [81]

Java programming language and the JVM were designed to support concurrent applications by allowing many threads of execution at once. These threads independently execute code that operates on values and objects residing in a shared main memory. Threads may be supported by having many hardware processors, by time-slicing a single hardware processor, or by time-slicing many hardware processors. The developer must ensure that read and write access to objects is properly synchronized between threads in order to guarantee that objects are modified by only one thread at a time and that threads are prevented from accessing partially updated objects during modification by another thread. The Java language includes a set of utility classes to support this coordination for concurrent programming. [82] [83]

The NMF's Java implementation takes advantage of the Java concurrent classes for parallelizing the software execution. For example: the Generic Transport includes an executor with multiple threads that handles the interactions between the transport binding and the services; the Archive service implementation includes two executors to increase its performance; the actions triggered from the Action service are executed in separate threads because this make a call to the layer above; if a configuration of a service is changed, the configuration isn't updated in the Archive immediately, instead it is delegated to a dedicated thread in order to allow the service to remain responsive.

Figure 50 presents a visual representation of all threads in a dummy NMF App in function of time. A few stores and queries were executed during the presented time period to be possible to visualize in green, the interactions with the COM Archive and transport binding when those threads are running tasks.



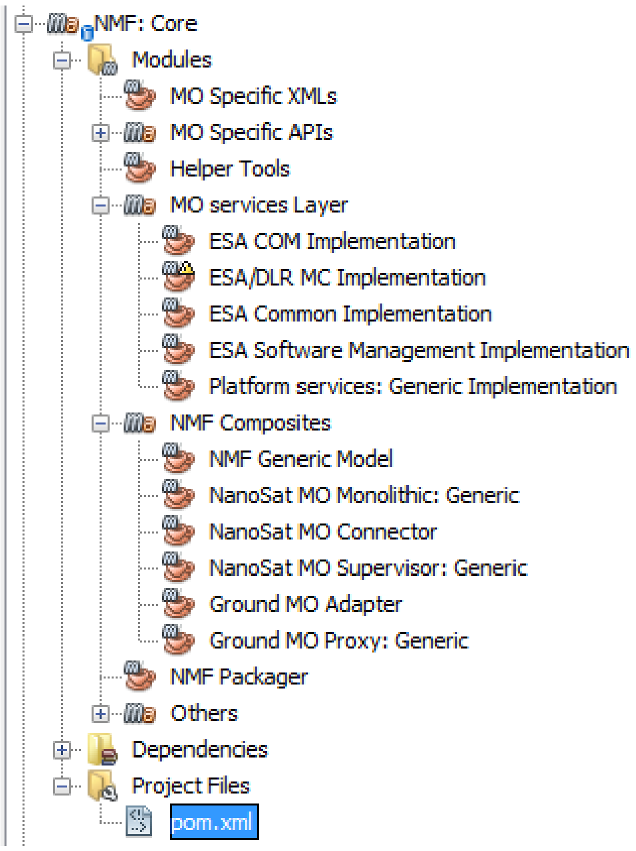
Name	Running
Attach Listener	100,096 ms
Finalizer	26 ms
Reference Handler	3 ms
Signal Dispatcher	100,096 ms
Archive_GeneralProcessor-thread-1	1,842 ms
Archive_GeneralProcessor-thread-2	401 ms
Archive_DBTransactionsProcessor-thread-1	4,296 ms
esa.mo.mal.transport.tcpip.TCPIPServerConnectionListener - Main	98,786 ms
HeartbeatThread	18 ms
DestroyJavaVM	96,925 ms
PushClockTimerThread	264 ms
pool-2-thread-1	14 ms
pool-2-thread-2	27 ms
pool-2-thread-3	75 ms
pool-2-thread-4	937 ms
pool-2-thread-5	26 ms
esa.mo.mal.transport.gen.util.GENMessagePoller URI:maltcp://13	51,315 ms
esa.mo.mal.transport.gen.sending.GENConcurrentMessageSender	788 ms
pool-1-thread-1	4 ms

Figure 50: Netbeans profiler displaying parallel threads' running.

3.3. Implementation

This section presents a brief explanation of the considerations taken during the implementation process and a description of the implementation details for each package.

Section 2.1.2 presents the three different problem domains that exist in the NMF: Ground development; Mission development; and App development. As a reference implementation, one must be able to use it across all these domains therefore the functionality that is common to all domains was implemented in a project named "NMF: Core". This includes all the service interfaces (APIs) and respective implementations for the consumer and provider side, the NMF Composites, and the implementation of the NMF Package concept. Figure 51 shows the NMF Core implementation package.



```
NMF: Core
    Modules
        MO Specific XMLs
        MO Specific APIs
        Helper Tools
        MO services Layer
            ESA COM Implementation
            ESA/DLR MC Implementation
            ESA Common Implementation
            ESA Software Management Implementation
            Platform services: Generic Implementation
        NMF Composites
            NMF Generic Model
            NanoSat MO Monolithic: Generic
            NanoSat MO Connector
            NanoSat MO Supervisor: Generic
            Ground MO Adapter
            Ground MO Proxy: Generic
        NMF Packager
        Others
    Dependencies
    Project Files
        pom.xml
```

Figure 51: NMF Core "mavenized" multi-module project presented in Netbeans' Projects window.

3.3.1. Implementation Considerations

Some considerations had to be kept in mind during the decision processes that were performed as these drove the implementation and its necessary optimizations. The considerations taken into account were:

- Target platform
- Fallacies of distributed computing [84]
- Lightweight NMF Apps

First of all, it is necessary to understand that this implementation was targeted to run on OPS-SAT's experimental platform which features a dual-core ARM Cortex A9 processor capable of running Linux and Java. Chapter 1 mentions that since 2017, the Nanomind Z7000, a radiation-hardened on-board computers with similar characteristics to OPS-SAT's experimental platform is commercially available and therefore it is expected that devices capable of running NMF's Java implementation become more common in nanosatellites over the upcoming years. [16]

In a final setup, the NMF Composites are deployed in at least 2 different computing nodes because we have the NanoSat and Ground segments that are separate. Thus, it is important to be aware of the 8 fallacies of distributed computing not only when deploying the components but also during the implementation phase: [84]

1. The network is reliable.
2. Latency is zero.
3. Bandwidth is infinite.
4. The network is secure.
5. Topology doesn't change.
6. There is one administrator.
7. Transport cost is zero.
8. The network is homogeneous.

Given the fact that this implementation is envisaged to be used in a "multiple applications" scenario, the NanoSat MO Connector was implemented and optimized to be lightweight while running on the target platform. Additionally, each NMF App has its own COM Archive that is used by almost all the other services therefore high priority was given for the optimization of the COM Archive implementation. Section 6.3 includes some analysis and metrics.

3.3.2. COM services

The implementation of the COM services includes all the 3 services: Activity Tracking, Event and Archive service.

The Activity Tracking service does not include any operations because it relies on the Event service for emitting the Activity Tracking COM Events. Therefore, the implementation of the service simply includes methods to allow the release of Activity Tracking COM Events when the provider needs to.

The Event service includes a single operation on its service interface that is used to emit COM Events of other services. In this implementation there are a set of methods to generate, store and publish COM Events that can be used by other services.

The Archive service implementation data is managed using a relational database management system that is based on Structured Query Language (SQL). Additionally, it uses the Java Persistence API (JPA) specification for the description of the relational data's management. [85] [86]

The Archive service interface includes a set of operations to retrieve, query, count, store, update, and delete. This interface is connected to a database transactions processor responsible for holding a queue of transactions to be performed with the actual database in a sequentially ordered manner. The selected implementation of JPA was EclipseLink and it uses the SQLite JDBC Driver implementation for accessing and creating SQLite database files. [87] [88]

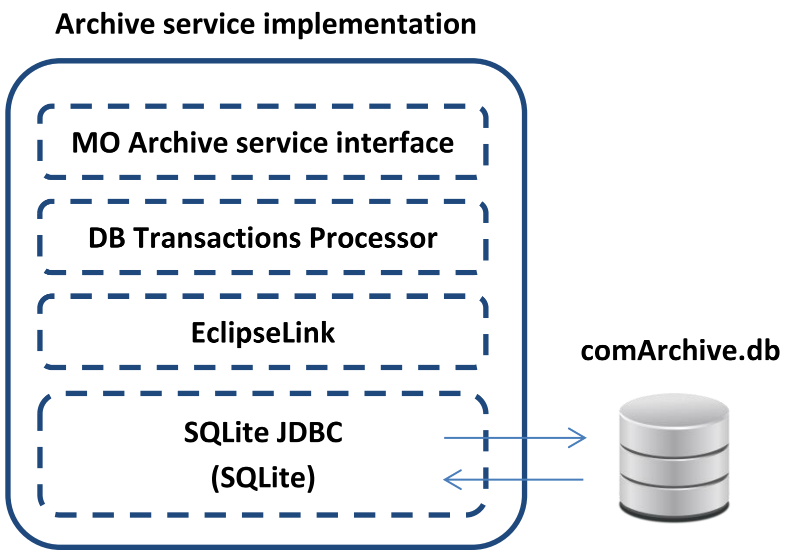
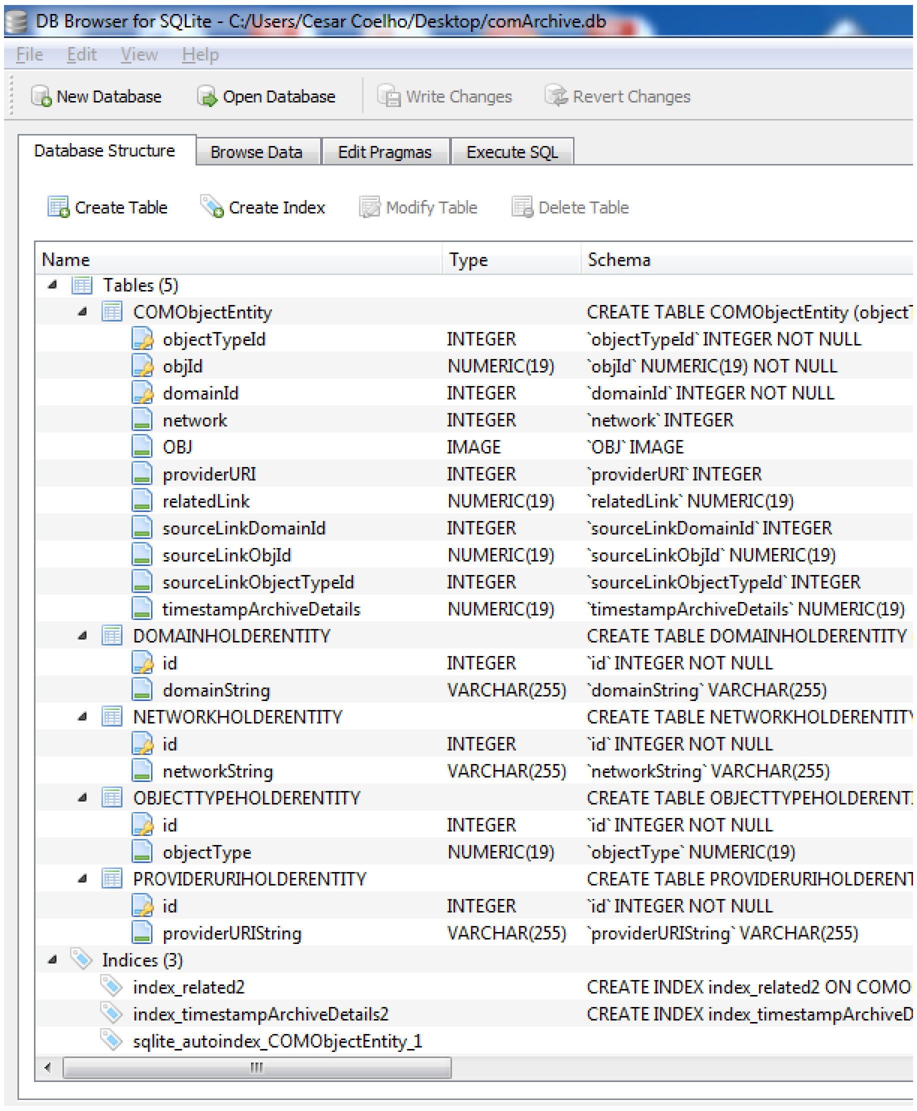

Archive service implementation
MO Archive service interface
DB Transactions Processor
EclipseLink
SQLite JDBC
(SQLite)
comArchive.db

Figure 52: Archive service provider implementation diagram.

There are optimizations on the MO Archive service interface layer. For example, the generation of the object instance identifier is actually faster than if a value is assigned directly because the service interface layer does not have to check in the database if it exists.

SQLite is an in-process library that implements a self-contained, serverless, zero-configuration, transactional SQL database engine. SQLite is an embedded SQL database engine which means that unlike others, it does not have a separate server process. SQLite reads and writes directly to ordinary disk files, and for example, a complete SQL database with multiple tables, indices, triggers, and views, is contained in a single disk file. For the Archive service implementation, the filename is “comArchive.db”. [89]

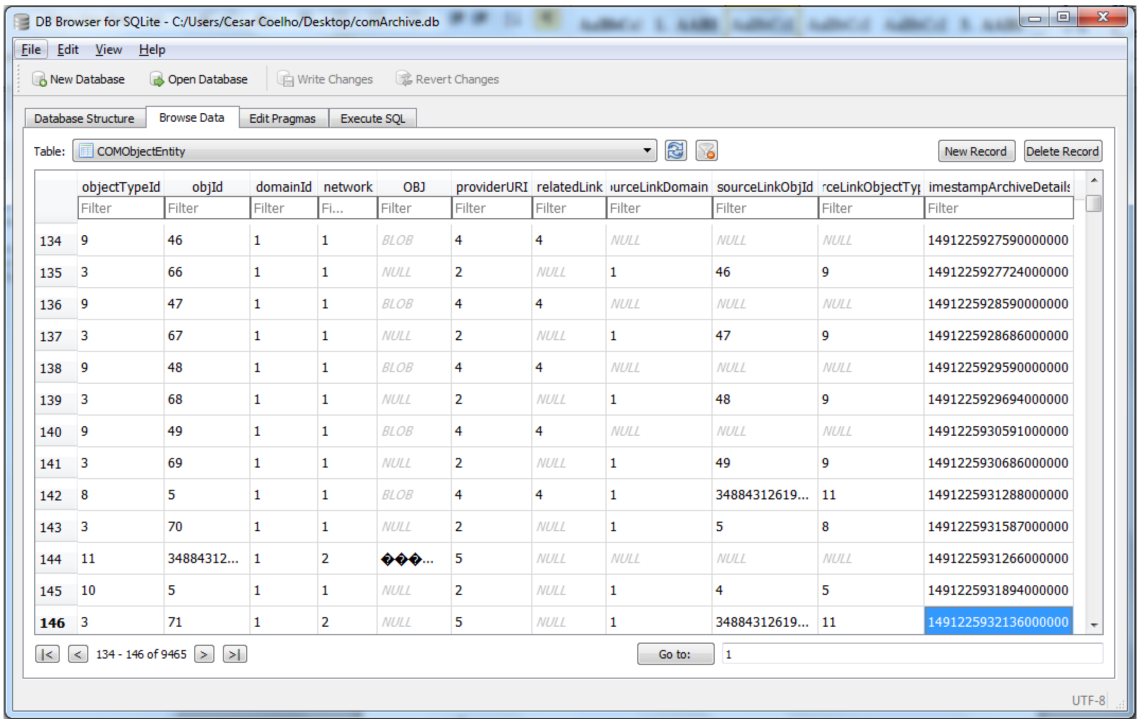
The database structure includes 5 tables and 3 indexes like Figure 53 presents.



Name	Type	Schema
Tables (5)		
COMObjectEntity		CREATE TABLE COMObjectEntity (object
objectTypeId	INTEGER	\`objectTypeId\` INTEGER NOT NULL
objId	NUMERIC(19)	\`objId\` NUMERIC(19) NOT NULL
domainId	INTEGER	\`domainId\` INTEGER NOT NULL
network	INTEGER	\`network\` INTEGER
OBJ	IMAGE	\`OBJ\` IMAGE
providerURI	INTEGER	\`providerURI\` INTEGER
relatedLink	NUMERIC(19)	\`relatedLink\` NUMERIC(19)
sourceLinkDomainId	INTEGER	\`sourceLinkDomainId\` INTEGER
sourceLinkObjId	NUMERIC(19)	\`sourceLinkObjId\` NUMERIC(19)
sourceLinkObjectTypeId	INTEGER	\`sourceLinkObjectTypeId\` INTEGER
timestampArchiveDetails	NUMERIC(19)	\`timestampArchiveDetails\` NUMERIC(19)
DOMAINHOLDERENTITY		CREATE TABLE DOMAINHOLDERENTITY
id	INTEGER	\`id\` INTEGER NOT NULL
domainString	VARCHAR(255)	\`domainString\` VARCHAR(255)
NETWORKHOLDERENTITY		CREATE TABLE NETWORKHOLDERENTIT
id	INTEGER	\`id\` INTEGER NOT NULL
networkString	VARCHAR(255)	\`networkString\` VARCHAR(255)
OBJECTTYPEHOLDERENTITY		CREATE TABLE OBJECTTYPEHOLDERENT
id	INTEGER	\`id\` INTEGER NOT NULL
objectType	NUMERIC(19)	\`objectType\` NUMERIC(19)
PROVIDERURIHOLDERENTITY		CREATE TABLE PROVIDERURIHOLDERENT
id	INTEGER	\`id\` INTEGER NOT NULL
providerURIString	VARCHAR(255)	\`providerURIString\` VARCHAR(255)
Indices (3)		
index_related2		CREATE INDEX index_related2 ON COMO
index_timestampArchiveDetails2		CREATE INDEX index_timestampArchiveD
sqlite_autoindex_COMObjectEntity_1		

Figure 53: Archive service database tables displayed in DB Browser for SQLite. [90]

The COMObjectEntity table holds the COM Objects of the Archive service implementation. The table’s primary key is composed by the fields: objectTypeId, objId, and domainId. It includes other fields such as network, OBJ, providerURI, relatedLink, sourceLinkDomainId, sourceLinkObjId, sourceLinkObjectTypeId, timestampArchiveDetails. Some of these are linking to the remaining tables mentioned before.



	objectTypeId	objId	domainId	network	OBJ	providerURI	relatedLink	ırceLinkDomain	sourceLinkObjId	rceLinkObjectTy|	imestampArchiveDetails
	Filter	Filter	Filter	Fi...	Filter	Filter	Filter	Filter	Filter	Filter	Filter
134	9	46	1	1	BLOB	4	4	NULL	NULL	NULL	1491225927590000000
135	3	66	1	1	NULL	2	NULL	1	46	9	1491225927724000000
136	9	47	1	1	BLOB	4	4	NULL	NULL	NULL	1491225928590000000
137	3	67	1	1	NULL	2	NULL	1	47	9	1491225928686000000
138	9	48	1	1	BLOB	4	4	NULL	NULL	NULL	1491225929590000000
139	3	68	1	1	NULL	2	NULL	1	48	9	1491225929694000000
140	9	49	1	1	BLOB	4	4	NULL	NULL	NULL	1491225930591000000
141	3	69	1	1	NULL	2	NULL	1	49	9	1491225930686000000
142	8	5	1	1	BLOB	4	4	1	34884312619...	11	1491225931288000000
143	3	70	1	1	NULL	2	NULL	1	5	8	1491225931587000000
144	11	34884312...	1	2	���...	5	NULL	NULL	NULL	NULL	1491225931266000000
145	10	5	1	1	NULL	2	NULL	1	4	5	1491225931894000000
146	3	71	1	2	NULL	5	NULL	1	34884312619...	11	1491225932136000000

Figure 54: Example of the content in a COMObjectEntity table displayed in the application DB Browser for SQLite. [90]

The COM services were the first services to be implemented. Optimizing the COM Archive implementation was set as high priority and therefore it has been continuously improved throughout the research and development. Originally, the backend was using the embedded Apache Derby driver but this was later changed to SQLite in order attempt to improve performance. Additionally, the "database transactions processor" was included to decouple the service interface calls from the actual insertions into the database by having a single thread executor dedicated to execute the transaction with the database. This makes the store operation to be much faster during runtime because the call will not block waiting for the actual insertion in the database to be completed. Additionally, another executor with 2 fixed threads was included in order to speed up the initialization process of the Archive service and to handle the generation of COM Events of the Archive service. The Archive service was also extended to support a special PaginationFilter for its queries.

A bespoke service was included in this implementation for the synchronization of the COM Archive between space and ground. This is not originally part of the standard.

More information about performance of the COM Archive service implementation is presented in section 6.3.

3.3.3. Common services

The implementation of the Common services includes all the 3 services: Configuration service, Directory service, and Login service.

The implementation of the Common services was done before there was an official CCSDS standard of the services. Throughout the development many problems were found and these were directly fed back into the draft version. There was also a strong collaboration in evolving the Configuration service from a single operation service with implementation/specific Configurations into something more complex which includes a concept itself of a “Configuration” in MO.

The NMF Composites do not instantiate the Configuration service itself however they strongly rely on its COM objects data model in order to store the state of other services. This is very useful because upon start up, it allows restoring the previous state of the service.

The Directory service plays an important role in the NMF because the provider side can be found in 3 different places of the system: in the NMF App itself, in the NanoSat MO Supervisor as Central Directory service, and in the Ground MO Proxy as a replica of the Central Directory service. The Directory service implementation includes all its operations except for the getServiceXML operation. [46]

The Login service implementation was developed by a student participating in ESA’s Summer of Code in Space (SOCIS). This implementation uses Apache Shiro and the MAL Access Control mechanism for handling the logins. [91] [92]

3.3.4. M&C services

The implementation of the Monitor and Control (M&C) services includes all the 6 main services and the 2 auxiliary services.

Each service contains dedicated managers in order to manage the existing definitions in the service. They have small differences among them although they all support adding, removing, or updating definitions.

Besides the functionality mentioned above, the Parameter Manager also includes the possibility of using an adapter that connects to another entity of software in order to link the getters and setters to interact with parameter values. Instead of having a fixed list of parameters, the adapter allows the developer to have any combination of parameters and shifts the responsibility of linking a parameter name to an actual value to the developer implementing the adapter. This is possible by using the Adapter pattern which is represented in Figure 55. [1]

For the Action service, there is an Action Manager that allows the management of the definitions and also allows the possibility of submitting an adapter to dispatch the actions. It is up to the implementer of the adapter how the actions are dispatched and linked to an actual activity. The Archive service allows the reporting of the different progress stages of the action.

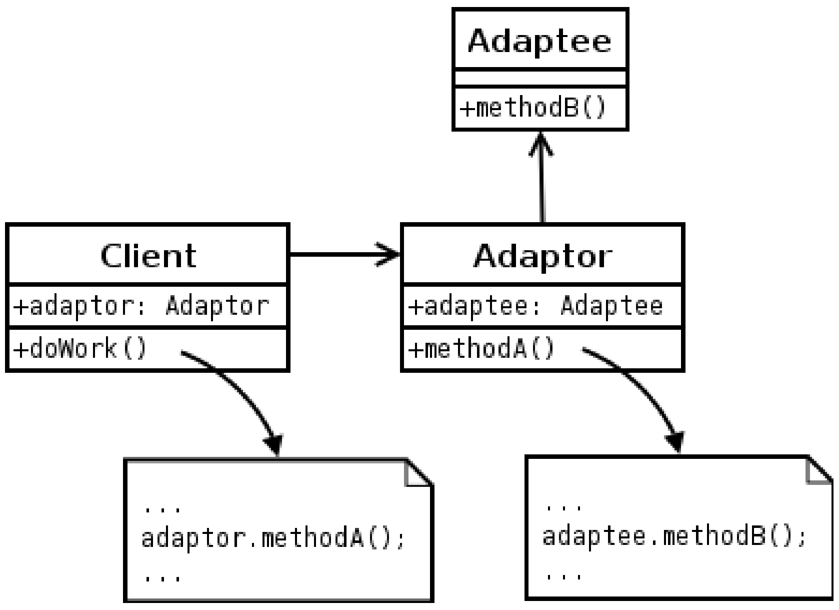


Figure 55: Adapter pattern. [93]

The Managers have interactions with the Archive service in order to store and retrieve the definitions. Additionally, each service has a certain configuration that can be restored at any time by the provider. For the NMF's Java implementation, this occurs during start up.

The implementation of the services was started when the 3rd revision of the M&C services specification was on-going. It was developed in collaboration with DLR and found many problems with the specification which were later fixed. Both the specification and implementation had to be updated to cope with these changes.

3.3.5. Platform services implementation

The implementation of the Platform services includes all the services with a “thread-safe” design as they are expected to be consumed by multiple NMF Apps simultaneously. For the consumer side, all the services’ stubs have been made available while for the provider, the skeletons of the services were implemented however without any specific backend. Just like in the M&C services, the Adapter pattern was used. [1]

The Adapter pattern allows an adapter to be submitted to the service and therefore different missions can reuse the same implementation of the service while the adapter holds just the mission-specific logic to interact with the actual unit. An example for the GPS service can be visualized in Figure 56.

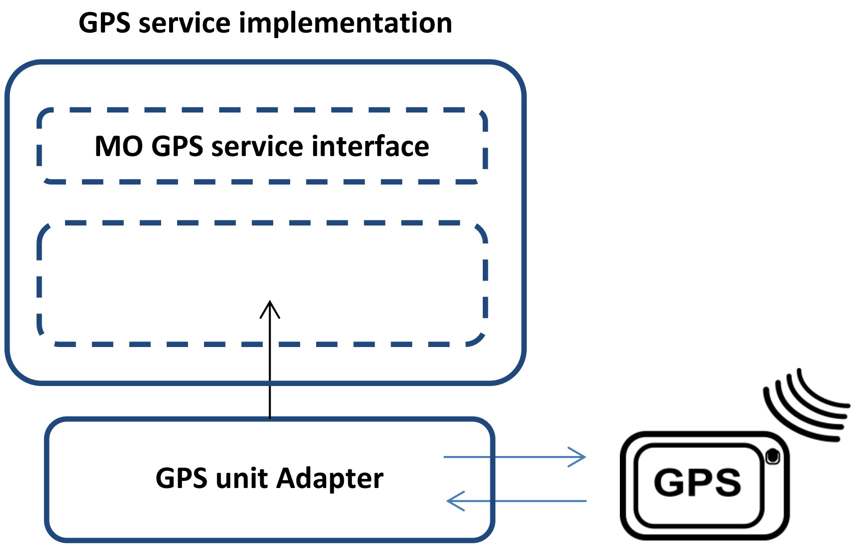


Figure 56: GPS service implementation using the Adapter pattern.

The Adapter pattern used in the Platform services implementation also allows possibility of integration with an Electronic Data Sheet (EDS) device by having an adapter making calls on the stubs generated from the EDS. Such integration is presented in more detail in section 7.3.1. [121]

3.3.6. Software Management services

The implementation of the Software Management services includes 3 services: Heartbeat service, Apps Launcher service, and Package Management service.

The Heartbeat service is very simple and was implemented with a Timer set to publish a heartbeat message every 10 seconds.

The Apps Launcher service scans the specified folder for new applications every time the list of applications is requested or if an application is started, stopped, or killed. When an application is started, the implementation starts a new process and passes as parameters the first text line of the file runAppLin.sh or runAppWin.bat depending on the operating system where the service is running. This allows the same service to operate both in a Linux and Windows operating system.

The mechanism mentioned above is not limited to NMF Apps and can be used to start scripts or any other applications. The service is starting a new process per application and each of them has its own memory space. The service does not allow running multiple instances of the same application. This was enforced for simplicity reasons.

If the service is to be used to start "untrusted" applications, this procedure should be further improved by running each process inside a sandbox environment. This approach is further explained in chapter 7.

The Package Management service was implemented using the Adapter pattern in order to be possible to exchange the backend with different types of package management systems. As part of this implementation there's a dedicated package defined and implemented. The NMF Package implementation is presented in section 2.7.3.

The set of Software Management services' specification presented in Chapter 2 are good candidates for future standardization of the CCSDS Software Management services. This implementation could be reused as a prototype for them.

3.3.7. NMF Composites

This section describes all the NMF Composites and how they are derived from the NMF Generic Model implementation.

Beside the NMF Generic Model implementation package, there are also the implementation packages of the NMF Composites: “NanoSat MO Monolithic: Generic”, “NanoSat MO Connector”, “NanoSat MO Supervisor: Generic”, “Ground MO Adapter”, and “Ground MO Proxy: Generic”. Some of these implementations are “Generic” because they can be extended for the mission that is using them.

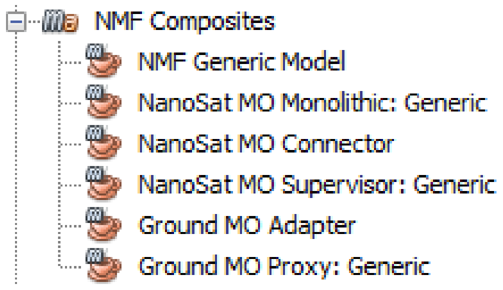

NMF Composites
- NMF Generic Model
- NanoSat MO Monolithic: Generic
- NanoSat MO Connector
- NanoSat MO Supervisor: Generic
- Ground MO Adapter
- Ground MO Proxy: Generic

Figure 57: NMF Composites implementation.

The NMF Generic Model implementation is a project that includes all the MO services implementation packages that were presented on chapter 2 and also includes the MAL implementation and the TCP/IP transport binding implementation as dependencies. Figure 58 presents these dependencies.

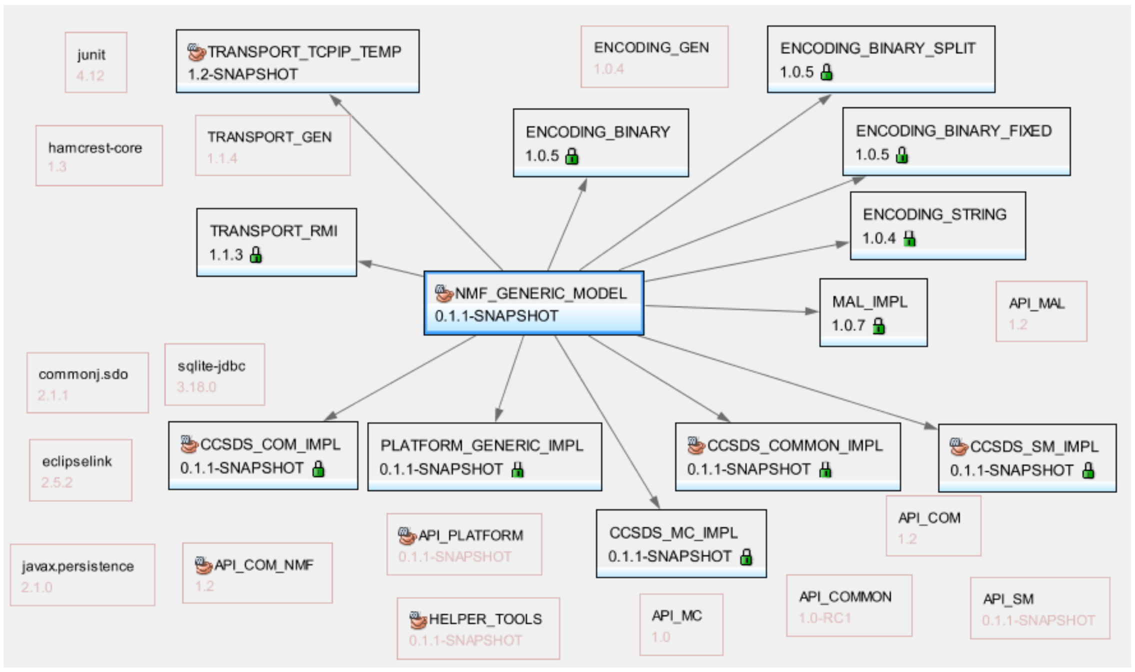

junit 4.12
TRANSPORT_TCPIP_TEMP 1.2-SNAPSHOT
ENCODING_GEN 1.0.4
ENCODING_BINARY_SPLIT 1.0.5
hamcrest-core 1.3
TRANSPORT_GEN 1.1.4
ENCODING_BINARY 1.0.5
ENCODING_BINARY_FIXED 1.0.5
TRANSPORT_RMI 1.1.3
ENCODING_STRING 1.0.4
NMF_GENERIC_MODEL 0.1.1-SNAPSHOT
MAL_IMPL 1.0.7
API_MAL 1.2
commonj.sdo 2.1.1
sqlite-jdbc 3.18.0
eclipselink 2.5.2
CCSDS_COM_IMPL 0.1.1-SNAPSHOT
PLATFORM_GENERIC_IMPL 0.1.1-SNAPSHOT
CCSDS_COMMON_IMPL 0.1.1-SNAPSHOT
CCSDS_SM_IMPL 0.1.1-SNAPSHOT
API_PLATFORM 0.1.1-SNAPSHOT
CCSDS_MC_IMPL 0.1.1-SNAPSHOT
API_COM 1.2
javax.persistence 2.1.0
API_COM_NMF 1.2
HELPER_TOOLS 0.1.1-SNAPSHOT
API_MC 1.0
API_COMMON 1.0-RC1
API_SM 0.1.1-SNAPSHOT

Figure 58: NMF Generic Model implementation dependencies.

The NMF Generic Model implementation includes an abstract class, the NMFProvider that is extended by other classes such as the NanoSatMOConnectorImpl, NanoSatMOSupervisor, and NanoSatMOMonolithic. The NMFProvider class implements the NMFInterface which allows the application logic to be always exposed to the same methods for accessing the 5 services' sets: COM, M&C, Common, Platform, and Software Management services. It also allows having a common interface to push parameter values, publish alert events or report the execution progress of an action instances.

All the 3 classes that extend the NMFProvider are initialized taking as argument the MonitorAndControlNMFAdapter class. It needs to be extended by the NMF App developer in the case of the NanoSatMOConnectorImpl, or by the NMF Mission developer in the case of NanoSatMOSupervisor and NanoSatMOMonolithic. This class is where the developers add the "glue" logic to connect the retrieval and set of parameters to the application logic and the dispatch of actions into concrete method calls. It also allows an implementer at start up to register the actions, alerts, parameters, and aggregations definitions.

A MonitorAndControlNMFAdapter that is used for an NMF App can also be used for the NanoSatMOMonolithic. This allows the App logic to be directly executed on top of the NanoSat MO Monolithic. The advantage of doing this is that it is possible to debug the NMF App logic using a simulator for the mission. Section 4 includes an example where this was done for a Monolithic Provider demo.

After developing and debugging an app with the Monolithic architecture, swapping it back to use the App architecture with the NanoSat MO Connector won't take much effort because the interfaces towards the app developer remains the exact same.

A generic implementation of the NanoSat MO Supervisor and NanoSat MO Monolithic must be extended for the specific platform of the mission because the peripherals available on the platform can be different between spacecraft. In that case, different implementations of the Platform services and different transport layers for the communication with ground might need to be developed. The generic nature of the design is flexible enough to accommodate this need.

The Ground MO Adapter was implemented and it includes the consumer stubs to all the services that are present in the NMF Generic Model which allows this component to connect to MO providers. It also facilitates the retrieval of a list of providers from a remote Directory service. After having this list, one can select one provider of the list and instantiate the Ground MO Adapter by passing the provider connection details.

The implementation of the Ground MO Proxy was implemented in a generic way to support mission-specific extensions. It includes a local Directory service that “mimics” the Central Directory service available on the NanoSat segment. It uses the Ground MO Adapter to connect to the Central Directory service and retrieve its information. It also includes a local Archive service to cache the information from the NMF Apps.

A dedicated SPP protocol bridge was implemented because it needs to do address translation from SPP to another transport and vice-versa. The MAL-SPP binding does not support routing natively as this would increase the size of the Space Packets and therefore address translation needs to be done.

3.3.8. Integration of NMF Package concept

In order to allow consistent distribution of apps between different nanosatellite platforms a package file format was defined. This package file format was inspired by the current packages used by popular package managers and it was implemented according to the specifications presented in chapter 2.

An adapter for the Package Management service was implemented for the use of NMF Packages with the service. This adapter is able to install, uninstall, and update packages by creating, deleting and updating folders in the file system.

This implementation is done for version 1 of the NMF Package’s descriptor. On a future version, more metadata could be included in the descriptor to provide more information about the content. Section 7.1 provides some suggestions.

The implementation could be also improved by creating a maven plugin hook for the automatic generation of NMF Packages at compilation time. This would dramatically facilitate the generation of NMF Packages for developers using the SDK.

3.4. Generic Deployment

The deployment of the NMF's Java implementation can be done by using the NMF Library concept explained in section 2.7.5. The NMF Library needs to be created for a specific mission and allows NMF Apps to be transferred to the spacecraft without the overhead of the whole framework implementation. This generic deployment is only valid for the multiple applications' way of deploying software and not for the monolithic deployment.

The Generic deployment of the NMF's Java implementation consists of a very simple folders structure that includes just 2 folders: apps; libs.

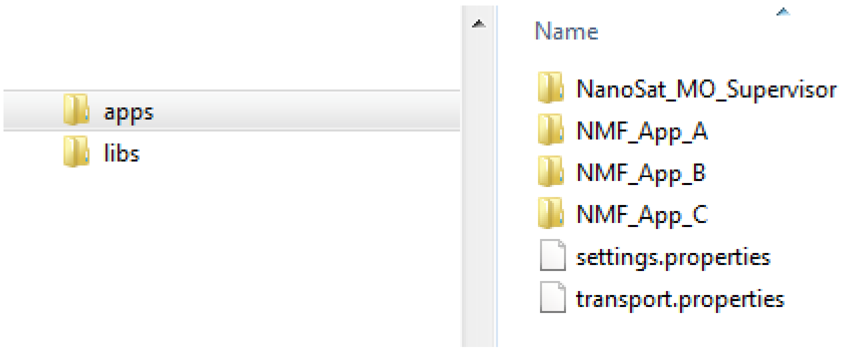
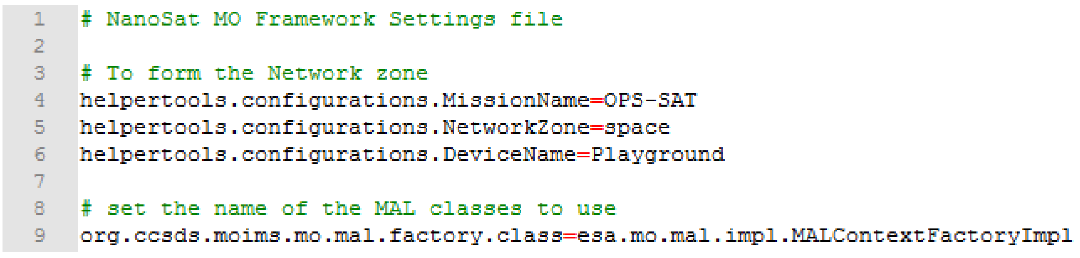


Figure 59: Example of the generic deployment folder structure.

The apps folder shall hold 2 properties files and additionally, folders where each of them corresponds to an application. The settings.properties includes mission-specific information that allows an NMF App to identify where it is running, while the transport.properties includes the properties to select the transport and their respective transport configurations. An example of the settings.properties is presented in Figure 60 for the NMF Playground.

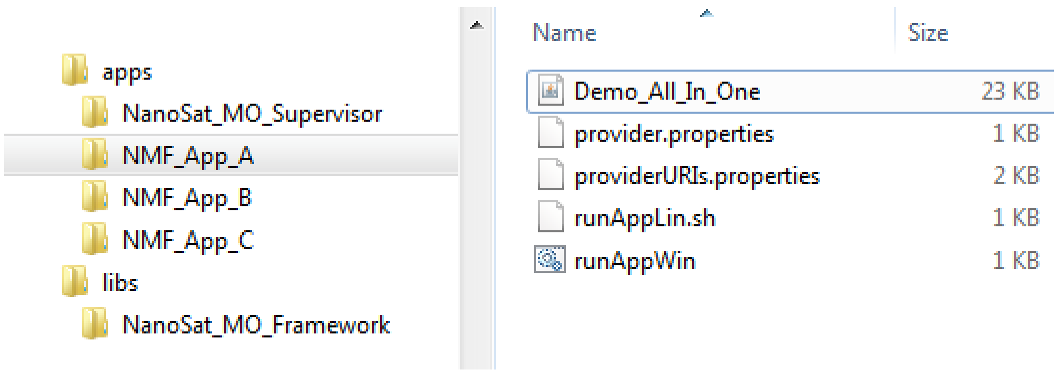


```
# NanoSat MO Framework Settings file

# To form the Network zone
helpertools.configurations.MissionName=OPS-SAT
helpertools.configurations.NetworkZone=space
helpertools.configurations.DeviceName=Playground

# set the name of the MAL classes to use
org.ccsds.moims.mo.mal.factory.class=esa.mo.mal.impl.MALContextFactoryImpl
```

Figure 60: Example of the content of the settings.properties file.



- apps
  - NanoSat_MO_Supervisor
  - NMF_App_A
  - NMF_App_B
  - NMF_App_C
- libs
  - NanoSat_MO_Framework

Name	Size
Demo_All_In_One	23 KB
provider.properties	1 KB
providerURIs.properties	2 KB
runAppLin.sh	1 KB
runAppWin	1 KB

Figure 61: Generic deployment example.

The libs folder shall hold folders where each of them corresponds to a library for other applications to use. Figure 61 presents an example with 4 applications and the NanoSat MO Framework library.

It is also possible to visualize from the example presented in Figure 61 that the simple Demo_All_In_One application has a size of just 23 KB. The NMF App only depends on the NanoSat_MO_Connector artifact which is not part of the Demo_All_In_One jar file but it is available in the library that is on the NanoSat_MO_Framework folder.

```
java -classpath "Demo_All_In_One.jar;.\..\..\libs\NanoSat_MO_Framework\*" esa.mo.nmf.apps.AllInOne
```

Figure 62: Example of the content in the runAppWin.bat file.

Figure 62 presents the content of the runAppWin.bat file with the command to start the Demo_All_In_One application. The command sets the classpath to both the application and to all files inside the NanoSat_MO_Framework library folder. The runAppLin.sh file includes a similar command.

3.5. Reuse in other ESA Projects

Some of the packages of the NMF's Java implementation have been used for other projects at ESOC. related with the CCSDS Mission Operations services technology.

The following projects used some modules and/or have been tested with parts of the NMF's Java implementation:

- EUD4MO
- METERON Robotic Services
- ESA/JPL Shadow project
- Support for the development of the HTTP Transport Binding
- Support for the development of the ZeroMQ Transport Binding

EUD4MO is a lightweight MCS for monitoring and control of remote MO providers. It is based on EUD and uses the Ground MO Adapter implementation for establishing the communications to the provider. Further explanation of this project can be found in chapter 5.

The METERON Robotic Services (MRS) were originally developed using prototyped web-services and were, later on, migrated to use CCSDS Mission Operations services. This migration took advantage of the COM and M&C implementation presented in this chapter. [94]

The ESA/JPL Shadow project is an interoperability activity between ESA and JPL to transfer Mars Express (MEX) telemetry data through the CCSDS Monitor and Control (M&C) services where ESA is the provider of the services and JPL the consumer. For this activity, ESA reused the NMF's Java implementation of the COM, M&C, and Common services. [46]

The HTTP and ZeroMQ Transport Bindings developed by ESA have been tested during their development using CTT and an NMF App. CTT provides a GUI with a set of gives high-level MO operations that can be triggered at the click of a button. It is possible and very easy to swap the underlying transport binding in order to test if a consumer can connect to a provider and exchange messages using a different transport binding.

Chapter 4

4. NMF SDK

The Software Development Kit (SDK) of the NanoSat MO Framework has the objective of facilitating the development of NMF-related software. The NMF SDK is a set of software development tools that allows the creation of applications with the NanoSat MO Framework for both ground and space. It includes the Consumer Test Tool (CTT) for interacting with NMF Apps, documentation, an application to assemble NMF Packages, a playground environment that includes a simple spacecraft simulator for experimentation, the software simulator client to control it, and finally, source code with demos of ground applications, NMF Apps, and others.

The starting point of the NMF SDK is the "Quick Start Guide" that includes a brief introduction, an explanation about MO services, and the new concepts introduced by NMF. Then it teases the reader to try out the NMF by starting an NMF App inside the Playground environment and connect to it using the CTT.

The NMF SDK is available online under ESA's open-source licence and can be downloaded by anyone around the world. [57]

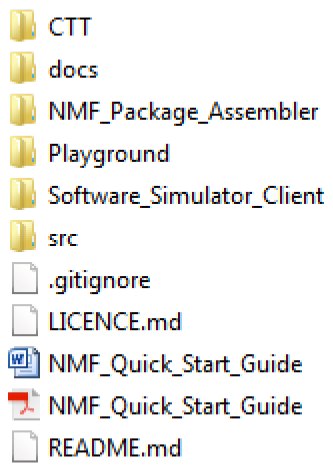

CTT
docs
NMF_Package_Assembler
Playground
Software_Simulator_Client
src
.gitignore
LICENCE.md
NMF_Quick_Start_Guide
NMF_Quick_Start_Guide
README.md

Figure 63: NMF SDK content.

An NMF developer does not need to write additional code related with specific requirements from the CCSDS MO services as these have already been pre-implemented by the NMF's Java implementation. This sharply decreases the difficulty of developing software based on CCSDS MO services.

4.1. Tackling the NMF Problem Domains

As mentioned in chapter 2, there are three different problem domains in the NanoSat MO Framework:

- NMF Ground Development
- NMF Mission Development
- NMF App Development

The SDK is aimed at helping the developers in the NMF Ground Development and NMF App Development problem domains. The on-board software portability capability of the framework allows the development of software that is agnostic to the actual mission and thus the developers can stay focused on developing the functionality required to achieve the mission goals without having to worry about the mission-specific implementation details.

The majority of developers are expected to be in these two problem domains and only a very restrict number of people will actually be developing on the NMF Mission Development problem domain. At least this is the expected case for OPS-SAT.

For the NMF App Development problem domain, the SDK contains a step-by-step guide as documentation that explains how to proceed in order to develop an NMF App. A guide for the development of NMF Ground applications is also available.

Although there are no dedicated guides for the NMF Mission Development problem domain, two NMF Mission implementations are made available online that can be taken as example by these developers:

- NMF Mission: Software Simulator
- NMF Mission: OPS-SAT

The first is presented in section 4.6.1 and it is connected to a lightweight software simulator that is used for the Playground environment. The second is presented in chapter 5 and it is implemented for the OPS-SAT mission.

4.2. NMF Apps Lifecycle

The NMF SDK is available online on GitHub under an open-source license and can be downloaded by anyone around the world. An NMF app can be developed by adding the correct dependency in the POM file of the project (in case maven is used) or by simply copying the source code of one of the many demo projects available in the SDK. During development, one can reuse existing libraries to speed up the development time and test it using CTT. [56]

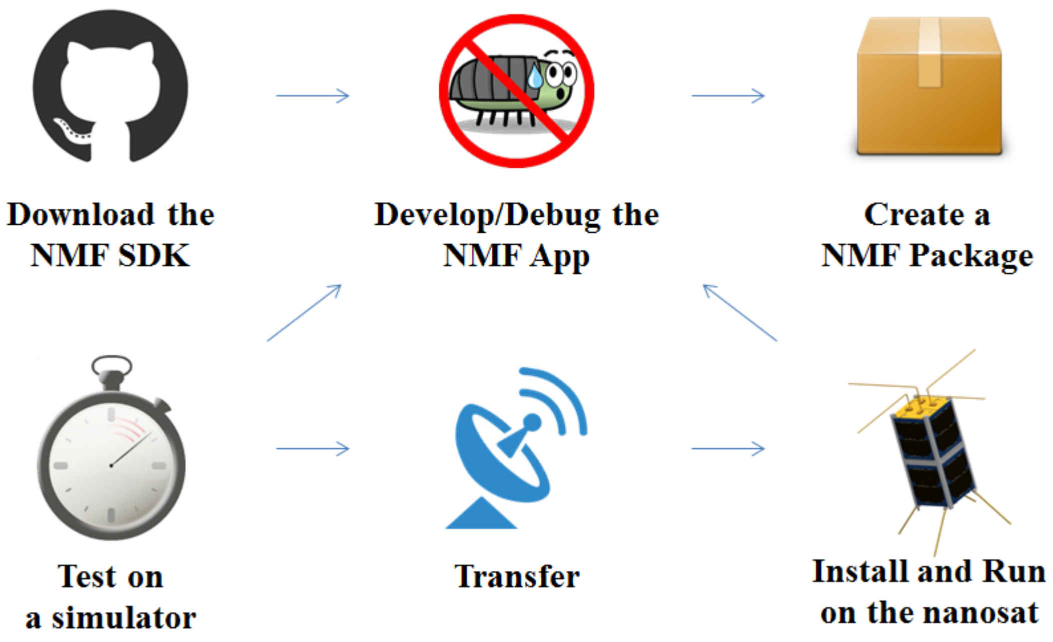


Figure 64: The lifecycle of an NMF App.

An NMF app can be developed using the NMF SDK and then packaged into a NMF Package using the NMF Package Assembler. After, the app can be tested on a simulator which can be purely software-based or a Flatsat emulating the nanosatellite’s hardware. Then transferred to the nanosatellite where it can be installed and started at any time. If a problem is found during the execution in the nanosatellite, it is possible to go back to the development phase and repeat the process. [56]

4.3. CTT: Consumer Test Tool

The Consumer Test Tool (CTT) is an application to aid developers in the process of testing newly developed NMF Apps from a user-friendly interface. It includes a set of GUI panels that provides a Graphical User Interface (GUI) to interact with the services on a provider. CTT can also be used as a learning tool for MO services, or to connect to MO providers other than NMF providers.

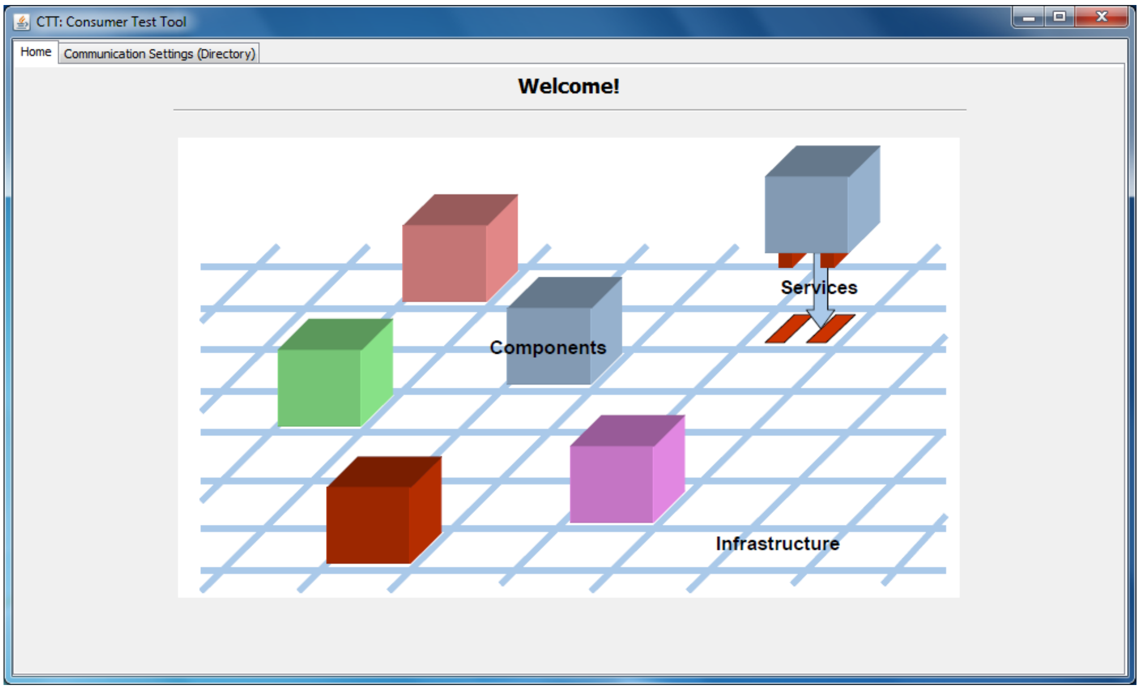


Figure 65: CTT splash screen.

In implementation terms, it uses the Ground MO Adapter for the connection to the providers. Although CTT is provided as "ready to be used" application, its source code is also available for developers to learn and take as example.

When CTT is used to test new NMF Apps, the Simple Scenario from chapter 2 is expected to be set, as represented in Figure 66. Other scenarios can also use CTT for testing providers of MO services.

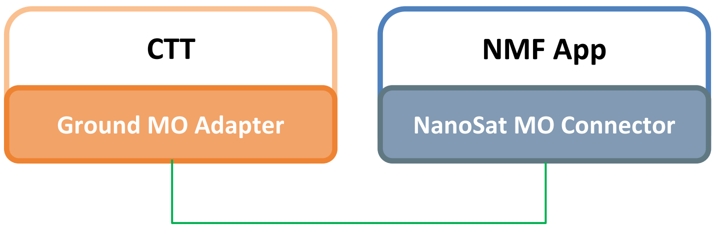
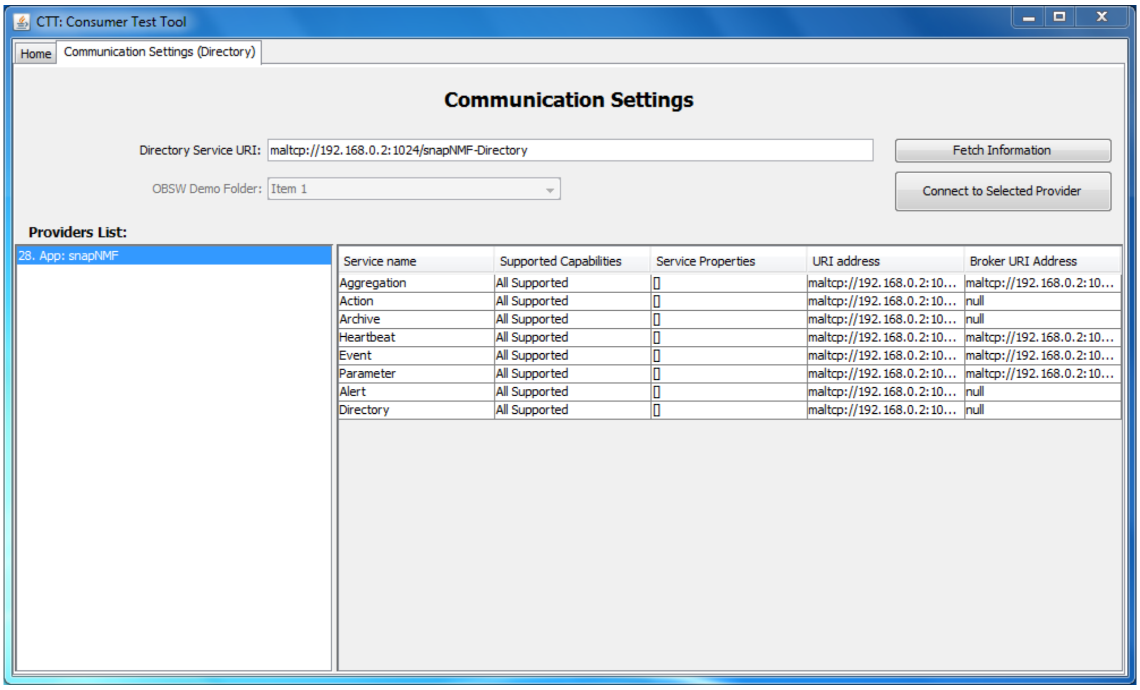


Figure 66: CTT in the Simple Scenario.

In order to connect to a provider, CTT provides two possible mechanisms, one using a single URI from the Directory service, and another using the URIs of all the services available on the provider by ingesting a file (providerURIs.properties) containing them. The former is promoted via the “Communication Settings (Directory)” as it greatly simplifies the connections setup. Figure 67 displays on top the URI from the Directory service to where CTT was connected to, and down below displays the list of providers on the left side and the corresponding connections details of the selected provider on the right side.



Directory Service URI: maltcp://192.168.0.2:1024/snapNMF-Directory

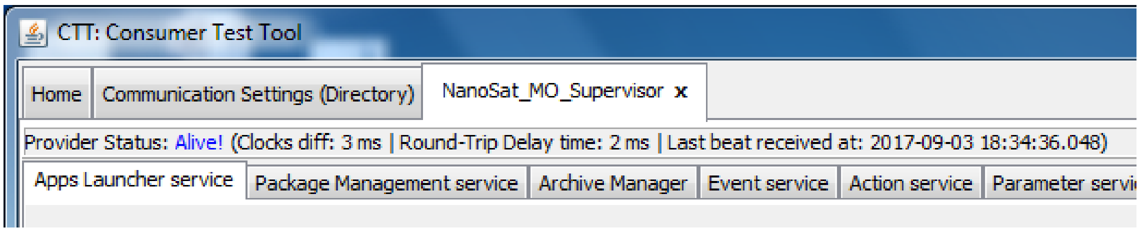
OBSW Demo Folder: Item 1

28. App: snapNMF

Service name	Supported Capabilities	Service Properties	URI address	Broker URI Address
Aggregation	All Supported	[]	maltcp://192.168.0.2:10...	maltcp://192.168.0.2:10...
Action	All Supported	[]	maltcp://192.168.0.2:10...	null
Archive	All Supported	[]	maltcp://192.168.0.2:10...	null
Heartbeat	All Supported	[]	maltcp://192.168.0.2:10...	maltcp://192.168.0.2:10...
Event	All Supported	[]	maltcp://192.168.0.2:10...	maltcp://192.168.0.2:10...
Parameter	All Supported	[]	maltcp://192.168.0.2:10...	maltcp://192.168.0.2:10...
Alert	All Supported	[]	maltcp://192.168.0.2:10...	null
Directory	All Supported	[]	maltcp://192.168.0.2:10...	null

Figure 67: CTT Connections.

CTT was designed to support the connection to multiple NMF Apps simultaneously via a tabbed view mechanism. After pressing the “Connect to Selected Provider” button, it creates a new instance of the Ground MO Adapter that connects to the provider, and then opens a new tab on top for interacting with it. Figure 68 presents an example of a tab corresponding to the connected provider.

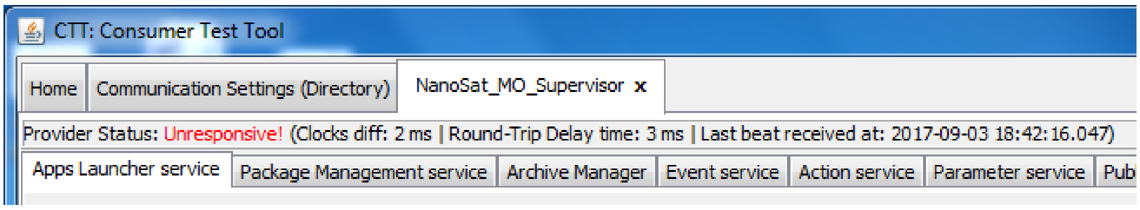


Provider Status: Alive! (Clocks diff: 3 ms | Round-Trip Delay time: 2 ms | Last beat received at: 2017-09-03 18:34:36.048)

Figure 68: The providers’ tabs and the status bar of a provider.

Inside the provider tab there is the provider status bar that indicates the status of the provider and calculates the round-trip delay time. In terms of implementation, this bar takes advantage of the Heartbeat service from the Software Management services presented in chapter 2 by subscribing and listening to the periodic beats coming from the service. If the beats are being received within the expected time window then the status is set as “Alive!”. If CTT stops receiving the beats then the status changes to “Unresponsive”, just as represented in Figure 69.

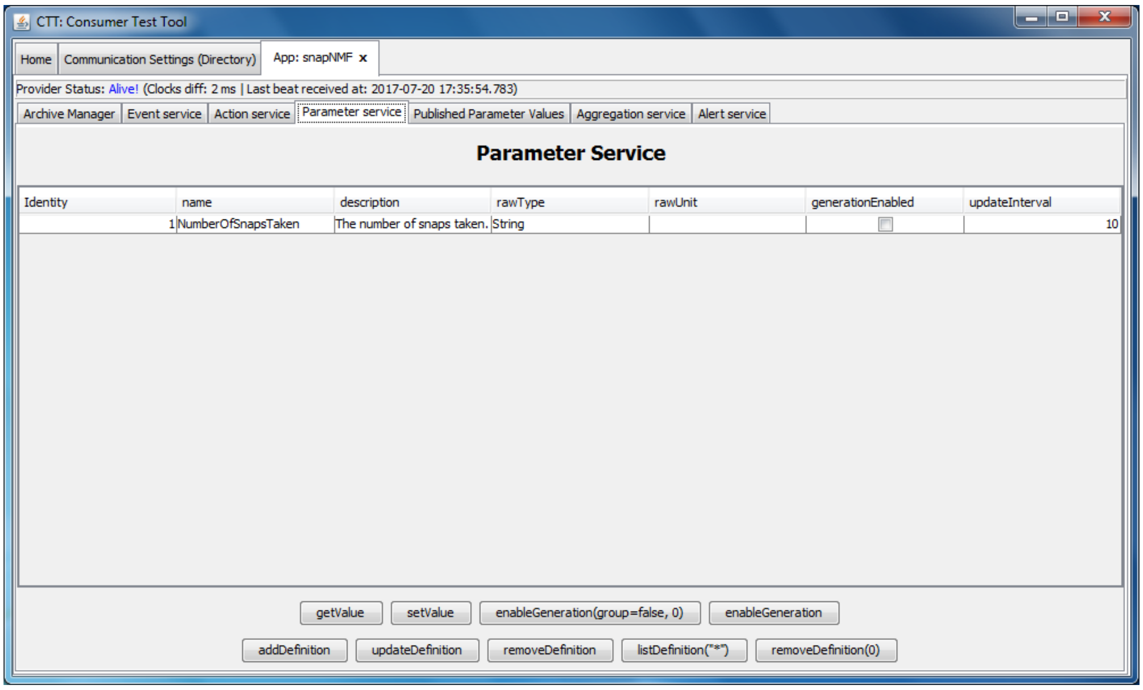
If the Heartbeat service does not exist on the provider, the bar displays the message: "Heartbeat service not available."



Provider Status: Unresponsive! (Clocks diff: 2 ms | Round-Trip Delay time: 3 ms | Last beat received at: 2017-09-03 18:42:16.047)

Figure 69: From top to bottom: providers’ tabs; provider status bar; and the services’ tabs.

Inside a provider tab, not only the provider status bar exists, but also a set of tabs dedicated to the services available on the provider. On the bottom of Figure 69, an example of a set of tabs for the services available on the NMF App is presented.



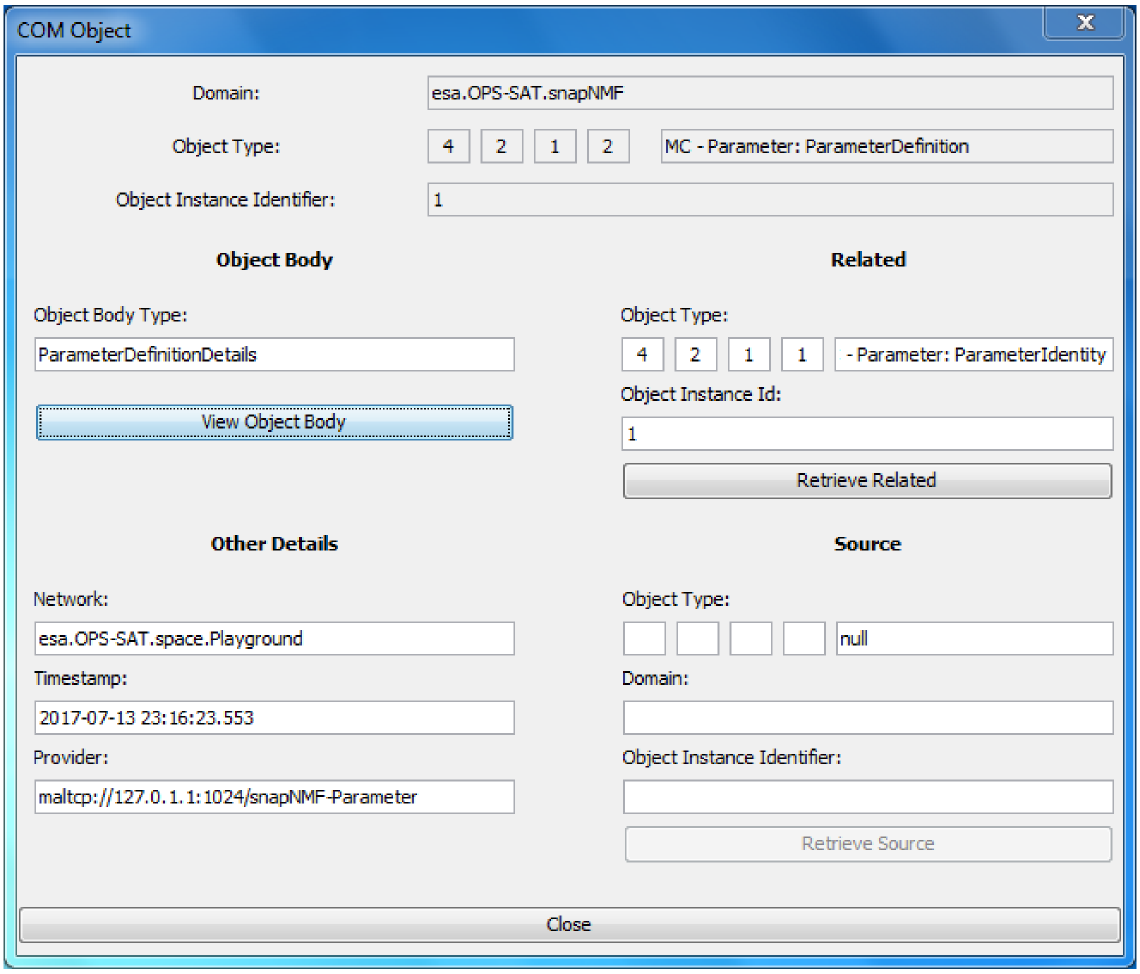
Provider Status: Alive! (Clocks diff: 2 ms | Last beat received at: 2017-07-20 17:35:54.783)

Identity	name	description	rawType	rawUnit	generationEnabled	updateInterval
1	NumberOfSnapsTaken	The number of snaps taken.	String		☐	10

Figure 70: CTT connected to one NMF app, displaying the Parameter service tab.

All the services' tabs provide a similar look and feel with a list of entries being displayed and a few buttons to trigger the operations specified in the button text. Figure 70 presents the Parameter service tab with one single parameter definition on the list of definitions.

One can double-click on any of the entries of the tables and a popup window will open displaying the COM Object of the associated COM Object with that entry. Figure 71 presents an example of a COM Object window displaying a Parameter Definition.



Domain: esa.OPS-SAT.snapNMF
Object Type: 4 2 1 2 MC - Parameter: ParameterDefinition
Object Instance Identifier: 1

Object Body

Object Body Type: ParameterDefinitionDetails

Related

Object Type: 4 2 1 1 - Parameter: ParameterIdentity
Object Instance Id: 1

Other Details

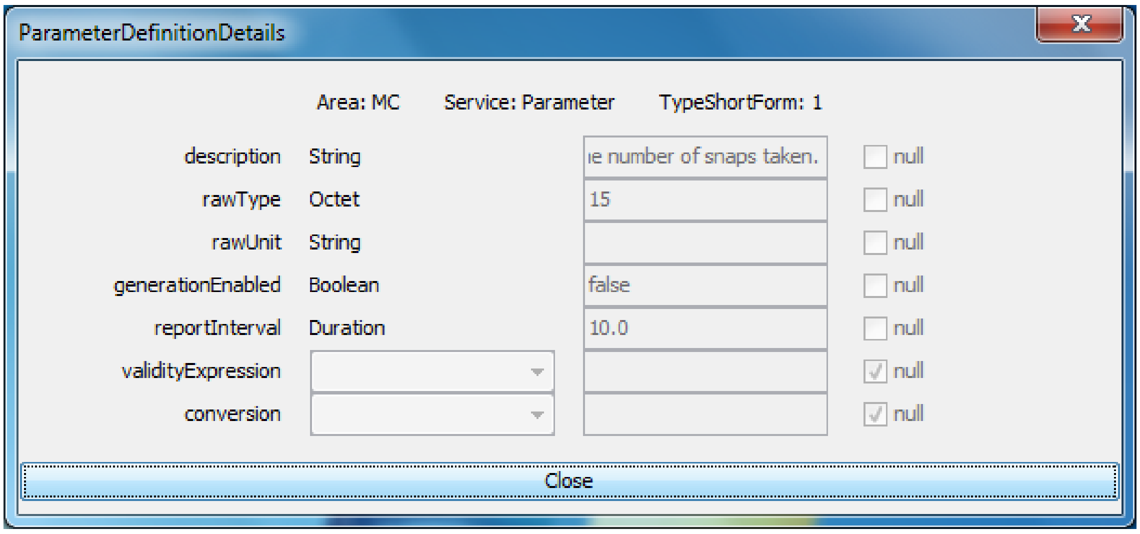
Network: esa.OPS-SAT.space.Playground
Timestamp: 2017-07-13 23:16:23.553
Provider: maltcp://127.0.1.1:1024/snapNMF-Parameter

Source

Object Type: null
Domain:
Object Instance Identifier:

Figure 71: CTT displaying an instance of a COM Object.

By pressing the "View Object Body" button of the COM Object window, a MAL Data Type window will open and display the object body of the COM Object. Figure 72 presents an example of a ParameterDefinitionDetails data structure and its respective fields.


ParameterDefinitionDetails

Area: MC Service: Parameter TypeShortForm: 1

description	String	ıe number of snaps taken.	☐ null
rawType	Octet	15	☐ null
rawUnit	String		☐ null
generationEnabled	Boolean	false	☐ null
reportInterval	Duration	10.0	☐ null
validityExpression			✓ null
conversion			✓ null

Close

Figure 72: CTT displaying an instance of a MAL Data Type Composite.

Historically, CTT derived from the "Consumer Interface (CCSDS MO)" application that was developed for the "Implementation of 2 CCSDS MO services on a MityArm board" of a master thesis project. During the research period, this code was reiterated multiple times for improvement and therefore multiple new features were implemented to a point where the original code of the project is almost inexistent. CTT has been called "Configuration Manager" during the first year of the research however the name was not appropriate and therefore it was changed to "Consumer Test Tool" or CTT. [22]

4.4. Documentation

The “Quick Start Guide” should be the entry point for anyone using the NMF SDK for the first time. This guide attempts to provide a high-level view of the NanoSat MO Framework, the new concepts, and a quick “test drive” by running an NMF App and connecting to it.

Additional digital documentation is also available in the “docs” folder of the NMF SDK and it helps the developers to move forward in the development process of their NMF App or NMF Ground application. For this, two additional guides can be found:

- Guide to develop NMF Apps
- Guide to develop NMF Ground applications

The documents with the MO services specification can be found inside the “MO_services” folder and they are very important for any developer that wants to know how the services are supposed to behave.

The APIs of the services are also available in the “Javadocs” folder and they can be visualized using a browser by the developers that are more familiar with javadocs in HTML format.

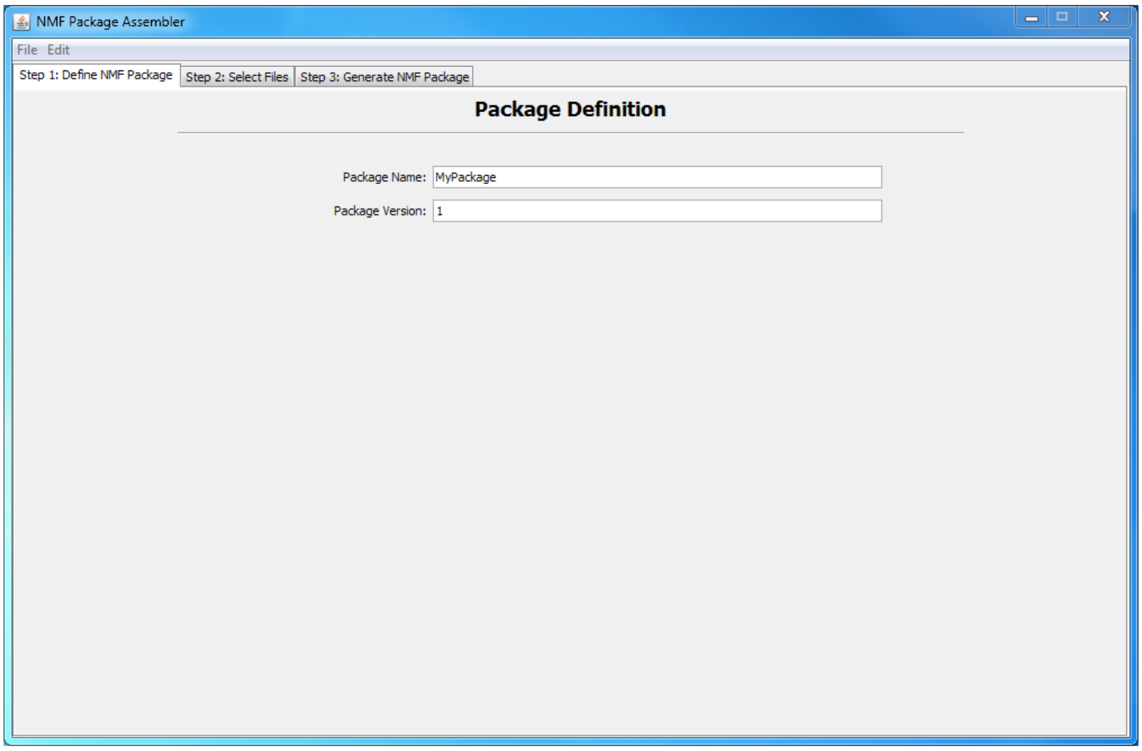
Some documentation following the recommendations from the ECSS-E-ST-40C standard, “ECSS, Space engineering: Software”, was produced. [95]

This Dissertation is also intended to be available in the NMF SDK.

4.5. NMF Package Assembler

The NMF Package Assembler is an application that allows a user to assemble an NMF Packages from a user-friendly GUI. The application includes three tabs: One for defining properties, another for selecting the files to be part of the package, and finally a tab for the generation of the NMF Package.

The first tab allows the selection of a package name and a package version.



Package Name: MyPackage

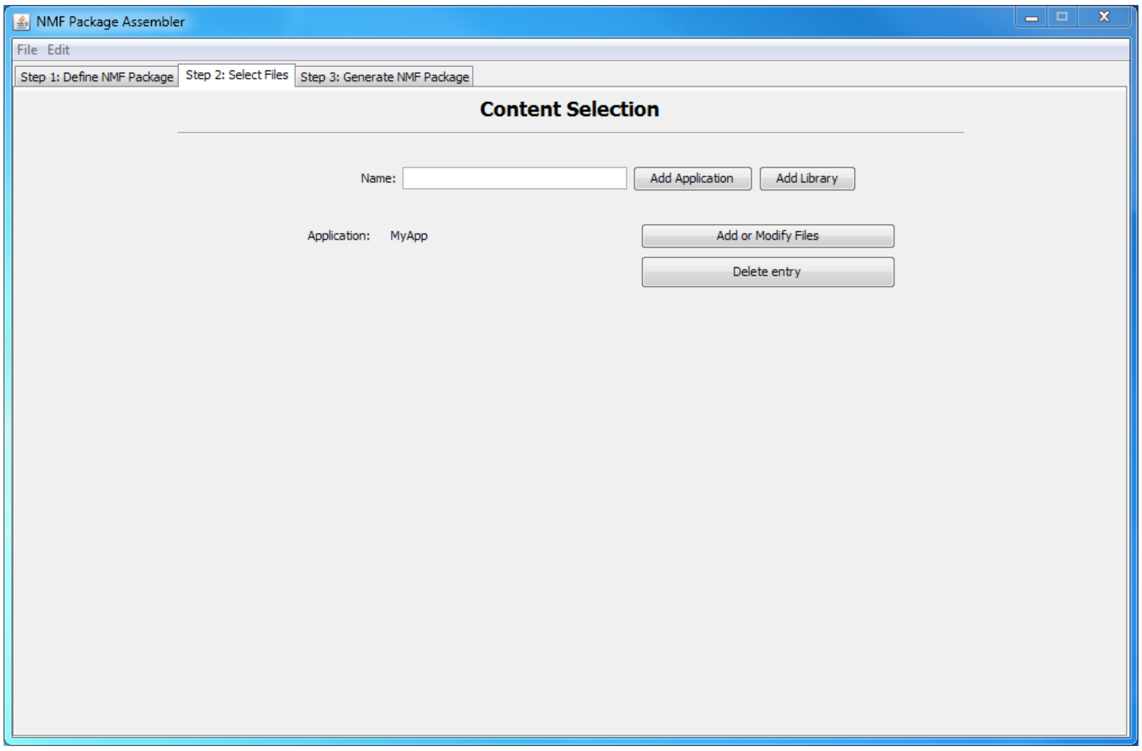
Package Version: 1

Figure 73: NMF Package Assembler, Step 1.

As explained in section 2.7.3, the NMF Packages are not tied to the NMF Apps concept and can also carry applications and files that are not NMF-related.

The second tab allows the selection of a name for either an application or a library. The third tab allows the selection of the output folder to store the newly generated NMF Package. The package can be generated by pressing “Generate NMF Package!”.

The different tabs can be visualized in Figure 73, Figure 74, and Figure 75.

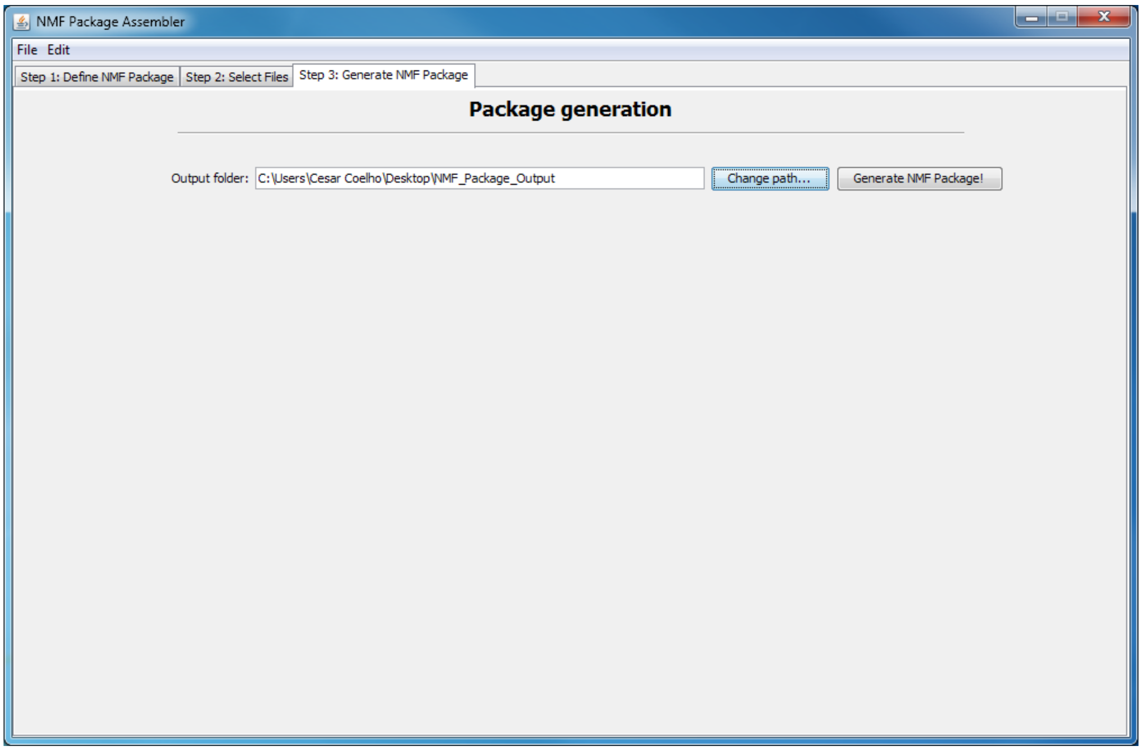


Content Selection

Name:

Application: MyApp

Figure 74: NMF Package Assembler, Step 2.



Package generation

Output folder: C:\Users\Cesar Coelho\Desktop\NMF_Package_Output

Figure 75: NMF Package Assembler, Step 3.

4.6. Playground

The Playground is a multi-purpose environment for experimenting with NMF-related software. It follows the Generic Deployment mentioned in chapter 3 and includes a set of NMF Apps ready to be used and an NMF Library for the "NMF Mission: Software Simulator".

The Playground environment can be used as a learning tool because the NMF Apps are already compiled and can just be started by running the script runAppWin.bat in Windows or runAppLin.sh in Linux. This allows a newcomer to get familiar with the NMF ideas without actually doing any code.

During the development of an NMF Ground application, usually it is necessary to test the application by connecting it to a provider side. The NMF Apps available in the Playground can be used for this purpose and save time to a ground developer.

As explained in section 4.5, one can create an NMF Package after developing the NMF App in order to have one single file containing all the necessary data for installation, uninstallation, and update. The Playground environment includes the NanoSat MO Supervisor application that allows the management of software, therefore it is possible to test the installation, uninstallation, and update of the generated NMF Package.

Additionally, applications can also be started, stopped, and killed from the NanoSat MO Supervisor. This includes NMF Apps but it can also be used for other applications. Chapter 3 includes further details on the implementation of the Apps Launcher Java implementation.

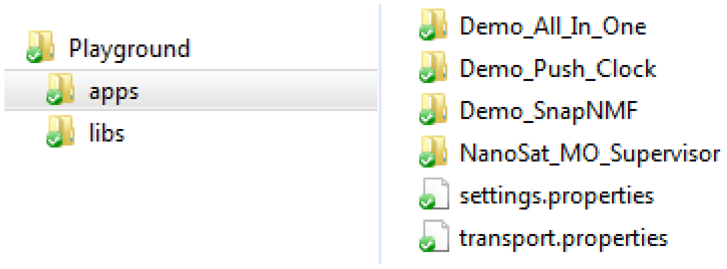

Playground
apps
libs
Demo_All_In_One
Demo_Push_Clock
Demo_SnapNMF
NanoSat_MO_Supervisor
settings.properties
transport.properties

Figure 76: Playground folder.

The Playground environment is available on the SDK as an out of the box solution, but it can be regenerated automatically from the "Playground Generator" project available in the "src" folder of the SDK. This is very helpful in case the user ends up in an irrecoverable situation, since the Playground environment can simply be deleted and generated again.

The Playground Generator project can be extended to support the generation of other software to include, for example, more applications in the Playground environment. A software developer developing an NMF App can add a "hook" to the Playground

Generator project to automatically include its NMF App on the Playground environment. This allows bypassing the package creation step between the develop/debug phase and testing phase of the NMF Apps Lifecycle.

An NMF Mission with its Platform services connected to a lightweight simulator was developed during the research and it is used by the Playground environment. The next two subsections present the "NMF Mission: Software Simulator" and its respective simulator.

4.6.1. NMF Mission: Software Simulator

The "NMF Mission: Software Simulator" is an implementation of the NanoSat MO Framework for a lightweight simulated mission using only software. It uses the Java implementation presented in chapter 3 with the Platform services using adapters connected to the peripherals of the OPS-SAT Software Simulator.

The OPS-SAT Software Simulator is presented in section 4.6.2.

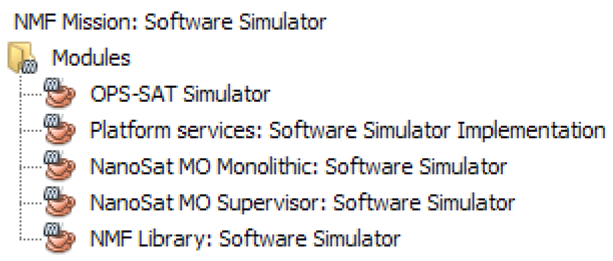

NMF Mission: Software Simulator
Modules
OPS-SAT Simulator
Platform services: Software Simulator Implementation
NanoSat MO Monolithic: Software Simulator
NanoSat MO Supervisor: Software Simulator
NMF Library: Software Simulator

Figure 77: "NMF Mission: Software Simulator" sets of implemented packages.

The "NMF Mission: Software Simulator" was implemented to support both the "Monolithic" deployment and the "Multiple applications" deployment on the NanoSat segment. Therefore, the NanoSat MO Monolithic and the NanoSat MO Supervisor components have been both implemented with a small set of parameters, aggregations, alerts, and actions. Additionally, the NMF Library was implemented to be used by other NMF Apps linking to it as explained in section 2.7.5.

The Playground environment uses the "Multiple applications" deployment and therefore includes the NanoSat MO Supervisor which is responsible for the management of the applications, and the NMF Library to be used by other applications. The NanoSat MO Monolithic is used in the "Monolithic Provider Demo" which is available as a source code example.

Both of the deployments mentioned above allow developers to get simple data from the Platform services without the need of having a full and heavy simulator running. This facilitates the development of software for both the NMF App Development and for the NMF Ground Development problem domains.

The “NMF Mission: OPS-SAT” is explained in chapter 5 and it includes the full description of the integration with the real OPS-SAT peripherals. A quick overview about the “OPS-SAT Software Simulator” is presented in section 4.6.2.

4.6.2. OPS-SAT Software Simulator

The OPS-SAT Software Simulator is an implementation of lightweight functional simulation models of OPS-SAT peripherals and orbit/attitude behaviour in order to provide a sufficiently realistic runtime environment for software development. [96]

The simulator exposes an API for executing commands which affect the OPS-SAT payload instruments and/or retrieve science data and telemetry. The commands can be run either from the “NMF Mission: Software Simulator” or manually, from a remote GUI. [96]

The space dynamics library Orekit was used for the simulation of the satellite’s position that is retrieved by GPS service implementation. Additionally, it was also used for retrieving and setting the desired spacecraft’s attitude from AutonomousADCS service implementation. [97]

For the camera, the simulator allows the retrieval of an actual picture that was previously taken by a Hyperion Technologies ST200 camera, the same model flying in OPS-SAT. The output of the SDR is a stream of I/Q samples (in-phase and quadrature) and these can be written into a .wav audio file for standardized storage. The simulator contains a .wav file reader and it offers commands for preloading .wav files and then reading a number of samples. For the Software Defined Radio, the simulation model relies on a data buffer that can be set directly with a command or from a file. Additionally, there is a success rate parameter that can be configured which represents the probability of reading the data from the buffer without errors. [96]

4.7. Source Code Examples

The NMF SDK’s source code examples are available in the “src” folder of the NMF SDK. This includes demos of NMF Apps, demos of NMF Ground applications, a demo of a Monolithic Provider, and some other projects.

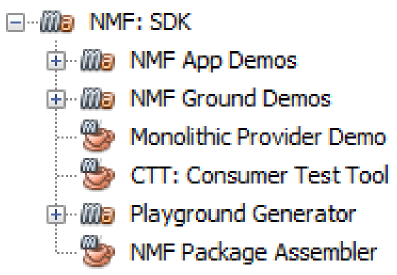


Figure 78: NMF SDK source code examples.

4.7.1. Demos of NMF Apps

The source code examples include a package containing a set of demos with NMF Apps. These demos are very simple applications which can be used by the developers as models. The following NMF Apps demos are available:

- Hello World App
- Push Clock App
- 10 seconds Alert App
- 5 stages Action App
- GPS data App
- All in One App
- SnapNMF
- Serialized object

Each one of these NMF Apps has a different functionality and it is intended to demonstrate something different. For example, the Hello World App demonstrates the use of the getter and setter of the Parameter service, and the SnapNMF demonstrates how to take a picture using the Camera service.

4.7.2. Demos of NMF Ground applications

The source code examples include a package containing a set of demos with NMF Ground applications. These demos are very simple applications which can be used by the developers as models. The following NMF Ground application demos are available:

- Ground Zero
- Ground with Directory service
- Ground Set and Command
- Ground Facebook

Each one of these NMF Ground applications has a different functionality and it is intended to demonstrate something different. For example, the Ground Zero application demonstrates how to establish the connection to a provider and how a simple adapter can be submitted in order to receive parameter values.

4.7.3. Monolithic Provider Demo

The source code examples include an application of a Monolithic Provider demo. This application uses the “Monolithic” deployment with the NanoSat MO Monolithic component that was developed as part of the “NMF Mission: Software Simulator”. Therefore, the Platform services of this application are connected to the OPS-SAT Software Simulator.

4.7.4. Other projects

The source code examples also include the source code for the projects: CTT, Playground Generator, and NMF Package Assembler.

The Playground Generator project allows the regeneration of the Playground environment with the selected projects. A similar project was done for the OPS-SAT implementation in order to generate the on-board side with some applications ready to be started.

Chapter 5

5. NMF MISSION: OPS-SAT

The “NMF Mission: OPS-SAT” is an implementation of the NanoSat MO Framework for ESA’s OPS-SAT mission. It uses the Java implementation presented in chapter 3 with the Platform services using adapters connected to the APIs for the payload peripherals of OPS-SAT.

OPS-SAT is one of the first missions that intent to take advantage of the NanoSat MO Framework in order to manage its software on the Experimental Platform.

The next sections present the overall OPS-SAT system deployment, the scenario that was defined for OPS-SAT’s NMF Mission, the details of the selected transport binding for the communications to ground, the integration details of the Platform adapters with OPS-SAT’s payload peripherals, and finally, the implementation customizations that were done for other services.

The majority of the software development for the NMF Mission was done for the spacecraft side. On ground, only the Ground MO Proxy for OPS-SAT had to be developed as part of the NMF Mission.

The Ground MO Proxy allows other NMF Ground applications to connect to any NMF App as this is part of the portability concept of the NMF. On one hand it means that an NMF Ground application is able to connect to other future NMF Apps, and on the other hand it means that NMF Apps developed for OPS-SAT can be reused by future spacecraft that have an NMF Mission in place.

On the spacecraft side, OPS-SAT’s main bus is a Controller Area Network (CAN) bus which is capable of a 1Mbit/second bit rate and it is used for the exchange of messages between the CCSDS Engine, the Experimental Platform, and the Nanomind. This bus is represented in yellow in Figure 79. [98]

It is also possible to visualize in Figure 79 the five payload devices in OPS-SAT: Camera, Optical Data Receiver, Software-defined Radio (SDR), FineADCS, and a GPS.

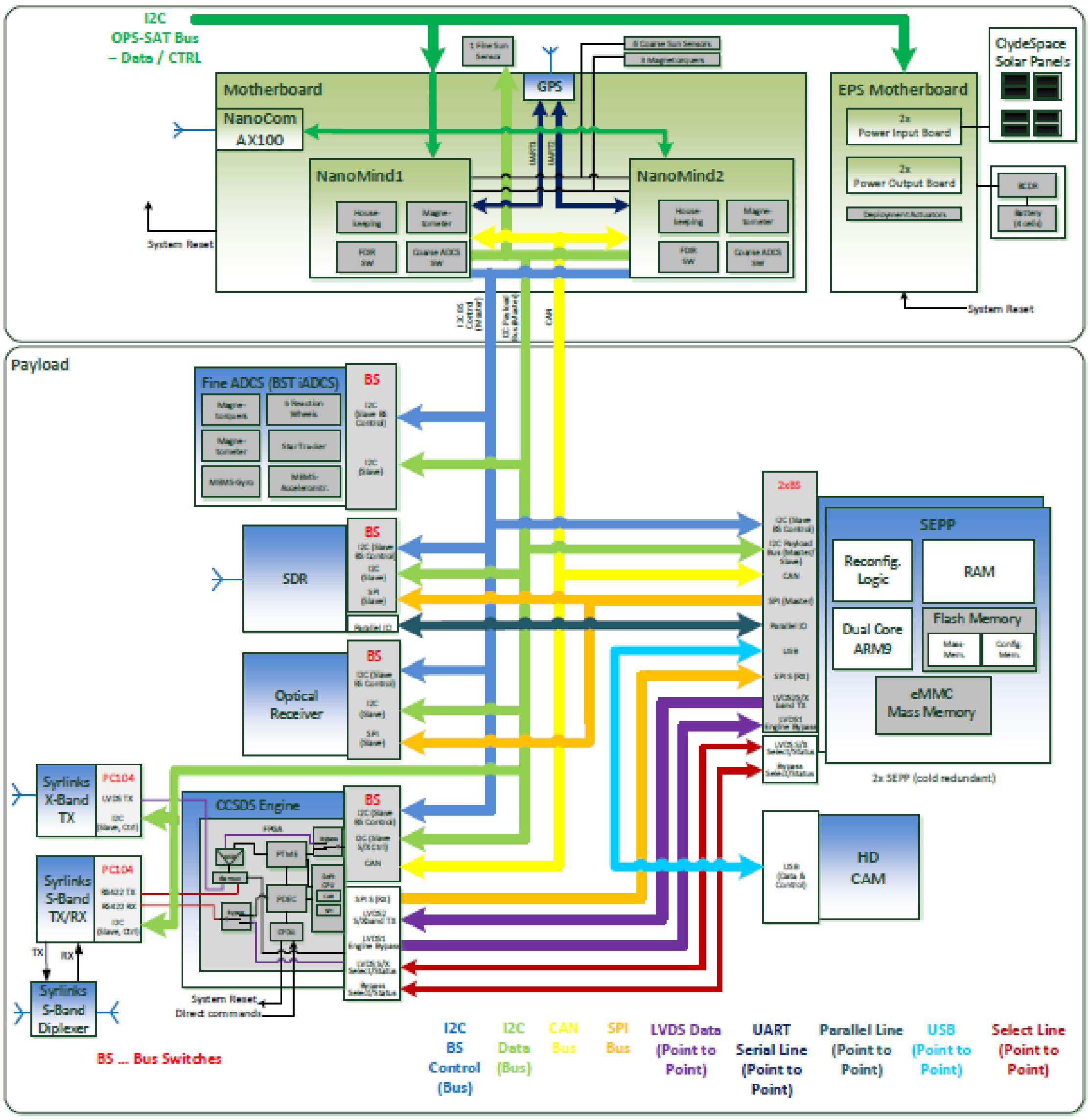


Figure 79: OPS-SAT spacecraft bus.

5.1. OPS-SAT System Deployment

The OPS-SAT System Deployment is composed of 4 main parts. On the Experimental Platform there's a mityARM device with 1 GB of RAM and an ARM processor with 900 MHz. On the OMCS and Data Relay Server, ESOC's infrastructure is ready to support complex missions and includes many technical experts, a dedicated network, and its own virtual cloud. On the Remote experimenter side, the resources are an issue to be dealt by the experimenter.

The infrastructure itself is not so relevant because the resources are not so scarce therefore the focus of this section is on the applications running inside the nodes and not on the hardware.

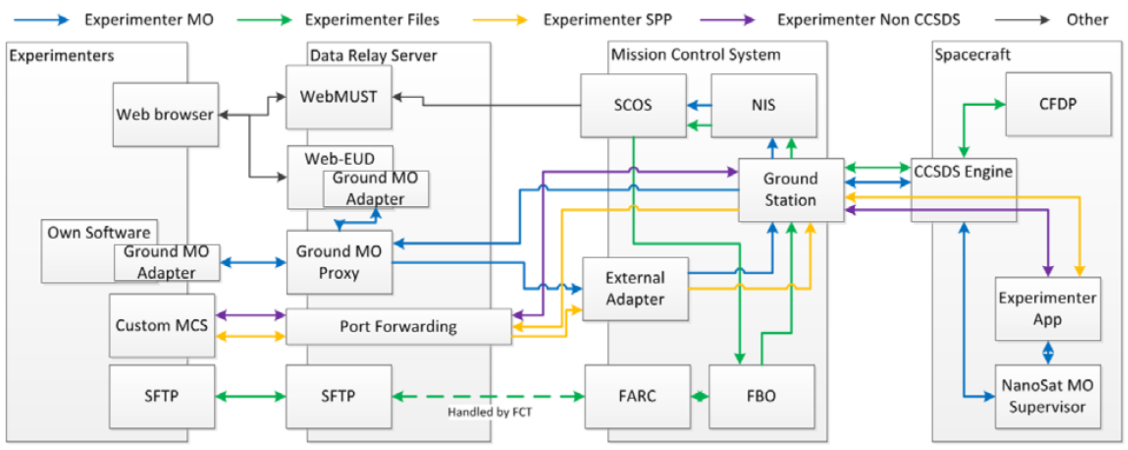


Figure 80: System Deployment Overall view. [21]

Figure 80 is an overall view of the system deployment however, it is a simplified view and does not represent all the details of the system. Some blocks were not represented in the Mission Control System node for two reasons. First, privacy concerns in ESOC's Mission Control System and second, the details of the node are not very relevant for this dissertation.

The next subsections present an overview of the software applications running on the different nodes.

5.1.1. Experimental Platform

The Experimental Platform node is where the software experiments will run and they can be just simple applications, FPGA images, or complete OS images. A Standard OS image will be available containing default operational software: FileApp, and NanoSat MO Supervisor. [21]

The FileApp application allows the transfer of files between ground and space using CFDP but this is out of scope for this dissertation. The NanoSat MO Framework allows the management of the experiments and in case they are NMF Apps, to find them from ground.

The NMF implementation for OPS-SAT follows the Generic Deployment mentioned in chapter 3 and includes a set of NMF Apps ready to be used for testing the software management mechanisms for the three different types of software experiment. An NMF Library for OPS-SAT was compiled and can be referenced by the NMF Apps running on-board.

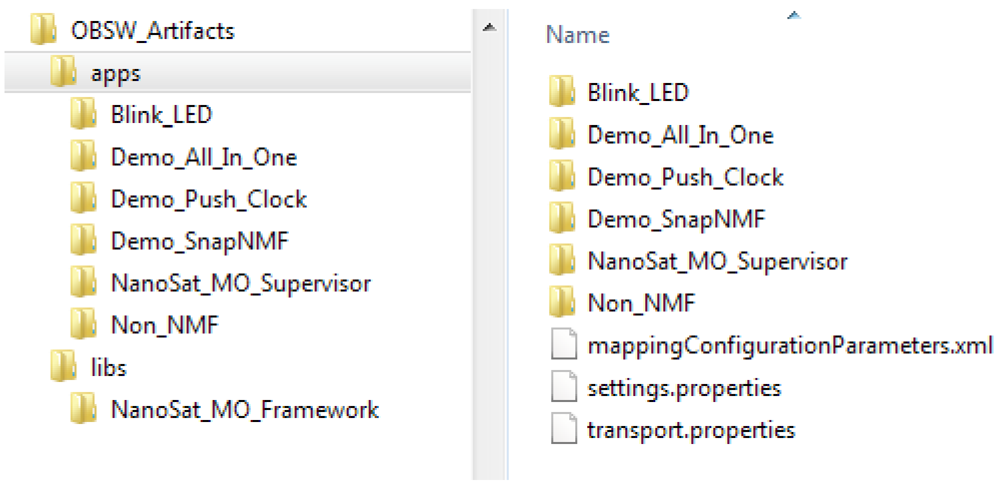


Figure 81: NanoSat MO Framework deployment in OPS-SAT.

The folder's structure presented in Figure 81 can be automatically generated from a maven project. Further details are presented in section 6.1.2.

Some additional software was installed in order to run the NMF in OPS-SAT including the Java Runtime Environment version 8 for Linux for an ARM device, and the socketcand application which exposes a simple TCP/IP socket to exchange data with the CAN bus. [99]

5.1.2. Mission Control System

The Mission Control System node is composed by a set of applications that run on separate virtual machines. This includes NIS, OMCS, FARC, and others that are not represented for simplicity.

The Network Interface System (NIS) is the gateway between the Mission Control System and the Ground Station and provides means for TM/TC transfer via the CCSDS SLE specification as well as file transfers from ESA's Telemetry and Telecommand System to the NIS. [100]

The OPS-SAT Mission Control System (OMCS) is composed of 3 main chains, the File Transfer chain, the Monitor and Control chain, and the Experimenter's chain. The File Transfer chain includes an application with an implementation of the CFDP standard, the Monitor and Control chain includes SCOS-2000 (Spacecraft Control and Operations System), and the Experimenter's chain provides encoding and decoding of space packets for the applications running on the Data Relay Server. [21]

No NMF-related application is running in the OPS-SAT Ground Data Systems node.

5.1.3. Data Relay Server

The Data Relay Server node is composed by a set of applications that run on separate virtual machines. This includes EUD4MO, Ground MO Proxy, Port Forwarding, and SFTP. [21]

EUD4MO and the Ground MO Proxy were developed using the NMF Composites and therefore they have dedicated subsections.

The Port Forwarding application allows forwarding data from/to the Experimenter's chain of OPS-SAT's Ground Data Systems to/from the Experimenter's Remote System. The interface towards the Experimenter's Remote System is TCP/IP and allows the exchange of space packets directly from/to the spacecraft. SFTP is also available in order to receive files that have been generated during the experiment's execution in space.

Ideally, these applications should be instantiated for every new experimenter but this mechanism is not defined at the time of writing.

5.1.4. Experimenter Remote System

The Experimenter Remote System node is in the experimenter's complete control and therefore it can be anything. Based on the interfaces exposed by the Data Relay Server node, A few possibilities are envisaged: a web browser, an MCS with the Ground MO Adapter, a dedicated MCS for the experiment using space packets, or a SFTP application. [21]

A web browser can be used to access EUD4MO for monitoring and control of an experiment. This is the least effort solution for an experimenter as most of the Operating Systems today come already bundled with a web browser. The only necessary requirement to meet is to develop the experiment in form of an NMF App.

An MCS with the Ground MO Adapter can be connected to the Ground MO Proxy which would provide protocol bridging, COM Archive mirroring, queuing of actions, and other functionalities.

A dedicated MCS for the experiment using space packets can be connected to the Port Forwarding application for directly sending/receiving space packets to/from the spacecraft. And finally, a SFTP client application can be used for exchanging files with the Data Relay Server.

5.2. OPS-SAT Scenario

The OPS-SAT scenario is a complex scenario created by the combination of different Generic Scenarios presented in chapter 2. On the NanoSat segment, it follows the Multiple NMF Apps scenario, and on the Ground segment, it follows the Multiple Consumers scenario combined with the Proxy scenario.

In OPS-SAT, an NMF App can be seen as an "experiment". Experimenters can develop their NMF App using the NMF SDK, create the respective NMF Package and then this will be tested, transferred and installed by ESOC. When the satellite is ready, the "experiment" is started.

Figure 82 represents the OPS-SAT scenario and it only goes down to the MAL binding layer therefore does not represent the components that operate based on data of the layers below. The green line between Ground MO Proxy and both EUD4MO and the External System, represent a communication using a MAL-TCP/IP binding. The blue line represents a MAL-SPP binding however the two different segments have different mechanisms of exchanging the space packets. The different mechanisms are further explained in section 5.3.

On ground, an "experiment" can be monitored and controlled either using EUD4MO or by connecting a NMF Ground application to the Ground MO Proxy. EUD4MO is a solution developed by ESA that allows a user to use a simple web browser to monitor and control an NMF App.

There is also a raw interface to exchange space packets directly with the spacecraft however it bypasses the Ground MO Proxy and therefore all the advantages of using a Ground MO Proxy are lost. Thus, it will not be present in this section but one can find it in section 5.1.

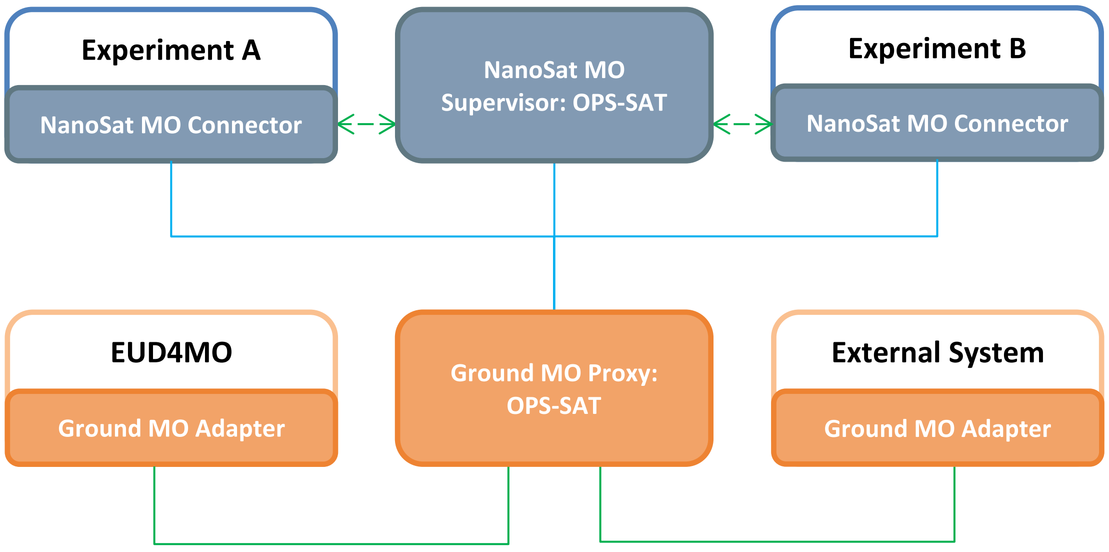


Figure 82: OPS-SAT scenario.

5.2.1. NanoSat MO Supervisor for OPS-SAT

The NanoSat MO Supervisor application acts as a supervisor to be deployed on the NanoSat segment of OPS-SAT. It extends the generic NanoSat MO Supervisor of the Java implementation presented in chapter 3 and therefore most of the default behavior of the component is already present.

It includes the Platform services implementation presented in chapter 3 with the additional adapters for the OPS-SAT platform devices. The integration of these adapters is further explained in section 2.4.

The Platform services can be consumed from both the MAL-SPP transport binding and the MAL-TCP/IP transport binding. The former is intended to be used between the NanoSat and the Ground segments, while the latter is intended to be used by IPC between NMF Apps and the NanoSat MO Supervisor.

A dedicated adapter for the monitoring and control was implemented. The getters and setters for the Parameter service were implemented for a few set of parameters and the correct dispatch of actions for certain method calls was also implemented.

Three parameters were defined with the following names and respective descriptions:

- CurrentPartition: "The Current partition where the OS is running."
- LinuxVersion: "The version of the software."
- CANDataRate: "The data rate on the can bus."

Three actions were defined with the following names and respective descriptions:

- GPS_Sentence: "Injects the NMEA sentence identifier into the CAN bus."
- Reboot_MityArm: "Reboots the mityArm."
- Clock.setTimeUsingDeltaMilliseconds: "Sets the clock using a diff between the on-board time and the desired time."

5.2.2. Ground MO Proxy for OPS-SAT

The Ground MO Proxy application acts as a proxy that is deployed on the Ground segment for ESA’s OPS-SAT mission. It extends the generic Ground MO Proxy of the Java implementation presented in chapter 3 and therefore most of the default behavior of the component is already present.

It is capable of acting as a protocol bridge and as a COM Archive mirror. As a result, multiple consumers can share the same ground-to-space connection to the spacecraft, and connect to independent NMF Apps simultaneously.

The protocol bridge includes on one side the dedicated MAL-SPP transport binding developed for OPS-SAT and on the other side includes the MAL-TCP/IP transport binding.

The COM Archive mirror on ground has a single instance of the COM Archive implementation. If multiple NMF Apps are running simultaneously on the NanoSat segment, then they will synchronize with a single instance of the COM Archive on the Ground segment.

If one assumes that one single experiment runs on a certain timeslot, then the Ground MO Proxy application for OPS-SAT should be deployed as a new instance whenever a new experiment is started.

Two NMF Apps can run simultaneously however the COM Archive data for both of them is stored on the same instance on ground. This might generate privacy concerns for experimenters that want to have private data.

5.2.3. EUD4MO

EUD4MO is a web-based lightweight Mission Control System (MCS) for monitoring and control remote MO providers. In implementation terms, is uses EUD for displaying the information in a GUI, and it uses the Ground MO Adapter in order connect and exchange data with an MO provider. Although EUD4MO is presented in this chapter, it is not limited to OPS-SAT and can be used by other future missions.

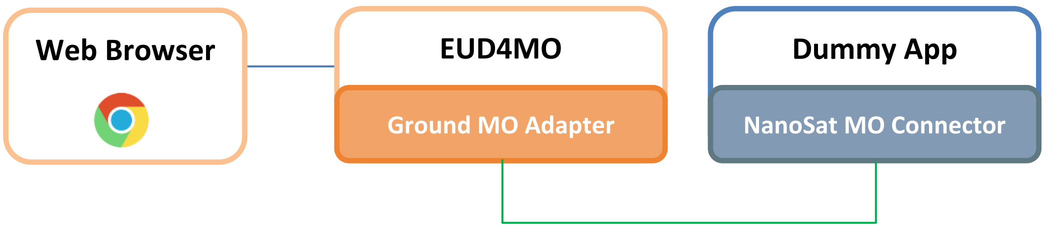
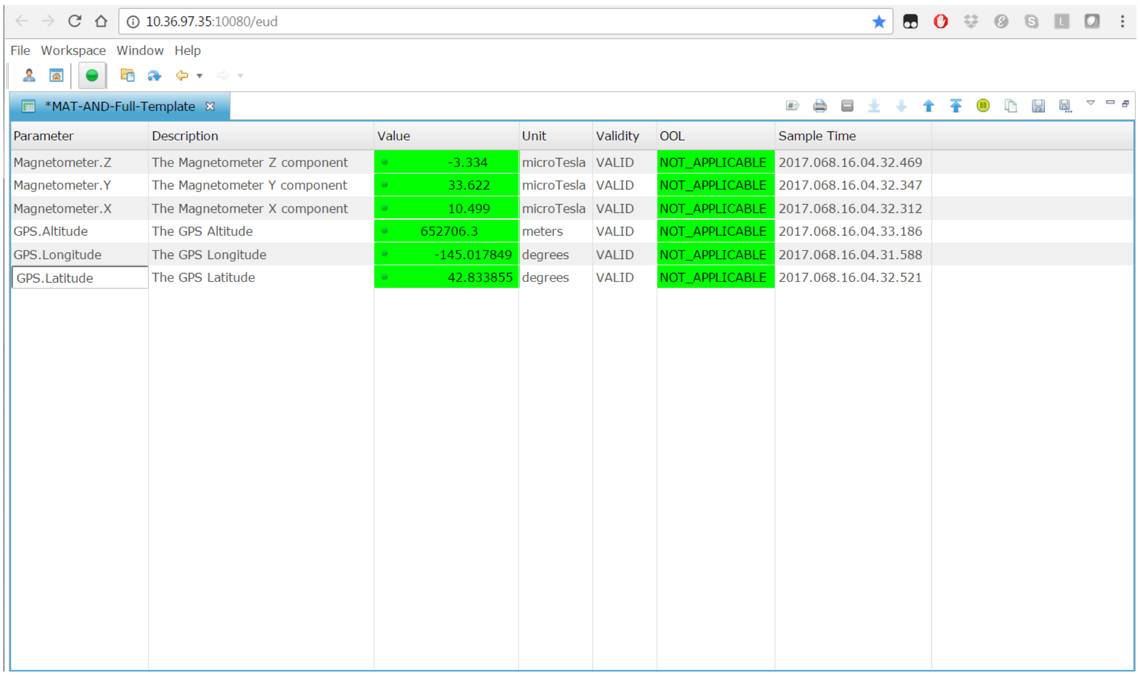


Figure 83: Example of EUD4MO connected to a Dummy App and accessed from a web browser.

ESA's EGOS User Desktop (EUD) provides a framework for M&C User Interfaces for all types of ground segment systems, including ground station backend systems and mission control systems. This not only allows exploiting synergies in the UI development but also facilitates a common look and feel across all ground segment systems. It uses the Eclipse RAP framework which makes it possible to run EUD as web application as is, with only minor adjustments to the code base. [101] [64] [52]



Parameter	Description	Value	Unit	Validity	OOL	Sample Time
Magnetometer.Z	The Magnetometer Z component	-3.334	microTesla	VALID	NOT_APPLICABLE	2017.068.16.04.32.469
Magnetometer.Y	The Magnetometer Y component	33.622	microTesla	VALID	NOT_APPLICABLE	2017.068.16.04.32.347
Magnetometer.X	The Magnetometer X component	10.499	microTesla	VALID	NOT_APPLICABLE	2017.068.16.04.32.312
GPS.Altitude	The GPS Altitude	652706.3	meters	VALID	NOT_APPLICABLE	2017.068.16.04.33.186
GPS.Longitude	The GPS Longitude	-145.017849	degrees	VALID	NOT_APPLICABLE	2017.068.16.04.31.588
GPS.Latitude	The GPS Latitude	42.833855	degrees	VALID	NOT_APPLICABLE	2017.068.16.04.32.521

Figure 84: EUD4MO Parameter values display.

Figure 84 is a screenshot of a view from EUD4MO that allows the visualization of parameter values in a matrix display. It is also possible to represent the received parameter values in a graphical display as the one presented in Figure 85.

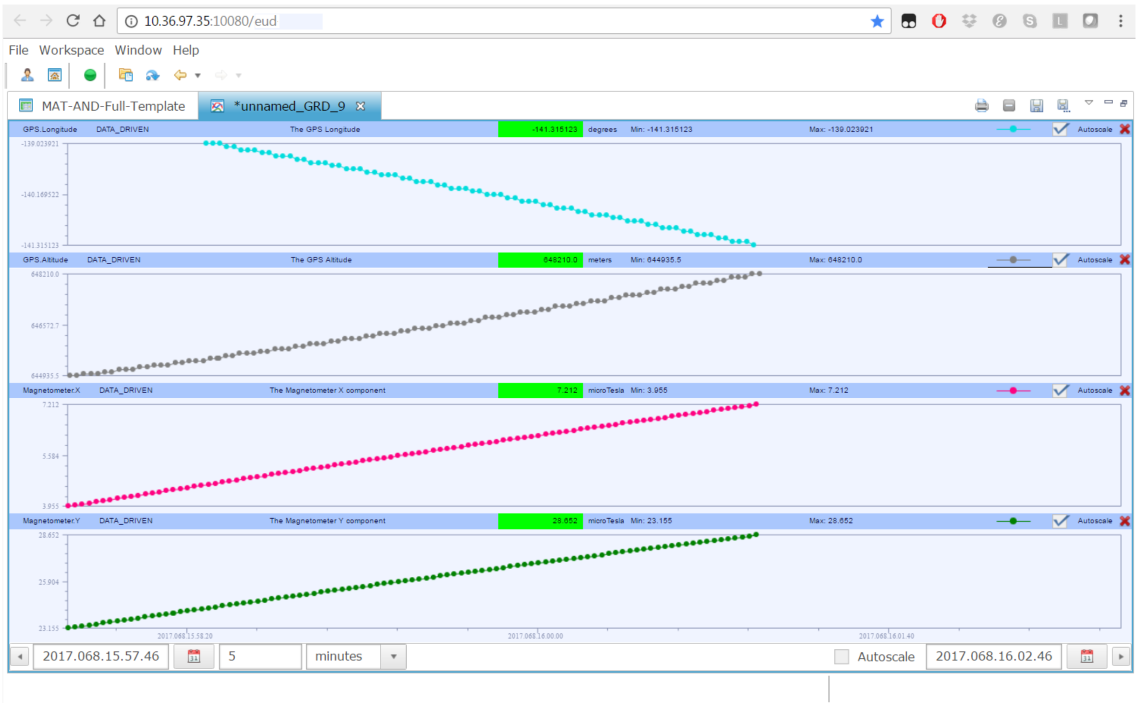
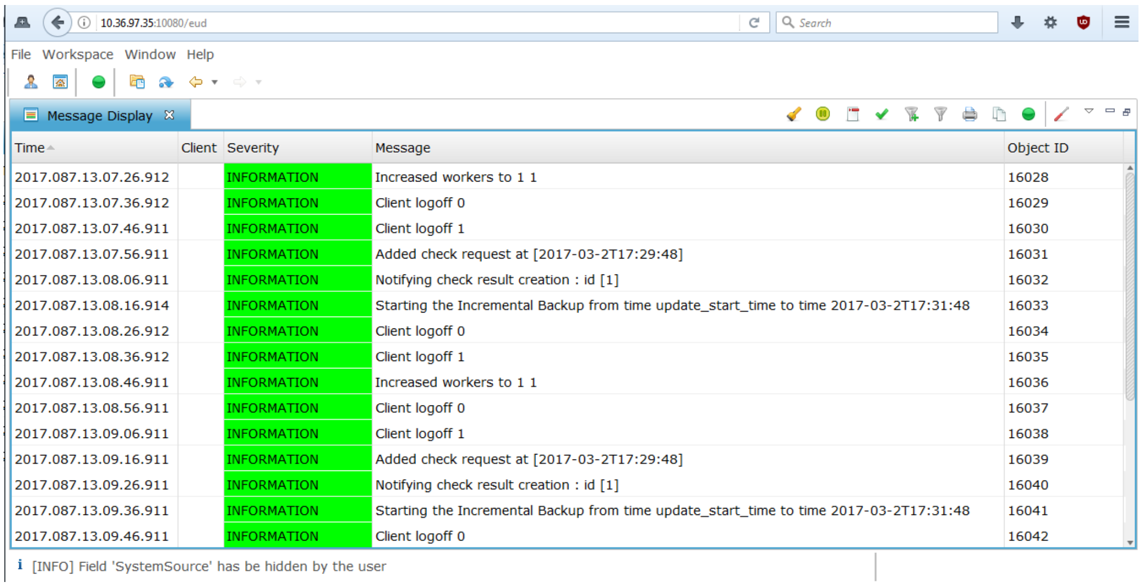

10.36.97.35:10080/eud
File Workspace Window Help
MAT-AND-Full-Template
*unnamed_GRD_9
GPS.Longitude DATA_DRIVEN The GPS Longitude -141.315123 degrees Min: -141.315123 Max: -139.023921 Autoscale
-139.023921
-140.169522
-141.315123
GPS.Altitude DATA_DRIVEN The GPS Altitude 648210.0 meters Min: 644935.5 Max: 648210.0 Autoscale
648210.0
646572.7
644935.5
Magnetometer.X DATA_DRIVEN The Magnetometer X component 7.212 microTesla Min: 3.955 Max: 7.212 Autoscale
7.212
5.584
3.955
Magnetometer.Y DATA_DRIVEN The Magnetometer Y component 28.652 microTesla Min: 23.155 Max: 28.652 Autoscale
28.652
25.904
23.155
2017.068.15.58.20
2017.068.16.00.00
2017.068.16.01.40
2017.068.15.57.46
5
minutes
Autoscale
2017.068.16.02.46

Figure 85: EUD4MO Parameter values plot display.

EUD4MO can also display Alert events in form of a list of messages.

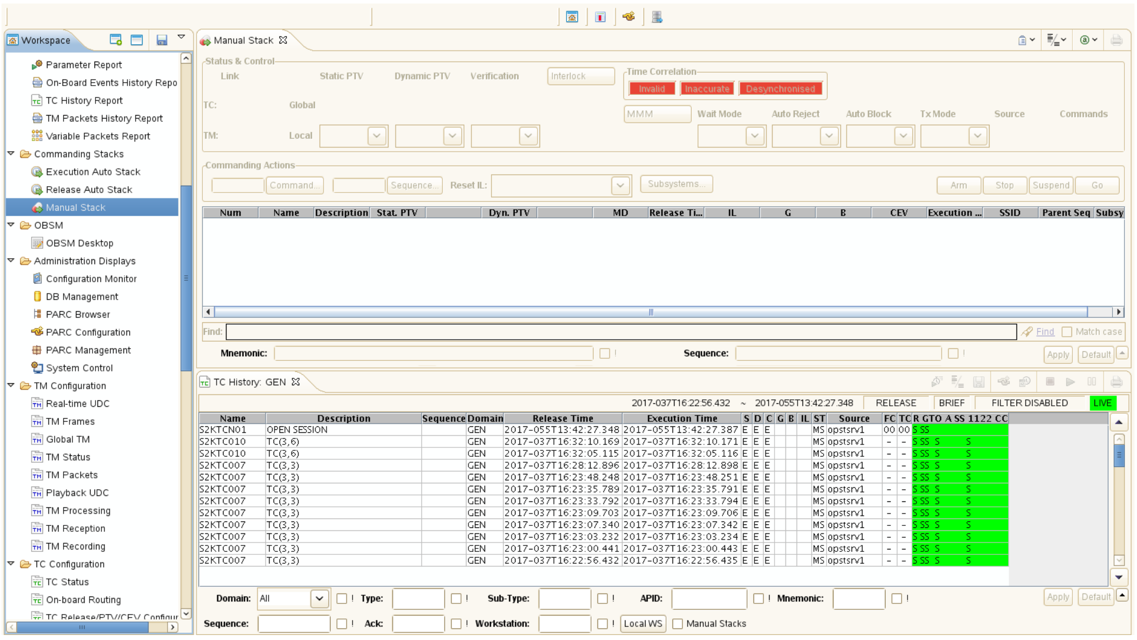


Time	Client	Severity	Message	Object ID
2017.087.13.07.26.912		INFORMATION	Increased workers to 1 1	16028
2017.087.13.07.36.912		INFORMATION	Client logoff 0	16029
2017.087.13.07.46.911		INFORMATION	Client logoff 1	16030
2017.087.13.07.56.911		INFORMATION	Added check request at [2017-03-2T17:29:48]	16031
2017.087.13.08.06.911		INFORMATION	Notifying check result creation : id [1]	16032
2017.087.13.08.16.914		INFORMATION	Starting the Incremental Backup from time update_start_time to time 2017-03-2T17:31:48	16033
2017.087.13.08.26.912		INFORMATION	Client logoff 0	16034
2017.087.13.08.36.912		INFORMATION	Client logoff 1	16035
2017.087.13.08.46.911		INFORMATION	Increased workers to 1 1	16036
2017.087.13.08.56.911		INFORMATION	Client logoff 0	16037
2017.087.13.09.06.911		INFORMATION	Client logoff 1	16038
2017.087.13.09.16.911		INFORMATION	Added check request at [2017-03-2T17:29:48]	16039
2017.087.13.09.26.911		INFORMATION	Notifying check result creation : id [1]	16040
2017.087.13.09.36.911		INFORMATION	Starting the Incremental Backup from time update_start_time to time 2017-03-2T17:31:48	16041
2017.087.13.09.46.911		INFORMATION	Client logoff 0	16042

[INFO] Field 'SystemSource' has be hidden by the user

Figure 86: EUD4MO Alerts display.

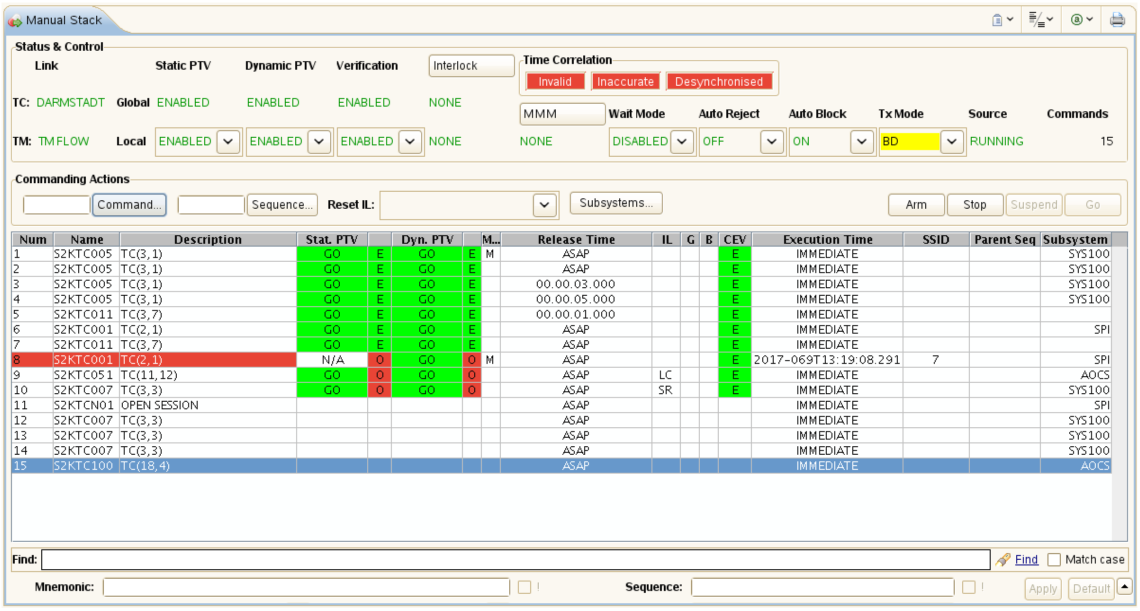
It includes a Manual Stack in order to create stacks of actions which can be invoked using EUD4MO. Figure 87 and Figure 88 present a screenshot of the Manual Stack display.



2017-037T16:22:56.432 ~ 2017-055T13:42:27.348

Name	Description	Sequence	Domain	Release Time	Execution Time	S	D	C	G	B	IL	ST	Source	FC	TC	R GTO A SS 1122 CC
S2KTCN01	OPEN SESSION		GEN	2017-055T13:42:27.348	2017-055T13:42:27.387	E	E	E				MS	opstsrv1	00	00	S SS
S2KTC010	TC(3,6)		GEN	2017-037T16:32:10.169	2017-037T16:32:10.171	E	E	E				MS	opstsrv1	-	-	S SS S S
S2KTC010	TC(3,6)		GEN	2017-037T16:32:05.115	2017-037T16:32:05.116	E	E	E				MS	opstsrv1	-	-	S SS S S
S2KTC007	TC(3,3)		GEN	2017-037T16:28:12.896	2017-037T16:28:12.898	E	E	E				MS	opstsrv1	-	-	S SS S S
S2KTC007	TC(3,3)		GEN	2017-037T16:23:48.248	2017-037T16:23:48.251	E	E	E				MS	opstsrv1	-	-	S SS S S
S2KTC007	TC(3,3)		GEN	2017-037T16:23:35.789	2017-037T16:23:35.791	E	E	E				MS	opstsrv1	-	-	S SS S S
S2KTC007	TC(3,3)		GEN	2017-037T16:23:33.792	2017-037T16:23:33.794	E	E	E				MS	opstsrv1	-	-	S SS S S
S2KTC007	TC(3,3)		GEN	2017-037T16:23:09.703	2017-037T16:23:09.706	E	E	E				MS	opstsrv1	-	-	S SS S S
S2KTC007	TC(3,3)		GEN	2017-037T16:23:07.340	2017-037T16:23:07.342	E	E	E				MS	opstsrv1	-	-	S SS S S
S2KTC007	TC(3,3)		GEN	2017-037T16:23:03.232	2017-037T16:23:03.234	E	E	E				MS	opstsrv1	-	-	S SS S S
S2KTC007	TC(3,3)		GEN	2017-037T16:23:00.441	2017-037T16:23:00.443	E	E	E				MS	opstsrv1	-	-	S SS S S
S2KTC007	TC(3,3)		GEN	2017-037T16:22:56.432	2017-037T16:22:56.435	E	E	E				MS	opstsrv1	-	-	S SS S S

Figure 87: EUD4MO Manual Stack display.



Manual Stack

Status & Control

Link | Static PTV | Dynamic PTV | Verification | Interlock

TC: DARMSTADT Global ENABLED ENABLED ENABLED NONE

TM: TM FLOW Local ENABLED ENABLED ENABLED NONE

Time Correlation: Invalid Inaccurate Desynchronised

MMM	Wait Mode	Auto Reject	Auto Block	Tx Mode	Source	Commands
NONE	DISABLED	OFF	ON	BD	RUNNING	15

Commanding Actions

Command... Sequence... Reset IL: Subsystems... Arm Stop Suspend Go

Num	Name	Description	Stat. PTV		Dyn. PTV		M...	Release Time	IL	G	B	CEV	Execution Time	SSID	Parent Seq	Subsystem
1	S2KTC005	TC(3,1)	GO	E	GO	E	M	ASAP				E	IMMEDIATE			SYS100
2	S2KTC005	TC(3,1)	GO	E	GO	E		ASAP				E	IMMEDIATE			SYS100
3	S2KTC005	TC(3,1)	GO	E	GO	E		00.00.03.000				E	IMMEDIATE			SYS100
4	S2KTC005	TC(3,1)	GO	E	GO	E		00.00.05.000				E	IMMEDIATE			SYS100
5	S2KTC011	TC(3,7)	GO	E	GO	E		00.00.01.000				E	IMMEDIATE			
6	S2KTC001	TC(2,1)	GO	E	GO	E		ASAP				E	IMMEDIATE			SPI
7	S2KTC011	TC(3,7)	GO	E	GO	E		ASAP				E	IMMEDIATE			
8	S2KTC001	TC(2,1)	N/A	O	GO	O	M	ASAP				E	2017-069T13:19:08.291	7		SPI
9	S2KTC051	TC(11,12)	GO	O	GO	O		ASAP	LC			E	IMMEDIATE			AOCS
10	S2KTC007	TC(3,3)	GO	O	GO	O		ASAP	SR			E	IMMEDIATE			SYS100
11	S2KTCN01	OPEN SESSION						ASAP					IMMEDIATE			SPI
12	S2KTC007	TC(3,3)						ASAP					IMMEDIATE			SYS100
13	S2KTC007	TC(3,3)						ASAP					IMMEDIATE			SYS100
14	S2KTC007	TC(3,3)						ASAP					IMMEDIATE			SYS100
15	S2KTC100	TC(18,4)						ASAP					IMMEDIATE			AOCS

Find: Find Match case

Mnemonic: Sequence: Apply Default

Figure 88: EUD4MO Manual Stack display in more detail.

5.2.4. External NMF Ground aplications

External NMF Ground applications can connect to the Ground MO Proxy in order to reach the spacecraft without being hosted in ESOC's premises. This allows external ground systems developed by experimenters to be deployed anywhere around the world and still be able to reach the spacecraft using a TCP/IP connection between the Ground MO Proxy and the external system. All the advantages of the Ground MO Proxy will then be available.

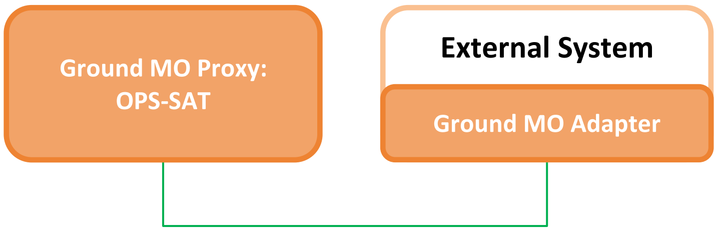


Figure 89: EUD4MO Manual Stack display in more detail.

5.3. Transport Binding

An NMF Mission has to specify the transport bindings used between the applications. As mentioned in chapter 2, the MAL-TCP/IP Transport Binding was selected to be part of NMF Generic Model and therefore it is a default transport available in all NMF Composites. For OPS-SAT, the default transport binding is not sufficient because the ground-to-space link is done using space packets. [48]

The MAL-TCP/IP Transport Binding is used for IPC between the NMF Apps and the NanoSat MO Supervisor. Additionally, it is used on ground between the Ground MO Proxy and other NMF Ground applications. In Figure 82 these connections are represented in green. [47]

The MAL-SPP Transport Binding and the Binary Encoding are used by the NMF Apps and also by the NanoSat MO Supervisor in order to exchange data with consumers on the ground-to-space link. On ground, it is used by the Ground MO Proxy where its protocol bridge allows other NMF Ground applications to connect to the spacecraft using to the default MAL-TCP/IP transport binding.

In OPS-SAT, the MAL-SPP Transport Binding does not completely follow the CCSDS standard. There are 2 octets being appended at the end of the Space Packet containing the CRC checksum of the message. Consequently, the Packet Data Length of the Space Packet Primary Header is incremented by 2.

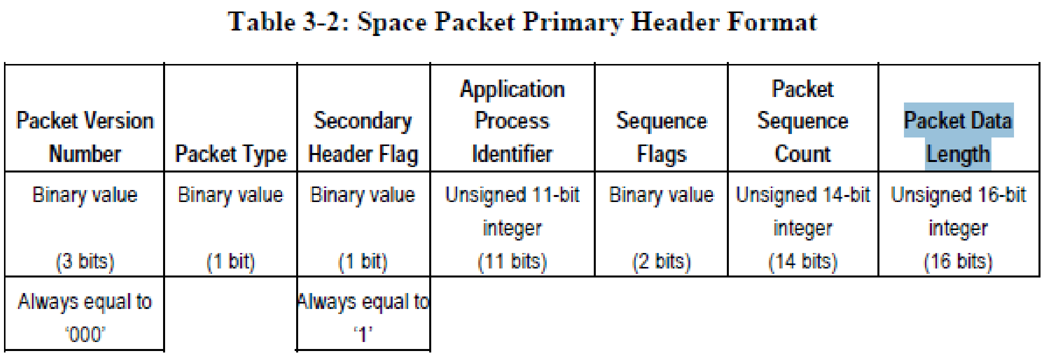


Table 3-2: Space Packet Primary Header Format

Packet Version Number	Packet Type	Secondary Header Flag	Application Process Identifier	Sequence Flags	Packet Sequence Count	Packet Data Length
Binary value (3 bits)	Binary value (1 bit)	Binary value (1 bit)	Unsigned 11-bit integer (11 bits)	Binary value (2 bits)	Unsigned 14-bit integer (14 bits)	Unsigned 16-bit integer (16 bits)
Always equal to '000'		Always equal to '1'				

Figure 90: Space Packet Primary Header. [47]

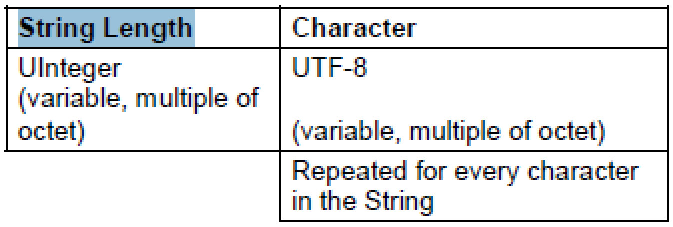
Additionally, some customizations on the Binary Encoding had to be done for ESA's OPS-SAT mission because third-party software could not operate with the default encoding. The Identifier, String, and URI MAL Data Types do not follow the Binary Encoding standard prescriptions. The following requirement is not met: [47]

5.21.3. The field 'String Length' shall be encoded as a UInteger (see 5.18).

In the standard, a UInteger MAL Data Type is encoded as an unsigned 32-bit Integer according to requirement: [47]

5.18.2. If the MCP VARINT_SUPPORTED is FALSE, then a MAL::UInteger shall be encoded as an Unsigned 32-bit Integer (see 5.25).

However, for OPS-SAT the UInteger MAL Data Type present in the 'String Length' field is being encoded as an unsigned 16-bit integer.



String Length	Character
UInteger (variable, multiple of octet)	UTF-8 (variable, multiple of octet)
	Repeated for every character in the String

Figure 91: Structure of the Identifier, String, and URI MAL Data Types after being encoded as Binary. [47]

The Binary Encoding allows configuring the Mapping Configuration Parameters (MCPs) in order to optimize the amount of data that is being exchanged by the transport. An out-of-band agreement was defined by all parties involved in the OPS-SAT project that specify the MCPs in order to save bandwidth.

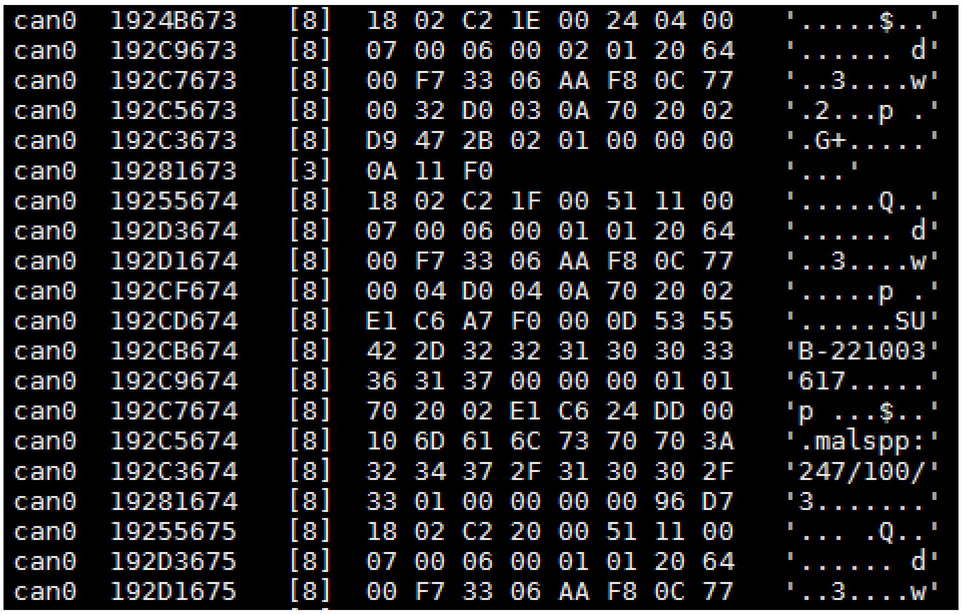
The implementation of the MAL-SPP Transport Binding used by the NanoSat MO Framework was developed by DLR and it includes the additional changes mentioned above for OPS-SAT. The implementation allows MCPs to be defined per Application Process Identifier (APID) however a default configuration was defined in order to avoid the hassle of having to define the MCPs to every single NMF App.

In OPS-SAT every experimenter has a dedicated APID that is pre-allocated by ESA. If an experimenter decides to develop an NMF App, then it will be using the APID that was assigned to the experiment. By default, all NMF Apps parse a configuration properties file (provider.properties) in order to set the APID.

After encoding the messages into space packets, they can be sent and received via different mechanisms on the layers below. On the spacecraft bus, the space packets are exchanged between the units using CAN Fragmentation Protocol (CFP), a dedicated protocol specifically defined for OPS-SAT that runs on top of the CAN bus. On ground, the space packets are exchanged between the machines using TCP/IP.

The CAN Frames have a maximum size of 8 bytes on the Data field and therefore messages longer than that have to be fragmented. CFP supports fragmentation of messages up to 32 segments which extends the limitation to 256 bytes messages. Additionally, CFP provides the possibility of having retransmission of messages that might have been lost, and finally, it also defines a source and destination fields in order to select the nodes that are sending and receiving the messages. [98]

Under high load it was observed that the CAN bus was having problems. With Linux kernel version 3.14 the mityARM would stall entirely. After updating to 3.16, the CAN bus was able to send messages without stalling however it was observed that some of the messages were being corrupted in two of the bytes of the Data field. The problem was found to be on the Linux Kernel driver of the CAN bus and a solution was found online. The patch was provided to the manufacturers of the device and informed of the problem. [102]



```
can0  1924B673   [8]  18 02 C2 1E 00 24 04 00   '.....$..'
can0  192C9673   [8]  07 00 06 00 02 01 20 64   '...... d'
can0  192C7673   [8]  00 F7 33 06 AA F8 0C 77   '..3....w'
can0  192C5673   [8]  00 32 D0 03 0A 70 20 02   '.2...p .'
can0  192C3673   [8]  D9 47 2B 02 01 00 00 00   '.G+.....'
can0  19281673   [3]  0A 11 F0                  '...'
can0  19255674   [8]  18 02 C2 1F 00 51 11 00   '.....Q..'
can0  192D3674   [8]  07 00 06 00 01 01 20 64   '...... d'
can0  192D1674   [8]  00 F7 33 06 AA F8 0C 77   '..3....w'
can0  192CF674   [8]  00 04 D0 04 0A 70 20 02   '.....p .'
can0  192CD674   [8]  E1 C6 A7 F0 00 0D 53 55   '......SU'
can0  192CB674   [8]  42 2D 32 32 31 30 30 33   'B-221003'
can0  192C9674   [8]  36 31 37 00 00 00 01 01   '617.....'
can0  192C7674   [8]  70 20 02 E1 C6 24 DD 00   'p ...$..'
can0  192C5674   [8]  10 6D 61 6C 73 70 70 3A   '.malspp:'
can0  192C3674   [8]  32 34 37 2F 31 30 30 2F   '247/100/'
can0  19281674   [8]  33 01 00 00 00 00 96 D7   '3.......'
can0  19255675   [8]  18 02 C2 20 00 51 11 00   '... .Q..'
can0  192D3675   [8]  07 00 06 00 01 01 20 64   '...... d'
can0  192D1675   [8]  00 F7 33 06 AA F8 0C 77   '..3....w'
```

Figure 92: Sniff of CAN messages on the bus.

Although the space packets are exchanged between the machines on ground using TCP/IP, reliable transmission of data to the spacecraft is not guaranteed. It is important to mention this because it might induce an experimenter into wrong assumptions.

5.4. Platform adapters integration with OPS-SAT

ESA's OPS-SAT mission uses the Java implementation presented in chapter 3 which includes the implementation of all the Platform services. Figure 93 represents how the Platform services take advantage of the adapter pattern in order to support different units.

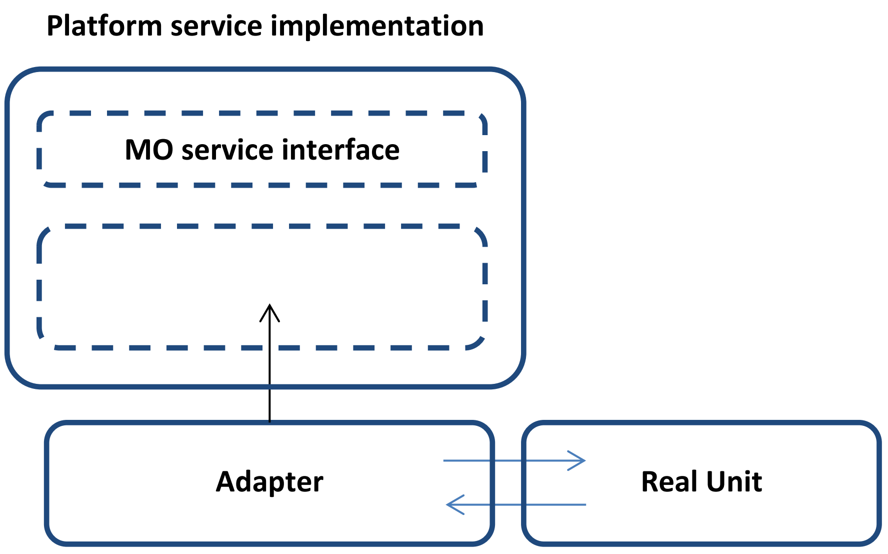


Figure 93: Generic Platform service implementation and its respective adapter to the real unit.

This section presents in more detail the integration details of the Platform adapters with the OPS-SAT peripherals. There are 5 payload peripherals in OPS-SAT that can be used by the experimenters: Camera, Optical Data Receiver, SDR, FineADCS, and GPS.

5.4.1. Camera service adapter

The Camera service adapter for OPS-SAT is connected to the Camera payload device via a USB virtual serial port. OPS-SAT's Camera is a ST200 from Hyperion Technologies and it has a power consumption of 650 mW in nominal mode, a mass of 42 grams, and an update rate of 5 Hz. [103]

Upon initialization, the adapter checks if there is a serial port device camera and if so, it attempts to connect to it. If the device is not available, the Camera service is still started however if for example, a consumer tries to take a picture, then the service will return an error.

When the camera is functional and a consumer takes a picture, the adapter will double-check if the device is connected and then, send the instructions to configure it and take the picture. Finally, it copies the content of the image that is stored as an external memory device into a Picture object and sends it back to the consumer.

Figure 94: ST200 from Hyperion Technologies. [103]

5.4.2. GPS service adapter

The GPS service adapter for OPS-SAT is connected to the GPS payload device via an MO interface exposed by the Nanomind device on the CAN bus. OPS-SAT's GPS is part of the OEM615 Family from NovAtel and it has low power consumption, dual frequency (L1, L2, and L2C for GPS and GLONASS), and supports multi-constellations (E1 for Galileo and B1 for BeiDou). [104]

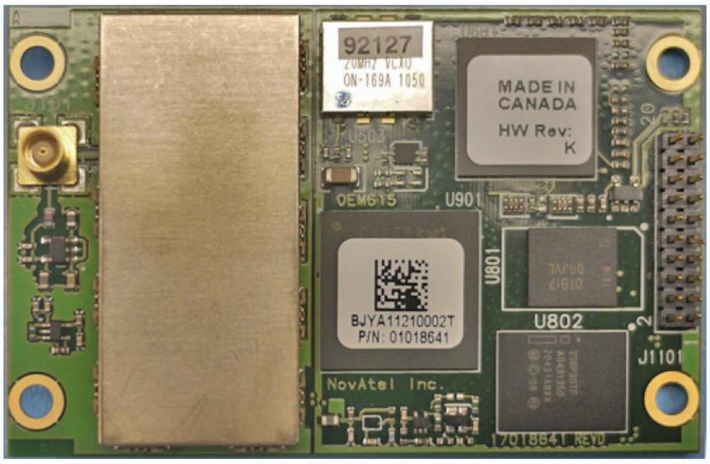


Figure 95: OEM615 from NovAtel. [104]

The Nanomind is connected to the GPS Unit via a UART connection that allows requesting information from a set of commands. Then, the Nanomind device exposes a GPS service for interacting with the GPS unit. This service can be consumed from ground or by the Experimental Platform. The setup can be visualized on Figure 96.

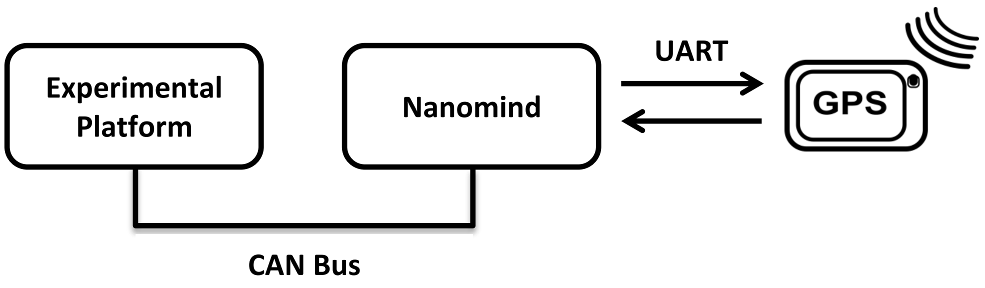


Figure 96: GPS access from the Experimental Platform. [104]

Although the GPS service from the Nanomind has the same name as the GPS service from the Platform services, they are completely different. The GPS service from the Nanomind includes one single operation as it can be seen in Table 14.

Area Identifier	Service Identifier	Area Number	Service Number	Area Version
OPSSAT_PF	GPS	75	21	1
Interaction Pattern	Operation Identifier	Operation Number	Support in Replay	Capability Set
INVOKE	getGPSData	1	No	1

Table 14: GPS service exposed by the Nanomind.

The getGPSData operation allows a consumer to receive requested data from the GPS device (for example, GPGSA data frame). The 'command' field must contain a request encoded as an ASCII string to be sent to the GPS receiver. Then, the returned 'data' field holds the GPS data frame encoded as a string of ASCII characters.

Operation Identifier	getGPSData	
Interaction Pattern	INVOKE	
Pattern Sequence	Message	Body Type
IN	INVOKE	command : (MAL::Blob)
OUT	ACK	
OUT	RESPONSE	data : (MAL::Blob)

Table 15: GPS service getGPSData operation.

5.4.3. Autonomous ADCS service adapter

The Autonomous ADCS service adapter for OPS-SAT is connected to the ADCS payload device via I²C. OPS-SAT's ADCS is an iADCS-100 from Berlin Space Technologies and consists of a star tracker, gyro module, reaction wheels, and magnetorquers, with the possibility to integrate external sensors such as sun sensors. [105]

It incorporates the ADCS algorithms from the LEOS platform and allows full ADCS functionality including nadir pointing as well as autonomous target acquisition and tracking. [105]

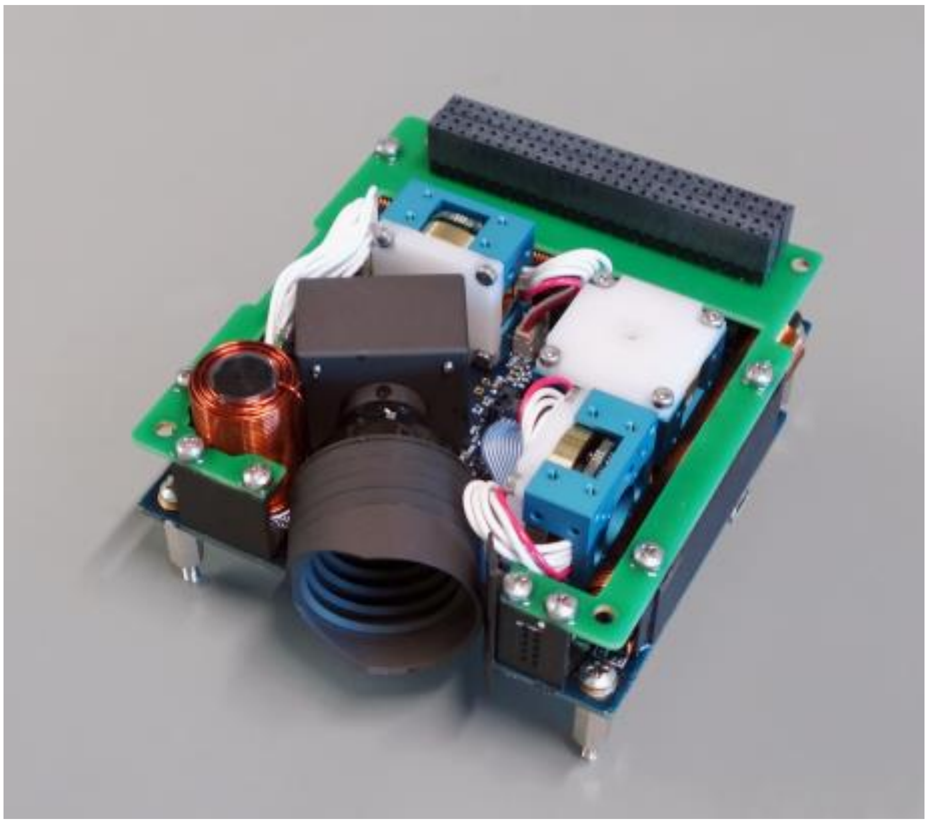


Figure 97: iADCS-100 from Berlin Space Technologies. [105]

5.4.4. Other services' adapters

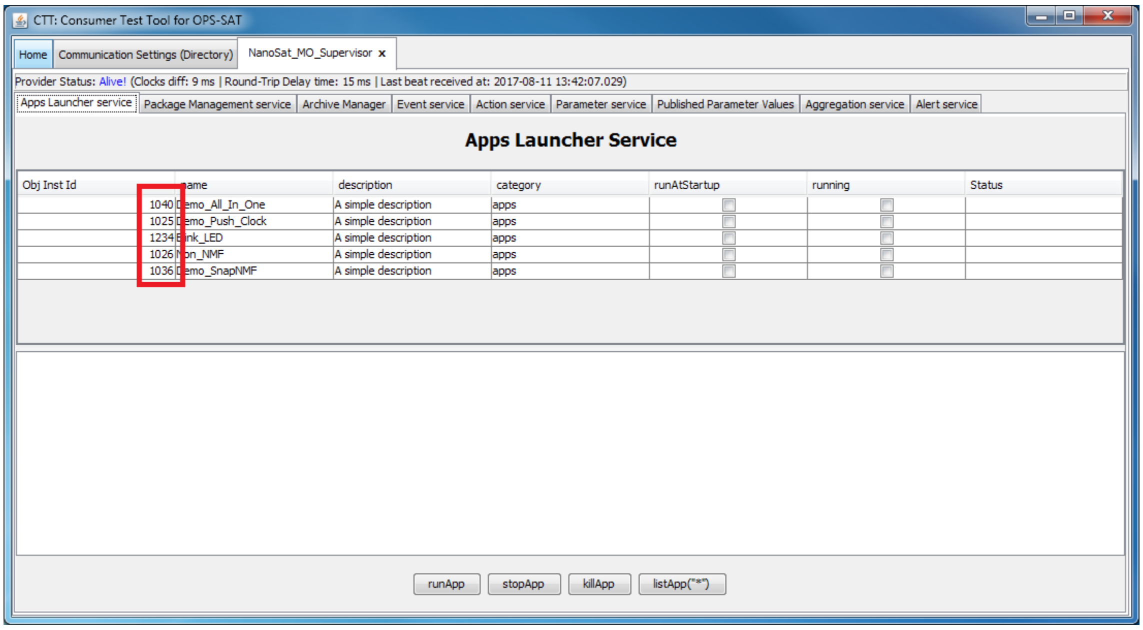
The Software-defined Radio service adapter and the Optical Data Receiver service adapter were not implemented because the units were not available during this research.

5.5. Customization of other services

Some services from NMF's Java implementation had to be customized to cope with ESA's OPS-SAT mission. This involves a requirement for the management of applications based on the APID for the Apps Launcher service and an optimization to reduce the amount of data transmitted on the space-to-ground link by the Directory service upon requesting a lookup.

The Apps Launcher service needs to be able to start, stop, and kill the applications from their respective APID that was assigned by the OPS-SAT's flight control team. The service uniquely identifies applications by an object instance identifier and therefore the service had to be tweaked to parse the APID value available in the provider.properties file and define the application's object instance identifier to be the same as the APID value.

In OPS-SAT there are three types of experiments: applications, OS images, and FPGA images. Originally, there were different mechanism for starting and stopping each type however these have all been simplified and now they can be started from a script that is executed by the Apps Launcher service. The Apps Launcher executes the script on the runAppLin.sh file after receiving the start operation command for that application. [106]



Provider Status: Alive! (Clocks diff: 9 ms | Round-Trip Delay time: 15 ms | Last beat received at: 2017-08-11 13:42:07.029)

Apps Launcher Service

Obj Inst Id	name	description	category	runAtStartup	running	Status
1040	Demo_All_In_One	A simple description	apps	☐	☐	
1025	Demo_Push_Clock	A simple description	apps	☐	☐	
1234	Blink_LED	A simple description	apps	☐	☐	
1026	Non_NMF	A simple description	apps	☐	☐	
1036	Demo_SnapNMF	A simple description	apps	☐	☐	

Figure 98: Apps Launcher service displaying applications with their respective object instance identifier assigned per APID. [105]

The Directory service holds a list of providers and their respective service connection details. The NanoSat MO Supervisor includes a Central Directory service that exposes all the existing providers on the NanoSat segment. When an NMF App or the NanoSat

MO Supervisor is started in OPS-SAT, its services are initialized and become operational for two different transports: TCP/IP, and SPP over CFP.

The TCP/IP transport is used for IPC and therefore this information is not relevant to a ground consumer. Using the AGSA Flatsat from chapter 6, it was observed that the majority of the data that was being exchanged upon a lookup on the Directory service was composed of strings of URIs with the TCP/IP connection details.

In order to reduce the amount of exchanged data, a small hack was introduced in the lookupProvider operation. A ServiceFilter object is passed as an argument to the operation and inside that object there's the field sessionName that can be abused. The hack consists of checking the sessionName field and if there's a string with the value "s2g", then the lookupProvider operation only returns the SPP connection details.

The "NMF Package" concept was theorized during the research process and a prototype was implemented that allows the installation, upgrade, and uninstallation of packages for demonstration. The Package Management service for OPS-SAT uses this prototype.

Chapter 6

6. VALIDATION AND PERFORMANCE

The verification of the NMF Java implementation for the OPS-SAT mission is presented in this chapter. A Flatsat containing OPS-SAT's hardware was assembled for the verification of the software in the AGSA Lab, which is one of ESOC's software laboratories. Then, OPS-SAT's SVT-0 is presented and its respective results. Finally, some performance metrics are also presented in this chapter for two different machines.

6.1. AGSA Lab

The Advanced Ground Software Applications Laboratory (AGSA Lab) is an ESOC laboratory for the development of software applications and prototyping of new concepts for the mission operations of future space assets. The AGSA Lab does research activities related with standardization, nanosatellite operations, robotic operations, and new virtual reality operational concepts.

Trainees, young graduates, and PhD students use the laboratory infrastructure for the development of projects during the traineeship. The laboratory includes a server, a Flatsat, a generic rover, a Virtual Reality headset, and many computer resources that allow them to try their developed software. [22] [96] [107]

The server in the AGSA Lab allows the simultaneous execution of virtual machines and in one of them an instance of Jenkins was configured to be continuously running. Every time code is pushed to Gitlab, Jenkins is triggered to start the compilation process, which then triggers the subsequent steps of the pipeline. At the end, the artifacts are deployed on the AGSA Flatsat.

6.1.1. AGSA Flatsat

A Flatsat is a spacecraft model that allows functional verification and validation of the software or hardware on a flat surface. In the AGSA Lab, a Flatsat was assembled during the research process that acts as an OPS-SAT hardware emulator where it is possible to test and validate the software running on the Nanomind and also on the Experimental Platform. Figure 99 shows a picture taken during the testing of software.

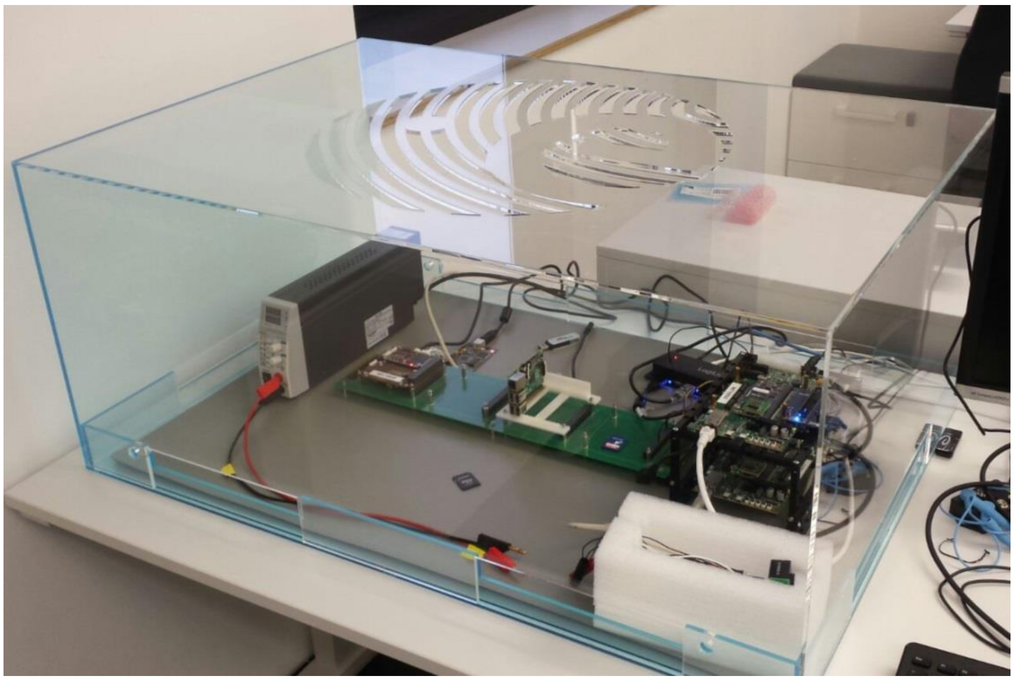


Figure 99: AGSA Flatsat.

In terms of hardware, the AGSA Flatsat includes the following material:

- a Nanomind A3200 from GOMspace
- a OEM615 from NovAtel
- two MitySOM-5CSX Development Kits from Critical Link
- a ST200 from Hyperion Technologies
- Auxiliary material (Raspberry Pis, cables, etc)

The AGSA Flatsat provides the infrastructure to test the developed NMF-related software for OPS-SAT's mission including NanoSat MO Supervisor, Ground MO Proxy, and EUD4MO. Additionally, it also enables the possibility to test some of the applications in the OPS-SAT Ground Data Systems node. The applications are described in chapter 5.

The hardware was assembled identically to OPS-SAT's spacecraft bus as presented in Figure 79. The CCSDS Engine is an important device in OPS-SAT because it enables the spacecraft to exchange information with ground but it was not possible to acquire the same hardware unit for the AGSA Flatsat therefore an alternative solution had to be found. Considering that there were two MitySOM-5CSX available, it was decided to use one of them to impersonate the CCSDS Engine and dedicated software was developed to mimic its behaviour.

The Advanced and Configurable CCSDS Engine Software Simulator (ACCESS) provides communication access from ground applications to the AGSA Flatsat board by simulating OPS-SAT's CCSDS Engine. ACCESS exposes a TCP/IP socket for the exchange space packets with ground applications and connects to the CAN bus in order to route the data from ground to the correct node of the bus. ACCESS runs on one of the MitySOM-5CSX of the AGSA Flatsat.

The Camera device is plugged into the Experimental Platform that runs OPS-SAT's Standard Image and the GPS device (OEM615 from NovAtel) is plugged into the Nanomind. AGSA's Flatsat includes a main CAN bus just like OPS-SAT. Figure 100 shows the software and the connected hardware devices.

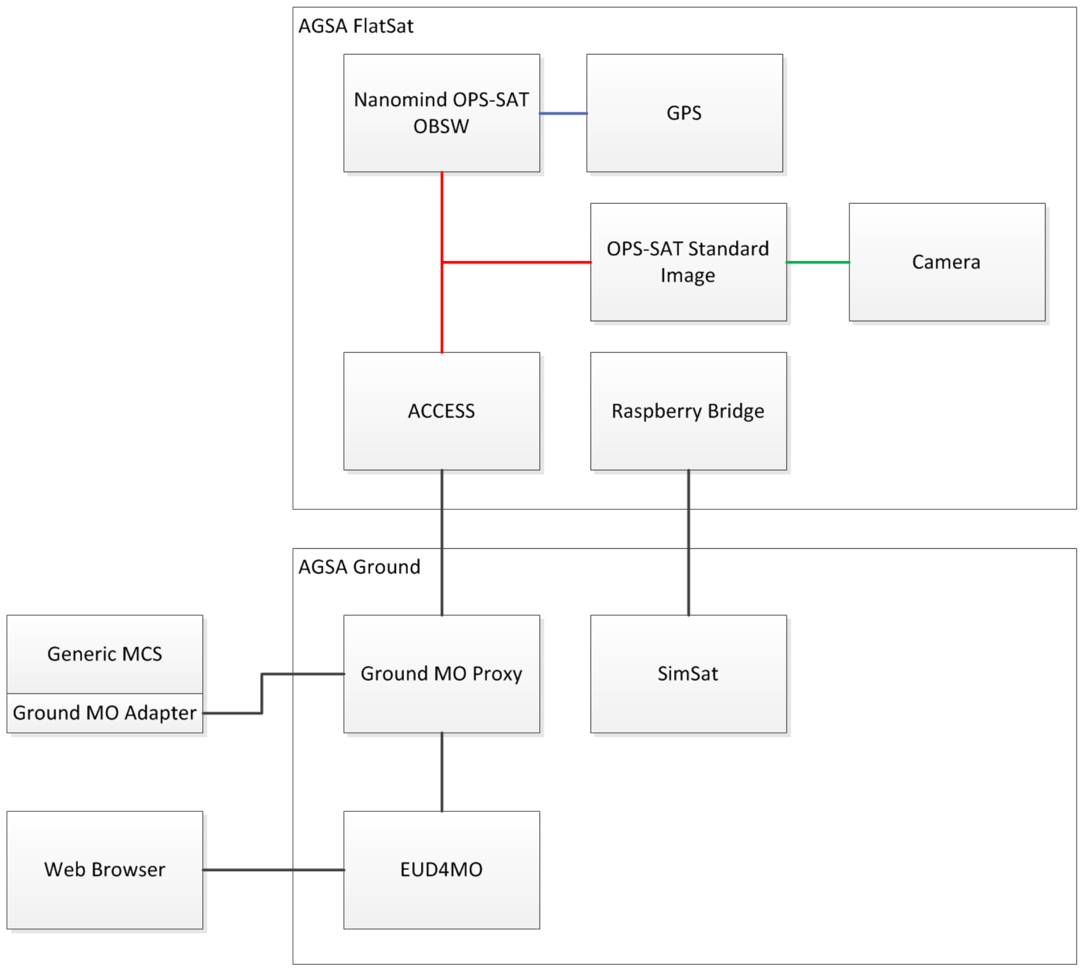


Figure 100: AGSA's Flatsat Software and Hardware diagram.

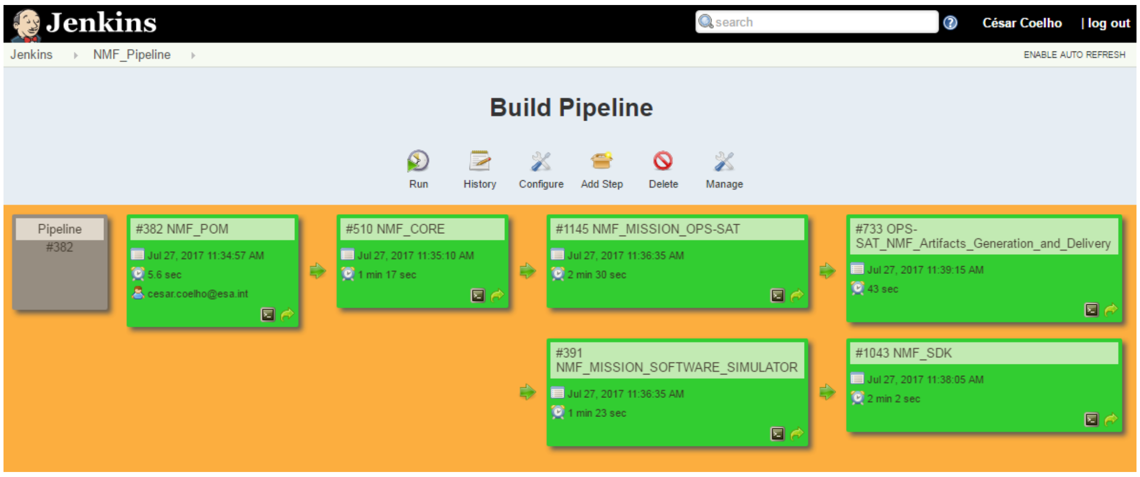
The AGSA Flatsat was incrementally built and therefore the integration of hardware was progressive. It can still be enhanced by introducing a simulator of OPS-SAT's ADCS unit by using a Raspberry Pi connected to one of the MitySOM-5CSX via I^2C. The model can be running on the Raspberry Pi itself, or on server bridged by the Raspberry Pi. Although the "Raspberry Bridge" and the SimSat components are represented in Figure 100, they are not connected to any other device.

6.1.2. Jenkins Pipeline

As previously mentioned, the AGSA Lab has a server running an instance of Jenkins that builds the code every time that it is pushed to the repository. Jenkins was configured to have a pipeline that builds the next stage after the on-going stage finishes successfully.

The pipeline includes several stages, which can be visualized in Figure 101:

- The first stage builds the NMF POM file that includes all the version numbers for the automatic resolution of dependencies by other projects.
- The second stage builds the NMF Core that is the Java implementation presented in chapter 3.
- The third stage builds the NMF Mission dedicated to a specific mission implementation of the NMF.



Pipeline #382

#382 NMF_POM
Jul 27, 2017 11:34:57 AM
5.6 sec
cesar.coelho@esa.int

#510 NMF_CORE
Jul 27, 2017 11:35:10 AM
1 min 17 sec

#1145 NMF_MISSION_OPS-SAT
Jul 27, 2017 11:36:35 AM
2 min 30 sec

#733 OPS-SAT_NMF_Artifacts_Generation_and_Delivery
Jul 27, 2017 11:39:15 AM
43 sec

#391 NMF_MISSION_SOFTWARE_SIMULATOR
Jul 27, 2017 11:36:35 AM
1 min 23 sec

#1043 NMF_SDK
Jul 27, 2017 11:38:05 AM
2 min 2 sec

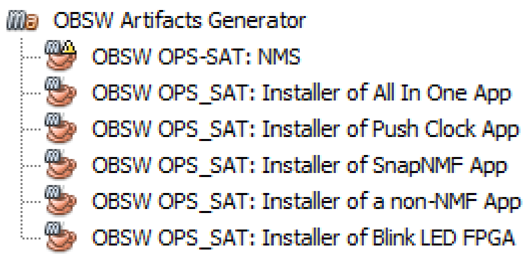
Figure 101: Continuous Integration and Continuous Delivery with Jenkins.

As presented in the dissertation, two NMF Missions were implemented, the first one for OPS-SAT's mission and the second one for an OPS-SAT Software Simulator. Additional stages were configured after the mission-specific stages.

The NMF SDK uses the Software Simulator for the Playground environment and for the Monolithic Provider demo therefore it builds only after. These are presented in chapter 4.

A stage after building the OPS-SAT's NMF Mission generates all the necessary files and folders in order to run the software for that mission. Additionally, it also transfers some of these artifacts to the AGSA Flatsat which allows it to have the latest version ready to be started.

One of the projects can automatically generate the folder's structure presented in section 5.1.1. This project is in the last stage, it is called "OBSW Artifacts Generator" and it includes a set of projects inside, one for the NanoSat MO Supervisor and one for each application to be "installed". This can be visualized in Figure 102.



```
OBSW Artifacts Generator
├── OBSW OPS-SAT: NMS
├── OBSW OPS_SAT: Installer of All In One App
├── OBSW OPS_SAT: Installer of Push Clock App
├── OBSW OPS_SAT: Installer of SnapNMF App
├── OBSW OPS_SAT: Installer of a non-NMF App
└── OBSW OPS_SAT: Installer of Blink LED FPGA
```

Figure 102: OPS-SAT Deployment generator project.

6.2. OPS-SAT SVT-0

System Validation Tests (SVTs) are usually performed to ensure the operability and functionality of the complete system, including both ground and space segments. During the research, OPS-SAT's SVT-0 took place on the first quarter of 2017.

The OPS-SAT's NMF Mission was deployed on the SVT-0 in order to validate the starting, stopping, and killing of three different experiments: NMF App experiment, FPGA experiment, and non-NMF App experiment. These were tested and performed successfully, however it was observed that the connectivity between the experimental platform and the ground segment had problems which caused CTT to sporadically crash.

The difference between the AGSA Flatsat and the SVT-0 Flatsat is that the former is more focused on the testing phase during software development while the latter was intended to validate the different systems of OPS-SAT operating together. Although OPS-SAT's NMF Mission had been previously tested in the AGSA Flatsat, SVT-0 gave a positive confirmation that it can also operate together with the other systems.

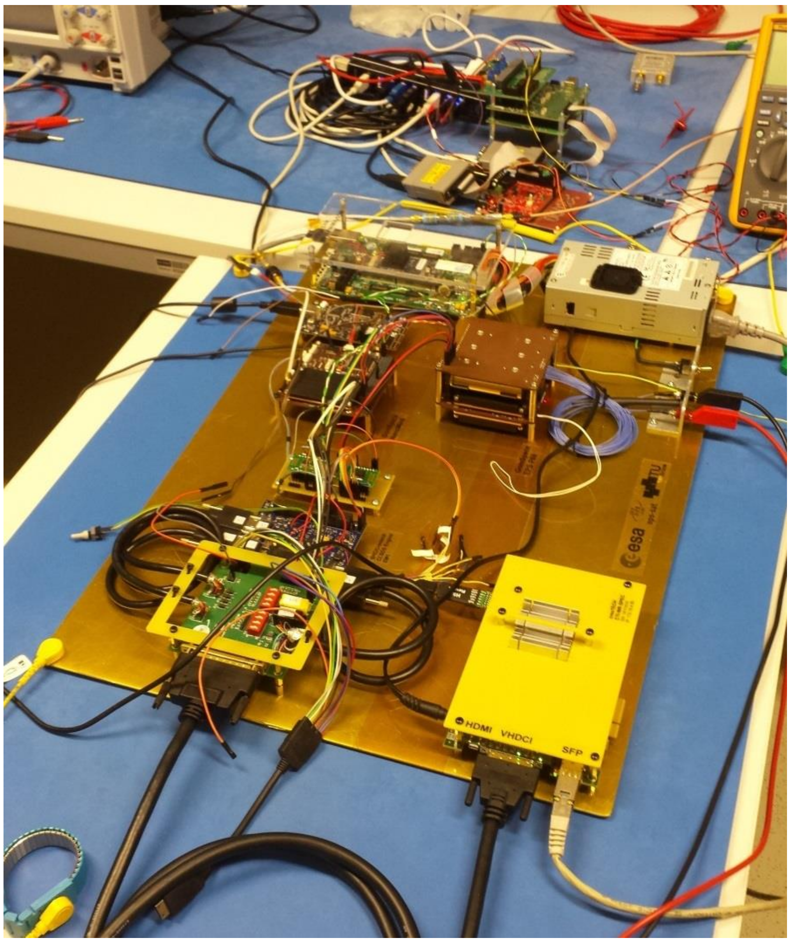


Figure 103: SVT-0 Flatsat.

6.3. Performance Metrics

Performance metrics are measurements of the software performance parameters for certain activities under certain conditions. A set of activities with different conditions were defined and their performance parameters were measured. Some of the parameters have different values for different trials (for example: time) therefore the test was repeated multiple times and the mean value of the samples was calculated.

Two different machines were selected in order to perform the tests:

- Workstation computer
- OPS-SAT's Experimental Platform: MitySOM-5CSX

The workstation computer was selected because this is a powerful machine with easy access, includes all the development tools and it is where most of the code was developed. The second selected machine was OPS-SAT's Experimental Platform, the MitySOM-5CSX because it is the selected target machine for NMF's Java implementation presented in chapter 3 and it is available on the AGSA Flatsat presented in section 6.1.1. The technical specifications of both machines are presented in Table 16.

	Workstation	**MitySOM-5CSX**
Operating System	Microsoft Windows 7 Enterprise	Linux mitysom-5csx (Yocto)
RAM	16.0 GB	1.0 GB
Processor	Intel Core i5-6200U	ARMv7 Processor rev 0 (v7l)
Processor Architecture	x64	ARM
Clock rate	2.30 GHz	925 MHz
Number of cores	2	2
Logical Processors	4	-

Table 16: Technical specifications of the two select machines.

The memory plots and thread plots in this section were collected using NetBeans Profiler while running on the workstation machine mentioned above. [68]

The objective of this section is to provide some performance metrics for future benchmark of other NMF Apps and/or to provide a rough estimate of the performance expectations. The following figures were probed:

- Memory usage
- Initialization time
- COM Archive database storage size
- Traffic volume on the CAN Bus

A dedicated NMF App, the “Benchmark App”, was developed in order to have a single application for the performance metrics that can be used in the different machines.

Two parameters were defined with the following names and respective descriptions:

- Periodic_Parameter: "A periodic parameter with a double value."
- COM_Archive.size: "The COM Archive size."

Two actions were defined with the following names and respective descriptions:

- StoreAggregations: "Stores NUMBER_OF_OBJS aggregation definition objects in the COM Archive."
- StoreParameters: " Stores NUMBER_OF_OBJS parameter value objects in the COM Archive."

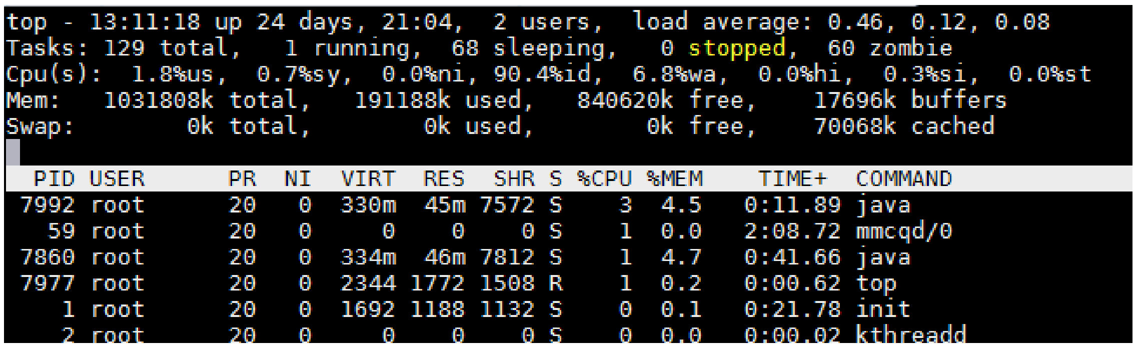
In terms of internal activity, only the Heartbeat service is periodically generating a beat.

The NanoSat MO Connector implementation used for the “Benchmark App” is the same for other NMF Apps and therefore its performance is not expected to change from one app to another as the logic is the same.

6.3.1. NMF App memory usage

The memory usage of the Benchmark App was probed not only from an Operating System point of view but also from a JVM point of view. The memory allocated in the system by the JVM is different from the memory allocated by application running inside it therefore it is relevant to probe both.

It is also important to mention that the memory taken by the JVM is system dependent and that there are parameters that can be configured to set the initial Java heap size (-Xms), the maximum Java heap size (-Xmx), and the java thread stack size (-Xss). [108]



```
top - 13:11:18 up 24 days, 21:04,  2 users,  load average: 0.46, 0.12, 0.08
Tasks: 129 total,   1 running,  68 sleeping,   0 stopped,  60 zombie
Cpu(s):  1.8%us,  0.7%sy,  0.0%ni, 90.4%id,  6.8%wa,  0.0%hi,  0.3%si,  0.0%st
Mem:   1031808k total,   191188k used,   840620k free,    17696k buffers
Swap:        0k total,        0k used,        0k free,    70068k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
 7992 root      20   0  330m  45m 7572 S    3  4.5   0:11.89 java
   59 root      20   0     0    0    0 S    1  0.0   2:08.72 mmcqd/0
 7860 root      20   0  334m  46m 7812 S    1  4.7   0:41.66 java
 7977 root      20   0  2344 1772 1508 R    1  0.2   0:00.62 top
    1 root      20   0  1692 1188 1132 S    0  0.1   0:21.78 init
    2 root      20   0     0    0    0 S    0  0.0   0:00.02 kthreadd
```

Figure 104: NMF App memory usage on the MitySOM-5CSX.

As mentioned before, the target machine for the NMF's Java implementation is OPS-SAT's Experimental Platform, the MitySOM-5CSX. The Benchmark App on the MitySOM-5CSX takes about 45 MB of memory after initialization with none of the JVM parameters configured. Although there are two java processes in Figure 104, only one of then corresponds to the Benchmark App, the second java process is the NanoSat MO Supervisor application.

The memory footprint displayed in Figure 104 does not correspond to the actual memory used by the application. The JVM allocates additional memory from the operating system in order to do its own memory management. Therefore, it is important to analyze the used heap memory, to have a better insight.

The heap size is not relevant here because it can be configured to be any value as mentioned before, however the used heap size needs to be assessed. The heap memory was analyzed for a continuous period of time without any direct consumer connected to it. Right after initialization of the Benchmark App, a garbage collection was forced to occur. It was verified that the used heap size was around 8 MB after the garbage collection. Figure 105 presents the evolution of the memory heap in a 10 minutes time period when there is no consumer connected to the application.

It is important to notice in Figure 105 that when there is no interaction with consumers, the used heap size does not increase significantly in function of time.

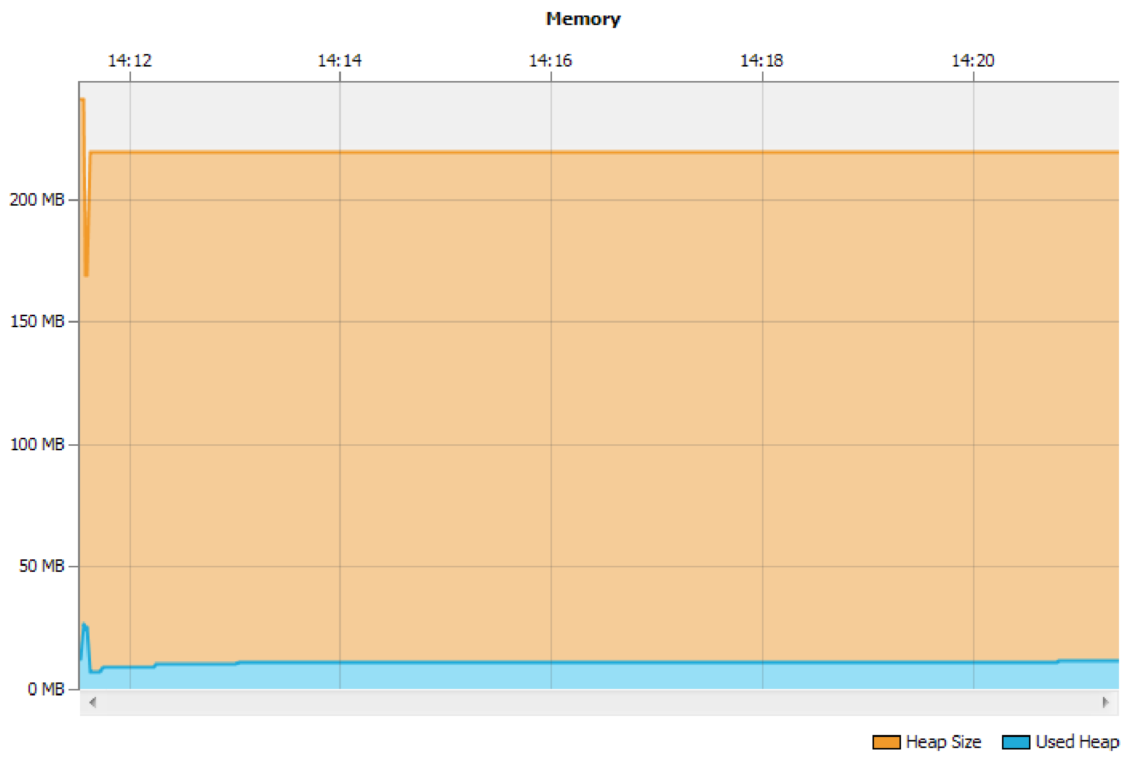


Figure 105: Benchmark App memory evolution in 10 minutes with no consumer interactions.

It is expected from NMF Apps the periodic reporting of parameter values. Figure 106 presents the evolution of the garbage generation in the used heap in a 10 minutes period. The Benchmark App was publishing a parameter value to the consumer every second. In the period between 15:00 and 15:05, there was an increase of the used heap memory of about 9 MB. Most it (if not all) is garbage that would be collected during the next garbage collection.

It is important to notice that the used memory does not increase significantly in function of time even when publishing a parameter value every second.

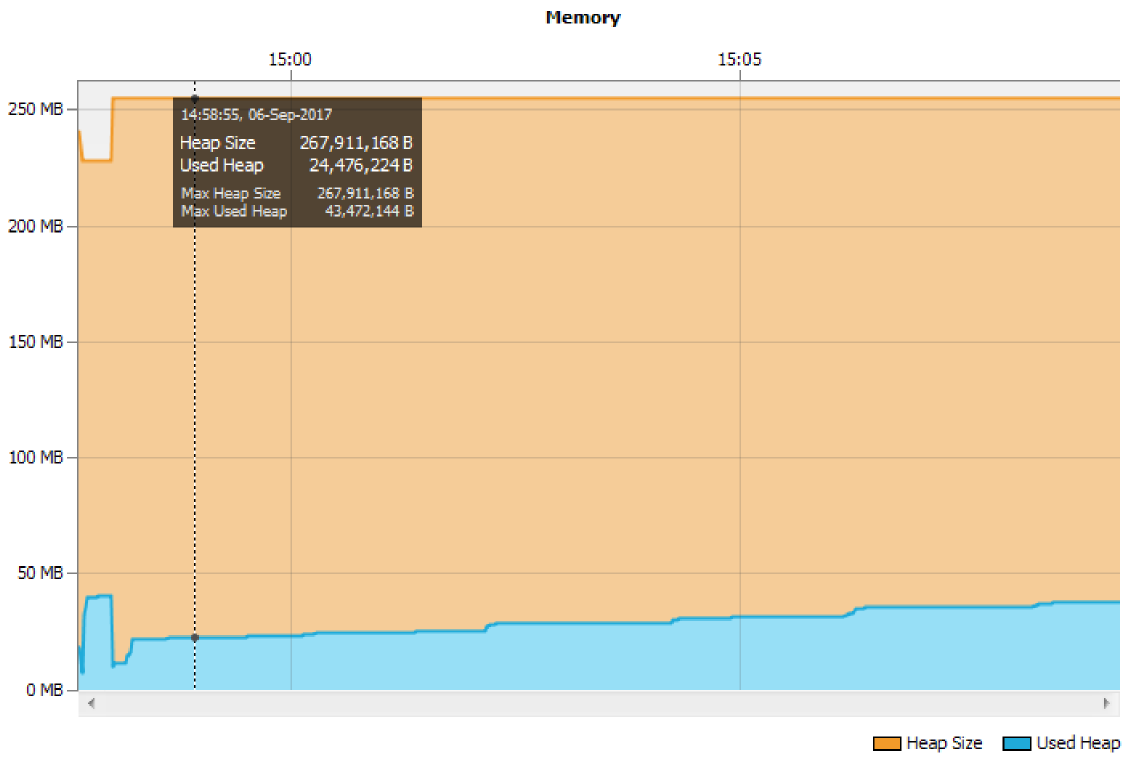


Figure 106: Benchmark App memory evolution in 10 minutes with one periodic parameter every second.

6.3.2. NMF App initialization time

The initialization of an NMF App is a process that takes a certain amount time. This performance parameter is a relevant metric to present because there are different layers and a considerable number of service providers available out of the box in the NanoSat MO Connector component.

Considering that the initialization time is different between tries, the values presented in this section are a calculated mean value of a set of five sequential tries. The value of each individual try is available in Appendix A.

There are external variables that can influence the outcome of each try. For example, it was observed that if there are other applications running at the same time executing computationally expensive tasks then the NMF App initialization time would increase. For example, when the NanoSat MO Supervisor application is initialized, a spacecraft simulation is running simultaneously to be able to provide values for the Platform services.

The two selected machines had different NMF Missions installed and therefore different NMF Libraries were available. The NMF App on the Workstation computer was using the NMF Library for the Software Simulator's NMF Mission while the MitySOM-5CSX was using the NMF Library for the OPS-SAT's NMF Mission. Although the NMF Libraries are different, the NanoSat MO Connector component that is being initialized is the same, which means that the NMF Apps will execute the same instructions. The only exception is for the dynamic loading of the mission-specific transport binding as OPS-SAT's mission requires the dedicated MAL-SPP transport binding. It is relevant to mention that the NanoSat MO Supervisor running on the Workstation is connected to a software simulator that takes some resources from the machine.

Table 17 presents a compilation of the results for the different NMF App's initialization cases and their respective performance values on the selected machines.

#	Conditions	Workstation	MitySOM-5CSX
1	**Started directly from folder**	1.02 seconds	6.13 seconds
2	**Started by NanoSat MO Supervisor**	1.22 seconds	6.39 seconds
3	**Started by NanoSat MO Supervisor (creating database file)**	1.93 seconds	9.26 seconds

Table 17: NMF App initialization time with different conditions and machines.

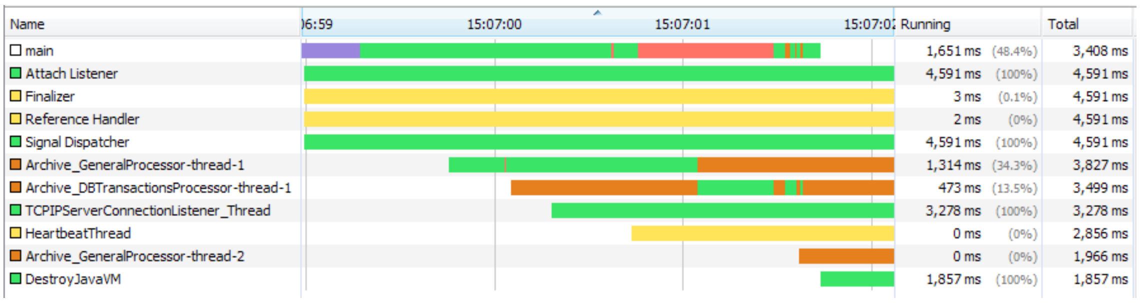
The full description of the conditions and what they represent is further described below:

1. The "Started directly from folder" condition is when the NMF App is started directly from the folder where it is running and the NanoSat MO Supervisor is

not running. Therefore, the NanoSat MO Connector will not connect to the NanoSat MO Supervisor.

2. The “Started by NanoSat MO Supervisor” condition is when the NMF App is started by the NanoSat MO Supervisor. In this case, the NMF App will connect to the NanoSat MO Supervisor to connect to the Platform services and also for registering in the Central Directory service. This is the expected case for the nominal startup of an NMF App.
3. The “Started by NanoSat MO Supervisor (creating database file)” condition is similar to number 2 with the only difference that the COM Archive will have to generate the database file at startup. Considering that it involves the creation of a file in the storage device, the time is expected to be longer than the previous. This is the equivalent of a first run right after installing the NMF App in the system.

The initialization time of the NMF Apps was improved during the research. Originally, the initialization was taking about 30% more of the time of the values presented on Table 17. After examining the execution time internally in each method of the code, it was observed that the initialization of eclipselink in the COM Archive was taking a considerable amount of time. In order to improve it, it was decided to take advantage of multi-threading and to use a dedicated thread just for the task of initializing eclipselink. In Figure 107 one can visualize the initialization of eclipselink task in the “Archive_GeneralProcessor-thread-1” thread while executing at the same time as the “main” thread.



Name	Running		Total
☐ main	1,651 ms	(48.4%)	3,408 ms
🟩 Attach Listener	4,591 ms	(100%)	4,591 ms
🟨 Finalizer	3 ms	(0.1%)	4,591 ms
🟨 Reference Handler	2 ms	(0%)	4,591 ms
🟩 Signal Dispatcher	4,591 ms	(100%)	4,591 ms
🟧 Archive_GeneralProcessor-thread-1	1,314 ms	(34.3%)	3,827 ms
🟧 Archive_DBTransactionsProcessor-thread-1	473 ms	(13.5%)	3,499 ms
🟩 TCPIPServerConnectionListener_Thread	3,278 ms	(100%)	3,278 ms
🟨 HeartbeatThread	0 ms	(0%)	2,856 ms
🟧 Archive_GeneralProcessor-thread-2	0 ms	(0%)	1,966 ms
🟩 DestroyJavaVM	1,857 ms	(100%)	1,857 ms

Figure 107: Threads running in parallel during the initialization of the PushClock NMF App.

To further improve the NMF App initialization time, it is suggested to completely remove eclipselink from the COM Archive implementation. This would also make the NMF Apps more lightweight.

6.3.3. NMF App COM Archive database storage size

The database storage size of the COM Archive is a metric that quantifies the size of a file from the result of executing a deterministic algorithm. Therefore the value only needs to be measured on a single machine as the second machine would produce the exact same value assuming that the same set of inputs and instructions being executed are the same. The workstation machine was selected for the execution of the assessment due to practicality.

A set of questions were defined:

1. What is the database file size expected to be after running an NMF App for the first time? We use Benchmark app as example.
2. What is the size of the Archive if we store a 10’000 parameter values with Double MAL attributes?
3. What is the average size increase per parameter value stored?

The Benchmark NMF App includes an action to trigger the store of 10000 parameter values. Table 18 presents the gathered values from the Benchmark NMF App for the first two questions.

#	Description	Workstation
1	Size of the comArchive.db file after the first run	36’864 bytes (36 KB)
2	Size of the comArchive.db file after the storing 10’000 parameter values	987’136 bytes (964 KB)

Table 18: Collected values of the database file size.

To answer the third question, one can subtracted to the size of the database file to the total size after storing 10’000 parameters and divide by the number of parameters. Both values were collected and therefore it is possible to determine the mean value of a parameter value in the database:

$$\bar{x} = \frac{987136-36864}{10000} = 95 \text{ bytes/parameter value}$$

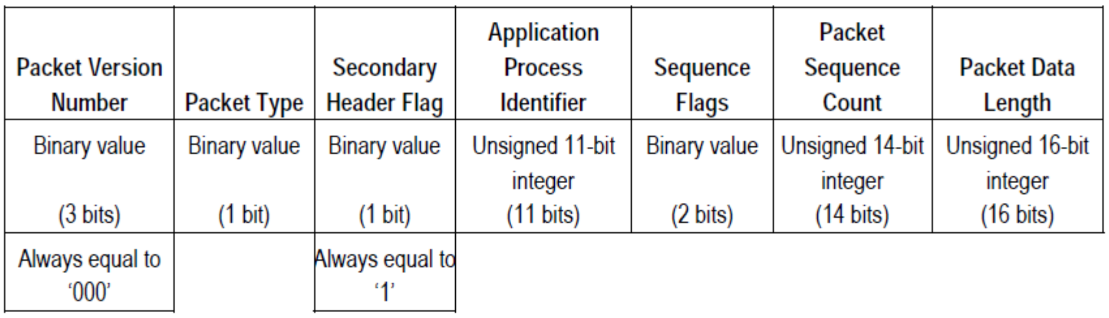
6.3.4. Analysis of SPPs on the CAN bus of NMF Apps

An analysis on the CAN bus of the AGSA Flatsat was performed. It includes an explanation of the MAL-SPP transport binding and a concrete visualization of the data exchanged between the MitySOM-5CSX and the CCSDS Engine for an NMF App. As explained before in chapter 5, the CAN messages have a limit of 8 bytes which is extended to 256 bytes by using the CFP.

The use of the 256 bytes for each CFP messages can be summed by three parts:

1. SPP Primary Header (always 6 bytes)
2. MAL-SPP Secondary Header (minimum of 21 bytes)
3. MAL message Data (remaining)

As mentioned in section 5.3, the exchange of MAL messages on the space link is done using space packets. All the space packets have a 6 bytes primary header as it is presented in Figure 108. This is part of the SPP standard and cannot be optimized. [47]



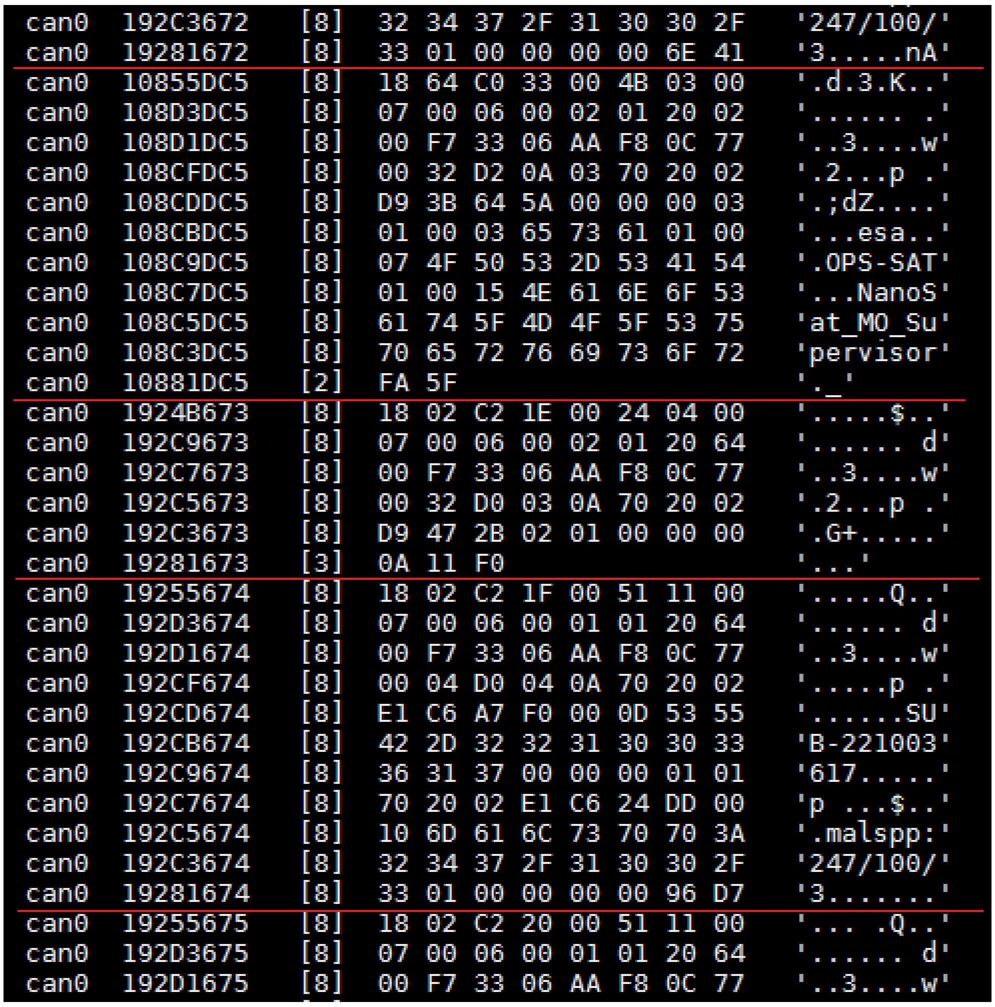
Packet Version Number	Packet Type	Secondary Header Flag	Application Process Identifier	Sequence Flags	Packet Sequence Count	Packet Data Length
Binary value (3 bits)	Binary value (1 bit)	Binary value (1 bit)	Unsigned 11-bit integer (11 bits)	Binary value (2 bits)	Unsigned 14-bit integer (14 bits)	Unsigned 16-bit integer (16 bits)
Always equal to '000'		Always equal to '1'				

Figure 108: MAL-SPP primary header of a space packet. [47]

The MAL-SPP transport binding adds a secondary header to the Packet Data field in order to allow mapping the exchanged message from and to a MAL message. The secondary header has a variable size which can be optimized by configuring the MCPs. The minimum overhead of the secondary header is 21 bytes when all the optional MAL header fields are not passed and without segmentation. [47]

The third part of the message includes the actual data being exchanged. MAL messages can be segmented into multiple segments and exchanged using multiple space packets. This allows bypassing the 256 bytes limitation imposed by CFP.

Complete SPP messages transported in CFP messages can be visualized in Figure 109. A red line separates the CFP messages.



```
can0  192C3672   [8]  32 34 37 2F 31 30 30 2F   '247/100/'
can0  19281672   [8]  33 01 00 00 00 00 6E 41   '3.....nA'
can0  10855DC5   [8]  18 64 C0 33 00 4B 03 00   '.d.3.K..'
can0  108D3DC5   [8]  07 00 06 00 02 01 20 02   '...... .'
can0  108D1DC5   [8]  00 F7 33 06 AA F8 0C 77   '..3....w'
can0  108CFDC5   [8]  00 32 D2 0A 03 70 20 02   '.2...p .'
can0  108CDDC5   [8]  D9 3B 64 5A 00 00 00 03   '.;dZ....'
can0  108CBDC5   [8]  01 00 03 65 73 61 01 00   '...esa..'
can0  108C9DC5   [8]  07 4F 50 53 2D 53 41 54   '.OPS-SAT'
can0  108C7DC5   [8]  01 00 15 4E 61 6E 6F 53   '...NanoS'
can0  108C5DC5   [8]  61 74 5F 4D 4F 5F 53 75   'at_MO_Su'
can0  108C3DC5   [8]  70 65 72 76 69 73 6F 72   'pervisor'
can0  10881DC5   [2]  FA 5F                     '._'
can0  1924B673   [8]  18 02 C2 1E 00 24 04 00   '.....$..'
can0  192C9673   [8]  07 00 06 00 02 01 20 64   '...... d'
can0  192C7673   [8]  00 F7 33 06 AA F8 0C 77   '..3....w'
can0  192C5673   [8]  00 32 D0 03 0A 70 20 02   '.2...p .'
can0  192C3673   [8]  D9 47 2B 02 01 00 00 00   '.G+.....'
can0  19281673   [3]  0A 11 F0                  '...'
can0  19255674   [8]  18 02 C2 1F 00 51 11 00   '.....Q..'
can0  192D3674   [8]  07 00 06 00 01 01 20 64   '...... d'
can0  192D1674   [8]  00 F7 33 06 AA F8 0C 77   '..3....w'
can0  192CF674   [8]  00 04 D0 04 0A 70 20 02   '.....p .'
can0  192CD674   [8]  E1 C6 A7 F0 00 0D 53 55   '......SU'
can0  192CB674   [8]  42 2D 32 32 31 30 30 33   'B-221003'
can0  192C9674   [8]  36 31 37 00 00 00 01 01   '617.....'
can0  192C7674   [8]  70 20 02 E1 C6 24 DD 00   'p ...$..'
can0  192C5674   [8]  10 6D 61 6C 73 70 70 3A   '.malspp:'
can0  192C3674   [8]  32 34 37 2F 31 30 30 2F   '247/100/'
can0  19281674   [8]  33 01 00 00 00 00 96 D7   '3.......'
can0  19255675   [8]  18 02 C2 20 00 51 11 00   '... .Q..'
can0  192D3675   [8]  07 00 06 00 01 01 20 64   '...... d'
can0  192D1675   [8]  00 F7 33 06 AA F8 0C 77   '..3....w'
```

Figure 109: Space packet being exchanged in the CAN bus.

It is also important to mention that the maximum data rate for the CAN bus (1 Mbit/s) is not being fully achieved neither by the NMF Apps nor by the NanoSat MO Supervisor. The achieved data rates are about 40% of the maximum for an NMF App. The reason for this is unknown.

The data rate mentioned above is easily surpassed if an NMF App is executed exactly in the same conditions with the only difference that instead of having SPP over CFP, there is SPP over TCP/IP. Therefore, one can state that the SPP layer and all the above layers are not the limiting factor.

Chapter 7

7. NMF OUTLOOK

In this chapter, some of the future prospects for the NanoSat MO Framework are presented. It includes the current limitations and possible expansions that can be done for its improvement, possible add-ons that are not directly related with the framework, the integration with other CCSDS Standards, feedback for the improvement of MO, potential missions, Pay-per-Orbit, and some ideas for potential “closely-related products”.

The NanoSat MO Framework creates many new opportunities in all three problem domains: NMF Ground development, NMF Mission development, and NMF Apps development. Developers can create NMF Ground applications and NMF Apps at a much faster rate than before. Those applications can be reused by different nanosatellite missions.

Many new ideas were conceptualized during the research process and therefore there is a lot of untapped potential still to be explored.

7.1. NMF current limitations and potential expansions

Some limitations exist and their suggested solutions are presented together with potential expansions for improvement.

At the time of writing, there’s only one single implementation of the NanoSat MO Framework which was used to prototype and demonstrate the concepts. This implementation was presented in chapter 3 and it allows developers to build new applications using a Software Development Kit presented in chapter 4. The NMF is not limited to Java and can be implemented with other programing languages however this was not done in this research.

There are some services that were not implemented and others that were only partially implemented. For example, in the Software Management services, the Memory Management service, the Software Image service, and the Process Management service were not implemented. In the Common services, the Configuration service was not completely implemented as the configurations cannot be retrieved in the XML format, and for the Directory service, it is not possible to return the XML file of the services available by the provider.

The NanoSat MO Supervisor application deploys the Platform services and exposes them to other consumers. This approach should not be promoted in the future and instead, the Platform services should be deployed in a dedicated application separate from the NanoSat MO Supervisor. The reason behind this is that the Platform services' adapters are implemented by the specific mission and therefore there is a higher risk for potential problems. If one of the adapters crash, it is better to happen outside the NanoSat MO Supervisor application. One suggestion is to conceptualize a new NMF Composite: “NanoSat MO Platform”. If this NMF Composite crashes, the NanoSat MO Supervisor would be able to start it again. Another option would be to break it even further and separate the individual Platform services into multiple applications.

The collection of Platform services can also be expanded, for example, a Propulsion service, a Spectrometer service, or a Telescope service could be defined.

Further research is necessary to mature and evolve the NMF Package concept:

- the content of the package contains the whole data to be installed even for small updates. A more advanced technique could be conceptualized where only the delta between the update and current version installed would be transferred. This would reduce the amount of data to transfer on the space link for an update. Delta compression libraries could be reused for this.
- the NMF Package concept is not “NMF App” aware. According to its definition, it is a container that carries files without awareness of what those files actually are. One could improve it by adding more metadata to the receipt file of the package. The receipt file was designed to support evolution.
- the NMF Package could have the parameter definitions, aggregations definitions, action definitions, and alert definitions defined on a file. This would avoid having to dynamically retrieve them from the COM Archive of the NMF App after the first run. In addition, it could state which Platform services are needed in order to execute the NMF App.

The NanoSat MO Framework does not support file transfer natively. If an NMF App is collecting data to a file that needs to be transferred to ground, then a third-party application needs to be available for executing the transaction of the files. Section 7.3.2 proposes a more advanced mechanism for an integrated solution.

7.2. NMF Add-ons

The NMF Add-ons are a set of independent ideas that are related with the NanoSat MO Framework but do not require any direct (or significant) changes on the framework itself. The add-ons can operate with the NMF's Java implementation with little to no customization.

7.2.1. Connector in C

The CCSDS MO Framework is technology-independent and therefore the NanoSat MO Connector doesn't necessarily need to be developed in Java. If other languages are to be supported, then the logic of the NanoSat MO Connector needs to be translated to the new target programming language.

A MAL implementation in C by CNES is available online and this could potentially be reused to build part of the MO stack that needs to be in place for the NanoSat MO Connector in a different programming language. [62]

A lightweight version of the NanoSat MO Connector could be developed that does not have any providers but that could consume the Platform services from the NanoSat MO Supervisor. This could be useful, for example, to simple applications that only want to get some data from the Platform but that do not require monitoring and control from ground.

7.2.2. Apps Launcher: Applications Sandboxing

The Apps Launcher service implementation developed during the research does not provide applications' sandboxing. The service is able to start and kill processes for the respective applications that are launched however it does not provide any other additional mechanisms of security.

Libvirt Sandbox is an application sandbox toolkit which could be integrated with the Apps Launcher service. This application supports different virtualization technologies to build the sandbox such as LXC, QEMU or KVM. [109]

Another possible solution would be to start and stop Docker containers instead of processes from the Apps Launcher service. Docker is a software container platform that intends to eliminate "works on my machine" problems when collaborating on code with co-workers and also to run and manage apps side-by-side in isolated containers. [110]

For reference, other types of sandboxes exist such as mBox and Firejail. [111] [112]

7.2.3. Envisaged NMF Ground applications

The NMF allows the development of many different NMF Ground applications. For example, EUD4MO presented in chapter 5 used to be one of the envisaged applications until it was decided by ESA to invest in its development. Additional ideas are presented in this subsection.

MATIS is a Mission Automation System developed in ESOC that aims at supporting automation of operations for elements of a standard mission round segment within the ESOC mission operations activities. It is possible to merge MATIS with the NMF by creating a SMF driver that is capable of exchanging MO messages with an MO Provider. Merging both technologies would allow automatic mission operations of space assets in the future. The suggested name for this project is: AutoMATISMO. [53]

Yamcs (short for Yet another mission control system) is a software package that positions itself as a lightweight alternative to traditional heavyweight Mission Control Systems. Similarly to EUD4MO, Yamcs can be integrated with the Ground MO Adapter and allow an operator to interact with MO providers. [113]

Open MCT (Mission Control Technologies) is a next-generation mission control framework for visualization of data on desktop and mobile devices, developed at NASA's Ames Research Center in Silicon Valley, in collaboration with the Jet Propulsion Laboratory. It is web-based, open-source, and similarly to EUD4MO, its integration with the Ground MO Adapter might be possible. [114]

7.2.4. Envisaged NMF Apps

As mentioned in chapter 1, the trend of space processors seems to indicate an increase in its computational capabilities. If the trend continues, there will be enough resources to run more advanced software applications with more functionalities and autonomy on-board.

The NMF allows the development of many different NMF Apps. For example, many OPS-SAT’s experimenters are expected to be developing their experiment in form of an NMF App in order to quickly achieve their objectives with minimum effort. This allows reuse of existing tools for mission operations such as monitoring and control with EUD4MO via a web browser.

An NMF App to monitor other applications and collect analytics can be very useful for debugging possible problems that might exist during their execution. This might include parameters such as processor usage, spacecraft resources usage, memory usage, or storage usage.

The Fault Detection, Isolation, and Recovery (FDIR) can be done as an NMF App which keeps track of the nanosatellite's health status and can act in case of a failure.

Different earth observation NMF App could be developed to do specific activities such as tracking crops or detecting fires from space. Earth observation can have other applications, such as in agriculture, forecasting weather, forestry, maritime, environmental, and others.

The same NMF App can be used by other nanosatellites as portability is one of the characteristics of the NMF Apps. In a scenario with a constellation of nanosatellites, the same NMF App could be deployed in all the nodes in order to achieve a common goal. Higher coordination can be achieved with more autonomy on-board when compared with classical satellites.

There are many possible NMF Apps that can be created and deployed in space with a wide range of different applications and purposes. Similarly, no one could imagine all the different "apps" that were or will be created for smartphones.

7.2.5. NMF integration with an AndroidOS

Android is an Operating System developed by Google, based on the Linux kernel and designed primarily for touchscreen mobile devices such as smartphones and tablets. [115]

There are some previous attempts to bring Android to nanosatellites. Some missions such as STRaND-1 and NASA AMES's PhoneSat have already flown Android in space. [41] [116]

There is also Android Beyond the Stratosphere (ABS) project which is aimed at providing a public access to space through the creation of a satellite constellation conformed by Android-based nanosatellites which allows the execution of Android applications in space. The OSS basic building block, the ABS Unit, encompasses both the essential hardware and software components required to build a highly modular payload-oriented nanosatellite. It might be possible to integrate the NMF with the ABS software in order to provide the NMF's advanced framework features. [117] [118]

The ABS project could be tested in OPS-SAT as an Operating System experiment. It can also be used on nanosatellites with similar advanced processing capabilities. [14]

7.2.6. Java Acceleration

As mentioned in chapter 2, although the first reference implementation is developed in Java, the NMF is not necessarily tied to Java and the architecture and concepts behind can be reimplemented in a different programming language.

For OPS-SAT, the NMF's Java implementation has reasonable performance metrics on its target platform as presented in section 6.3. However, if a different mission uses the same implementation instead of redeveloping the NMF from scratch, then it might be worth to know that there are new Java acceleration techniques using an FPGA in case the performance becomes a bottleneck and needs to be optimized.

The JUNIPER project is developing a framework for the construction of large-scale distributed systems in which execution time bounds can be guaranteed. It includes automatic implementation of input Java code on FPGAs, both for speed and predictability. [119] [120]

7.2.7. NMF Packages Repository

The NMF Package concept is defined in section 2.7.3 as a digital content distribution package intended to be used to install, uninstall and update applications on a spacecraft system that uses the NanoSat MO Framework.

An open repository of NMF Packages online would increase the reuse of software among nanosatellite developers and users. This would contribute to the digital distribution of packages on a wider worldwide scale.

Potentially, a plugin could be created that would automatically generated the NMF Package after the compilation process of an NMF App. This would facilitate the delivery of NMF Apps as they could be bundled automatically in an automated continuous delivery pipeline.

7.3. NMF with other CCSDS Standards

During the design of the NanoSat MO Framework, MO was the primary focus among the standards in the CCSDS however other standards were not completely ignored and were also taken into account. This section includes a short analysis of the NMF in relation to other CCSDS standards.

At the time of writing, the CCSDS covers 6 different areas: Space Internetworking Services, Mission Ops. And Information Management Services, Spacecraft Onboard Interface Services, System Engineering, Cross Support Services, and Space Link Services. Examples of popular standards developed by the CCSDS include Electronic Data Sheets (EDS), Space Link Extension (SLE), CCSDS File Delivery Protocol (CFDP), Bundle Protocol (BP), and Space Packet Protocol (SPP). [34] [121] [122] [123] [124]

Figure 110 presents a future system architecture based upon MO, SOIS, and CFDP. [125]

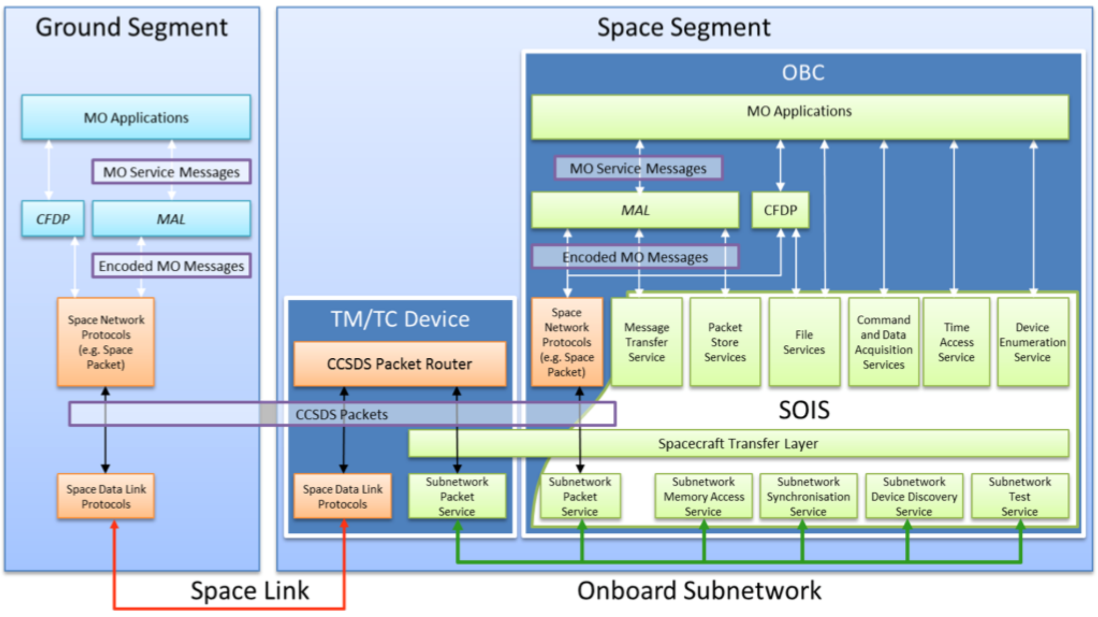

Ground Segment
Space Segment
OBC
MO Applications
MO Service Messages
CFDP
MAL
Encoded MO Messages
Space Network Protocols (e.g. Space Packet)
TM/TC Device
CCSDS Packet Router
CCSDS Packets
Message Transfer Service
Packet Store Services
File Services
Command and Data Acquisition Services
Time Access Service
Device Enumeration Service
SOIS
Spacecraft Transfer Layer
Space Data Link Protocols
Subnetwork Packet Service
Subnetwork Memory Access Service
Subnetwork Synchronisation Service
Subnetwork Device Discovery Service
Subnetwork Test Service
Space Link
Onboard Subnetwork

Figure 110: Architecture based on CCSDS standards.

The integration of MO and SPP was done by standardizing the MAL-SPP transport binding which allows the exchange of MAL messages using space packets. [47]

The integration of MO and CFDP was previously investigated at ESA and a proof of concept was implemented and it is intended to be used for OPS-SAT. A draft File Management service was defined in MO to handle the management of files and folders while CFDP is used for the actual transfer of the files.

At the time of writing, efforts within the CCSDS to integrate MO and EDS are on-going. [121]

The NanoSat MO Framework is able to operate together with all the standards mentioned above because it is built on top of MO. The next two subsections suggest dedicated approaches for the integration of the NMF with EDS, and also the integration of CFDP with advanced capabilities for other NMF Apps. [121]

7.3.1. EDS device integration with the Platform services

There are three possible approaches for the integration of EDS and MO: [121]

1. Direct mapping from an EDS into a generated MO service
2. Use the Action service and Parameter service to interact with an EDS device
3. Use specialized MO services to interact with the EDS devices

The NMF includes a set of Platform services which are specialized for a device therefore the third option would be the best fit. The Platform services allow the exchange of semantically meaningful data instead between the consumer and provider rather than just a command to trigger an action or a parameter containing a value. The Platform services are explained in chapter 2.

In terms of implementation, the NMF's Java implementation supports different vendors for a certain device by taking advantage of the Adapter pattern.

7.3.2. Dropblocks

As mentioned before, there is a proof of concept implementation of CFDP with MO that will be demonstrated in flight for the first time with OPS-SAT.

A more advanced application with file transfer capabilities can be developed with better integration within the new NMF concepts. Dropblocks is a conceptual NMF App for the transfer of files between the NanoSat and Ground segments. The application would expose a set of MO services internally on the nanosatellite for other NMF Apps to use.

This would allow sending/receiving files to/from ground in an abstract way to the actual transfer mechanism (which could be CFDP, FTP, or other). The development of such application can be done using the NanoSat MO Framework Software Development Kit and then later on, tested in the AGSA Lab Flatsat.

A very simple 3-folders system is envisaged:

- inbox
- outbox
- sync

The expected technology to be used for exchanging files between the inbox and outbox folder, is CFDP however the application should not be tied to a single file delivery mechanism and therefore it should be developed in an independent way from the underlying technology. This should allow the use of other file transfer mechanisms such as FTP, SFTP, and/or SCP.

The sync folder is an envisioned way of exchanging files based on file synchronization. In terms of needed technology, this should be possible using rsync over DTN.

7.4. MO Improvement Feedback

There are many improvements in MO that have been provided directly to the CCSDS SM&C working group during the research process. This includes the improvement of service definitions, Review Item Dispositions (RIDs) for the books during their respective revision phases, the prototyping of many standardized services, including the COM, M&C, and Common sets of services. This section compiles a set of suggestions for improvement of MO.

During the research, it was observed that some people (students, trainees, and even professionals) working with MO, even after a few months, still don't know exactly MO is and it is not uncommon to hear statements such as "MO is a protocol", or "MO is in Java". The problem lies in the fact that MO is multi-domain, transport-agnostic, and programming language independent. This is not necessarily bad however many people struggle to understand these abstractions.

The suggestion to improve this situation is to rebrand MO as a "technology" and then build smaller derivatives on top of it that target a specific niche. For example, for mission operations of nanosatellites, the NanoSat MO Framework is the framework that allows quick development and assembly of end-to-end systems for nanosatellites. Other MO niches can be envisioned, for example, for robotic operations or human space operations. These niches are presented in further detail in section 7.5 with concrete ideas for future projects.

The following definition is suggested: "MO is a technology for mission operations of space assets". Then, it is important to move from "technology" into concrete software frameworks that can be used for mission operations of particular niches.

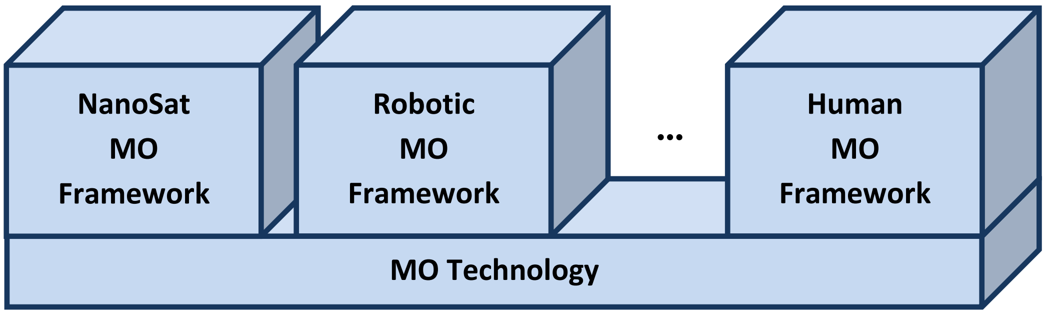


Figure 111: Example of possible MO niches.

By providing pre-developed interoperable components within the derivatives, new developers can quickly create software solutions that are able to deliver complete end-to-end systems for their particular use case. This would promote reusability of software, and considerably accelerate the development time of a new solution that is guaranteed to work with previous products developed based on that derivative.

On the COM services, there are many improvements that can be made. First, the definition of a COM Object needs to be clearly stated in the standard instead of implicitly defined as it is today. This should include all its fields regardless of them being mandatory or optional.

The Archive service stores COM Objects with a timestamp field defined as FineTime type. By changing the type from FineTime to Time, it is possible to save 4 bytes per COM object in the Archive. This makes a big difference when a big set of COM Objects are stored. If higher resolution is needed (very seldom case) then a dedicated COM Object can be defined for that particular use case that includes an object body with a FineTime field. However in most of the cases, COM Objects don't need to be timestamped with FineTime resolution and Time type with a milliseconds resolution is enough.

A new service is proposed to be created in the COM services set, the COM History service. This service would allow a consumer to retrieve the history of a COM Object and know when it has been edited in the past. For this, a set of new COM Objects need to be defined in order to store the differences done on the mutated COM Object. This would allow having a consistent track of the COM Object's evolution.

The current "Entity Model" concept implemented in the M&C services to keep history of definitions relies on the replication of the whole COM Object and then to apply the new change. A new definition COM Object is created every single time that a definition is edited, and also when its parameters/aggregations/alerts reporting is enabled or disabled. This inefficient "Entity Model" concept can be replaced by the new History service.

The encoding of the MAL Identifier, MAL URI, and MAL String is always done by having a string length field followed by the characters encoded as UTF-8. This occurs for both the Binary encoding, and for the Split Binary encoding. During the low-level CAN bus sniffing of the AGSA Lab, it was noticed that many of the messages are long because of the way strings are encoded. Better solutions can potentially be explored based on dictionaries (fixed or dynamic) for strings. [47] [48]

The resolution and epoch for FineTime and Time MAL data structures should be fixed for the MAL Java API standard book. Conversions between different programming languages and different encodings should be carefully assessed because they can cause real big damage in case of conversion lost. When dealing with space assets, handling time should have a high priority in order to avoid possible numerical errors. NASA's Deep Impact mission is an example of a lost mission because of time-related problems. [126] [127]

The Platform services and the Software Management services defined in chapter 2 are good candidates for future standardization.

7.5. “Closely-related products” ideas

As mentioned on the previous section, it is possible to explore other niches where MO can provide solutions such as robotic mission operations or human space operations.

7.5.1. Robotic MO Framework

A software framework for robotics operation could be created. The Robotic MO Framework would be a software framework for mission operations of robotic assets.

The hard lessons on how to make a software framework based on MO have already been learnt during the research process of the NMF and therefore the task of developing a similar product targeting a different niche is easier. Many of the implemented blocks of the NMF’s Java implementation could potentially be reused such as COM, Common and M&C services.

The Telerobotics working group of the CCSDS has published an informational report which includes overview and descriptive material supporting analysis, requirements, and example scenarios, that will help bound the scope of a proposed CCSDS Recommended Standard for telerobotic operations. [128]

The Robotic MO Framework could take advantage of many of the ideas presented in the CCSDS informational report and could also take inspiration in some of the ideas introduced by the NMF such as the concept of portable apps that can be started and stopped on demand which are focused on specific functionality. Instead of having a Ground MO Proxy, one could have a Space MO Gateway.

7.5.2. Human MO Framework

A software framework for human space operations could be also created. The Human MO Framework would be a software framework for mission operations of humans in space.

The framework should support not only remote monitoring and control of astronauts but also local interactions between them. It is envisioned a system where the astronaut can be aware of its environment, exchange information with other entities (astronauts, rovers, ground control, etc) and that it keeps track of the important information that the astronaut needs to know in order to execute the on-going task.

This would be present in spacesuits, ground control, or local area control (for example, another astronaut in a space station). A Human Machine Interface inside the spacesuit could include voice recognition, gesture recognition, and facial motion capture for interacting with the astronaut. New voice recognition algorithms would allow sending text messages to the team, and interacting with rovers or auxiliary support systems.

It should be possible to integrate the Human MO Framework with the solutions developed using the Robotic MO Framework and the NanoSat MO Framework.

The use of Virtual Reality or Augmented Reality environments would allow the testing of the software and also aid the training of astronauts before their mission. Another option would be “Aouda.X”, an advanced spacesuit simulator for future human Mars missions of the Austrian Space Forum that could potentially run the Human MO Framework. [129]

7.6. Other ideas

A new type of service can be envisioned: “Pay-Per-Orbit”. This allows a customer to pay based on the duration of the software execution on the nanosatellite. With small customizations, the NMF could start and stop apps autonomously based on a location or schedule and that would allow the creation of a business model where users can “rent” the satellite and deploy their software only over certain areas of the world or only at a certain time. [56]

If two applications are independent and don’t interfere with each other’s execution, then it is possible to deploy them at the same time. This is similar to the “ride share” concept on earth. Section 7.2.2 presents possible ways to sandbox applications in order to isolate them.

The concepts above could also be used on a constellation of nanosatellites. This would allow scaling the “Pay-Per-Orbit” concept into a multi-node system. New ground automation techniques would then have to be employed in order to manage the fleet. A “multi-purpose” constellation would be a game-changer in the way space services are provided. Instead of creating a complete new mission with risky launches, a company could just deploy its mission software on the constellation provider.

Conclusions

The content of this research challenges the current view on software for space systems. A software framework for nanosatellites based on the latest CCSDS standards was designed, prototyped, and tested successfully. This software framework was branded as NanoSat MO Framework.

Its design follows the latest CCSDS standards on MO Services and it is inspired by current smartphone technologies. It is independent of any particular mission and high priority was given to support reuse. Modular components were defined as NMF Composites and they allow quick development of software capable of operating in end-to-end scenarios. New concepts were defined such as NMF Package, NMF App, and NMF Mission.

A prototype was implemented in Java that provides not only a proof of concept demonstration but also the foundation to develop other software on top of it. A Software Development Kit (SDK) was put together in order to facilitate the development of software using the prototype implementation. This allows the development of NMF Apps that can run on a spacecraft and the development of NMF Ground applications. The prototype target device was selected to be the same as OPS-SAT's experimental platform, an advanced on-board computer when compared with today's traditional on-board computers.

A Flatsat was built in ESOC's AGSA laboratory that contains the target device. This test bench was used to validate the concepts, to try the NMF Mission for OPS-SAT in an environment similar to the final setup, and to demonstrate that the NanoSat MO Framework can be used operationally in OPS-SAT. Performance metrics were collected for future benchmark activities.

The NanoSat MO Framework unlocks many new possibilities. It can be expanded with new features and many possible add-ons.

Software Management services were defined for the management of software applications and packages. These services were successfully integrated with the NMF Apps and NMF Packages concepts. The standardized MO Monitor and Control services provide an interface for the monitoring and control of NMF Apps by a consumer. Platform services were defined for interacting with the platform peripherals of the spacecraft. These services provide a simple high-level interface that hides the low-level details of each specific nanosatellite platform. The abstraction between the application logic and system interfaces allow the creation of portable applications that can be reused by different nanosatellites.

A new world of apps for nanosatellites was created!

Scientific Papers

- C. Coelho, D. Evans, O. Koudelka, "CCSDS Mission Operations Services on OPS-SAT", IAA Symposium on Small Satellites for Earth Observation, 2015
- C. Coelho, O. Koudelka, M. Sarkarati and M. Merri, "NanoSat MO Framework: Achieving On-board Software Portability", SpaceOps 2016 Conference, 2016.
- C. Coelho, S. Cooper, O. Koudelka, M. Sarkarati, M. Merri, "CCSDS Mission Operations Directory Service: Paving the road for a new world of services' discoverability", 4S Symposium, 2016.
- C. Coelho, M. Merri, O. Koudelka and M. Sarkarati, "OPS-SAT Experiments' Software Management with the NanoSat MO Framework", AIAA SPACE 2016, 2016.
- C. Coelho, O. Koudelka and M. Merri, "NanoSat MO framework: When OBSW turns into apps", 2017 IEEE Aerospace Conference, 2017.
- C. Coelho, S. Cooper, M. Merri, M. Sarkarati, and O. Koudelka, "NanoSat MO Framework: Drill down your nanosatellite's platform using CCSDS Mission Operations services", 68th International Astronautical Congress (IAC), 2017.
- D. Evans, A. Lange, J. Feiteirinha, J. Noertemann, C. Coelho, " OPS-SAT: Preparing for the Operations of ESA's First NanoSat", SpaceOps 2016 Conference, 2016.

Activities

Attended Conferences:

- IAA Symposium on Small Satellites for Earth Observation, 2015
- 14th International Conference on Space Operations (SpaceOps 2016)
- Small Satellites, Systems and Services Symposium (4S), 2016
- AIAA Space 2016
- IEEE Aerospace Conference, 2017

Others activities:

- CCSDS active support and participation in the Technical Meetings
- OPS-SAT Software Workshop in Graz
- ESTEC – CubeSat Industry day 2017 – Presentation delivered
- CEiiA Workshop – Presentation delivered
- Participation in the 2nd International PhD Summer School in Palanga, Lithuania organized by the Kaunas University of Technology
- Mentor in ESA Summer of Code in Space 2017 (SOCIS 2017) of 2 students

Dedicated CCSDS activities:

- Multiple Review Item Dispositions for improvements
- Implementation of all the standardized services
- Prototyping of future MO services
- Research for the improvement of the Directory service
- ESA's Review Leader for CCSDS 524.2-B-1: "Mission Operations –Message Abstraction Layer to TCP/IP Transport and Split Binary Encoding"
- ESA's Review Leader for CCSDS 523.2-M-1: "Mission Operations Message Abstraction Layer – C++ API" book

Active participation on the following CCSDS Technical meetings:

- CCSDS Fall 2014 (London, UK)
- CCSDS Spring 2015 (Pasadena, USA)
- CCSDS Fall 2015 (Darmstadt, Germany)
- CCSDS Spring 2016 (Cleveland, USA)
- CCSDS Fall 2016 (Rome, Italy)

Source Code

The source code developed as part of this research is available on the link:

https://github.com/NanoSat-MO-Framework

The Software Development Kit of the NanoSat MO Framework is also available on the same link.

Appendix A

The collected Benchmark App initialization times for the performance metrics:

1	
Workstation	
Trial	Time (s)
1	0.999
2	1.031
3	1.015
4	1.033
5	1.014
Average:	1.02

1	
MitySOM-5CSX	
Trial	Time (s)
1	6.143
2	6.147
3	6.13
4	6.102
5	6.132
Average:	6.13

2	
Workstation	
Trial	Time (s)
1	1.234
2	1.251
3	1.267
4	1.172
5	1.17
Average:	1.22

2	
MitySOM-5CSX	
Trial	Time (s)
1	6.541
2	6.546
3	6.279
4	6.323
5	6.271
Average:	6.39

3	
Workstation	
Trial	Time (s)
1	1.951
2	1.811
3	1.873
4	2.017
5	1.982
Average:	1.93

3	
MitySOM-5CSX	
Trial	Time (s)
1	9.398
2	9.28
3	9.189
4	9.14
5	9.283
Average:	9.26

References

[1] C. Larman, Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and the Unified Process, Second Edition. Prentice Hall, 2001.

[2] "OPS-SAT", European Space Agency, 2017. [Online]. Available: http://www.esa.int/Our_Activities/Operations/OPS-SAT. [Accessed: 02- Sep- 2017].

[3] "CCSDS.org - The Consultative Committee for Space Data Systems (CCSDS)", Ccsds.org, 2017. [Online]. Available: http://www.ccsds.org. [Accessed: 02- Sep- 2017].

[4] "Networking/Partnering Initiative", European Space Agency, 2017. [Online]. Available: http://www.esa.int/Our_Activities/Space_Engineering_Technology/Networking_Partnering_Initiative. [Accessed: 02- Sep- 2017].

[5] "Where missions come alive", European Space Agency, 2017. [Online]. Available: http://www.esa.int/About_Us/ESOC/Where_missions_come_alive. [Accessed: 02- Sep- 2017].

[6] "2014 Nano/Microsatellite Market Assessment", Atlanta, Georgia: SpaceWorks Enterprises, Inc. (SEI), 2014.

[7] E. Kulu, "Nanosatellite & CubeSat Database", Nanosatellite & CubeSat Database, 2017. [Online]. Available: http://nanosats.eu. [Accessed: 02- Sep- 2017].

[8] Planet, 2017. [Online]. Available: https://www.planet.com/. [Accessed: 02- Sep- 2017].

[9] G. Moore, Cramming more components onto integrated circuits. New York: McGraw-Hill, 1965.

[10] J. Wu, Y. Shen, K. Reinhardt and H. Szu, "A NANO enhancement to Moore's law", Independent Component Analyses, Compressive Sampling, Wavelets, Neural Net, Biosystems, and Nanoengineering X, 2012.

[11] J. Cressler and H. Mantooth, Extreme environment electronics. Boca Raton: CRC Press, 2013.

[12] A. Keys, J. Adams, R. Darty, M. Patrick, M. Johnson, & J. Cressier "Radiation Hardened Electronics for Space Environments (RHESE) Project Overview ", International Planetary Probes Workshop (IPPW), Atlanta, GA, June 2008.

[13] "European Commission : CORDIS : Projects and Results : Final Report Summary - APEX (Advanced Processor Core for Space Exploration)", Cordis.europa.eu, 2017. [Online]. Available: http://cordis.europa.eu/result/rcn/194613_en.html. [Accessed: 02- Sep- 2017].

[14] X. Iturbe, D. Keymeulen, E. Ozer, P. Yiu, D. Berisford, K. Hand and R. Carlson, "Designing a SoC to control the next-generation space exploration flight science instruments", 2015 28th IEEE International System-on-Chip Conference (SOCC), 2015.

[15] NanoMind Z7000 - Datasheet On-board CPU and FPGA for space applications. 2017. [Online]. Available: https://gomspace.com/UserFiles/Subsystems/datasheet/gs-ds-nanomind-z7000-13.pdf. [Accessed: 02- Sep- 2017].

[16] MitySOM-5CSx System on Module - Datasheet. 2017. [Online]. Available: http://www.criticallink.com/wp-content/uploads/MitySOM-5CSx-Datasheet.pdf. [Accessed: 02- Sep- 2017].

[17] O. Koudelka, M. Wittig, D. Evans, ESA's OPS-SAT Nanosatellite Mission - A Laboratory in the Sky, 10th IAA Symposium on Small Satellites and Earth Observation, 2015

[18] "OPSSAT Montage", Pressearchiv.tugraz.at, 2017. [Online]. Available: http://pressearchiv.tugraz.at/webgalleryBDR/data/OPSSAT_20150305/pages/OPSSAT%20Montage.htm. [Accessed: 02- Sep- 2017].

[19] D. Evans and M. Merri, "OPS-SAT: A ESA nanosatellite for accelerating innovation in satellite control", SpaceOps 2014 Conference, 2014.

[20] "THE OPS-SAT NANOSATELLITE MISSION", Prague, Czech Republic, 2015.

[21] D. Evans, A. Lange, J. Feiteirinha, J. Nörtemann, C. Coelho, "OPS-SAT: Preparing for the Operations of ESA’s First NanoSat", SpaceOps 2016 Conference, 2016.

[22] C. Coelho, "Comparison of the CCSDS Mission Operations Services with the Packet Utilization Standard Services", Master Thesis, Luleå University of Technology, 2014.

[23] M. Merri and M. Sarkarati, "Retire Legacy Technology with the CCSDS MO Services", SpaceOps 2016 Conference, 2016.

[24] ECSS-E-70-41A, "Ground systems and operations — Telemetry and telecommand packet utilization". Space engineering. ESA-ESTEC, 2003.

[25] M. Merri, "Cheaper, Faster, and Better Missions with the CCSDS SM&C Mission Operations Framework", AIAA SPACE 2009 Conference & Exposition, 2009.

[26] M. Sarkarati, M. Merri, M. Spada and S. Cooper, "Intrinsic Interoperability of Services: A Dream or a Key Objective for Mission Operation Systems", Space Operations: Experience, Mission Systems, and Advanced Concepts, pp. 353-366, 2013.

[27] Mission Operations Services Concept. Issue 3. Report Concerning Space Data System Standards (Green Book), CCSDS 520.0-G-3. Washington, D.C.: CCSDS, December 2010.

[28] Sam Cooper, "CCSDS Mission Operations Services in Space," Proceedings of SpaceOps 2012, The 12th International Conference on Space Operations, 2012

[29] P. Kaur and S. Sharma, "Google Android a mobile platform: A review", 2014 Recent Advances in Engineering and Computational Sciences (RAECS), 2014.

[30] J. Thomson, A. Guerreiro, P. Trezentos and J. Johnson, "Package Upgrade Robustness: An Analysis for GNU/Linux® Package Management Systems", IFIP Advances in Information and Communication Technology, pp. 299-306, 2011.

[31] M. Robichaud, E. Sandjaya, C. Wagner, "Contribution of the On Board Software Management Tool to the Galileo Operations", SpaceOps 2012 Conference, 2012.

[32] N. 2017, "App stores: number of apps in leading app stores 2017 | Statista", Statista, 2017. [Online]. Available: https://www.statista.com/statistics/276623/number-of-apps-available-in-leading-app-stores/. [Accessed: 02- Sep- 2017].

[33] "Ground segment", En.wikipedia.org, 2017. [Online]. Available: https://en.wikipedia.org/wiki/Ground_segment. [Accessed: 02- Sep- 2017].

[34] Space Link Extension Services. Informational Report for Space Data System Standards, CCSDS 910.0-G-2. Green Book. Washington, D.C.: CCSDS, March 2006

[35] "Software portability", En.wikipedia.org, 2017. [Online]. Available: https://en.wikipedia.org/wiki/Software_portability. [Accessed: 02- Sep- 2017].

[36] "Android Open Source Project", Android Open Source Project, 2017. [Online]. Available: https://source.android.com/. [Accessed: 02- Sep- 2017].

[37] A. Koltashev, "A Practical Approach to Software Portability Based on Strong Typing and Architectural Stratification", Lecture Notes in Computer Science, pp. 98-101, 2003.

[38] D. McComas, S. Strege, J. Wilmot, “core Flight System (cFS): A Low Cost Solution for SmallSats”, Goddard Space Flight Center, 2015

[39] "SAVOIR", Savoir.estec.esa.int, 2017. [Online]. Available: http://savoir.estec.esa.int/. [Accessed: 02- Sep- 2017].

[40] Terraillon, J.-L & Jung, A & Arberet, P & Monenegro, S & Rossignol, A & Gérald, Garcia & Li, J & I. Rodriquez, A & Mazzini, S & Hougaard, P & Fowell, S & Ferraguto, Massimo & Panunzio, Marco & working Group, Savoir-Faire. (2010). “Space On-Board Software Reference Architecture”. 682. 57-.

[41] C. Bridges, B. Yeomans, C. Iacopino, T. Frame, A. Schofield, S. Kenyon and M. Sweeting, "Smartphone qualification & linux-based tools for CubeSat computing payloads", 2013 IEEE Aerospace Conference, 2013.

[42] Mission Operations Common Object Model. Issue 1. Recommendation for Space Data System Standards (Blue Book), CCSDS 521.1-B-1. Washington, D.C.: CCSDS, February 2014.

[43] Mission Operations Monitor and Control services - Draft Revision 4. Issue 1. Recommendation for Space Data System Standards (Red Book), CCSDS 522.1-R-4. Washington, D.C.: CCSDS, April 2017.

[44] Mission Operations Common services - Draft Revision 5. Issue 2. Recommendation for Space Data System Standards (Red Book), CCSDS 522.0-R-2. Washington, D.C.: CCSDS, February 2017.

[45] "esa/CCSDS_MO_GraphicalEditor", GitHub. [Online]. Available: https://github.com/esa/CCSDS_MO_GraphicalEditor. [Accessed: 02- Sep- 2017].

[46] C. Coelho, S. Cooper, O. Koudelka, M. Sarkarati, M. Merri, "CCSDS Mission Operations Directory Service: Paving the road for a new world of services' discoverability", 4S Symposium, 2016

[47] Mission Operations – MAL Space Packet Transport Binding and Binary Encoding. Issue 1. Recommendation for Space Data System Standards (Blue Book), CCSDS 524.1-B-1. Washington, D.C.: CCSDS, August 2015.

[48] Mission Operations – MAL TCP/IP Transport Binding and Binary Encoding. Issue 1. Draft Recommendation for Space Data System Standards (Red Book), CCSDS 524.2-R-1. Washington, D.C.: CCSDS, August 2016.

[49] "esa/CCSDS_MO_TRANS", GitHub, 2017. [Online]. Available: https://github.com/esa/CCSDS_MO_TRANS/tree/master/CCSDS_MAL_TRANSPORT_RMI. [Accessed: 02- Sep- 2017].

[50] "LocationManager | Android Developers", Developer.android.com, 2017. [Online]. Available: https://developer.android.com/reference/android/location/LocationManager.html. [Accessed: 02- Sep- 2017].

[51] B. Darmawan, Power systems and SOA synergy. [Poughkeepsie, NY]: International Technical Support Organization, 2008.

[52] C. Coelho, O. Koudelka, M. Sarkarati and M. Merri, "NanoSat MO Framework: Achieving On-board Software Portability", SpaceOps 2016 Conference, 2016.

[53] M. Koller, V. Reggestad, K. Adamson and R. Kay, "ESOC Earth Observation Missions and the Automation of Operational Routine Tasks", SpaceOps 2010 Conference, 2010.

[54] Mission Operations Reference Model. Issue 1. Recommendation for Space Data System Practices (Magenta Book), CCSDS 520.1-M-1. Washington, D.C.: CCSDS, July 2010.

[55] M. Merri, "Mission Operations Services by the CCSDS: a step towards the future", 2007. [Online]. Available: http://sunset.usc.edu/GSAW/gsaw2007/s2/merri.pdf. [Accessed: 02- Sep- 2017].

[56] C. Coelho, O. Koudelka and M. Merri, "NanoSat MO framework: When OBSW turns into apps", 2017 IEEE Aerospace Conference, 2017.

[57] "NanoSat-MO-Framework", GitHub, 2017. [Online]. Available: https://github.com/NanoSat-MO-Framework. [Accessed: 02- Sep- 2017].

[58] J. Gosling, B. Joy, G. Steele, G. Bracha and A. Buckley, The Java® language specification. Oracle America, Inc. and/or its affiliates, 2015. [Online]. Available: https://docs.oracle.com/javase/specs/jls/se8/jls8.pdf. [Accessed: 02- Sep- 2017].

[59] "The Java Language Environment", Oracle.com, 2017. [Online]. Available: http://www.oracle.com/technetwork/java/intro-141325.html. [Accessed: 02- Sep- 2017].

[60] Mission Operations Message Abstraction Layer – Java API. Issue 1. Recommendation for Space Data System Practices (Magenta Book), CCSDS 523.1-M-1. Washington, D.C.: CCSDS, April 2013.

[61] Mission Operations Message Abstraction Layer – C++ API. Issue 1. Draft Recommendation for Space Data System Practices (Red Book), CCSDS 523.2-R-1. Washington, D.C.: CCSDS, March 2017.

[62] "ccsdsmo/malc", GitHub, 2017. [Online]. Available: https://github.com/ccsdsmo/malc. [Accessed: 02- Sep- 2017].

[63] "European Space Agency", GitHub, 2017. [Online]. Available: https://github.com/esa. [Accessed: 02- Sep- 2017].

[64] J. Schütz, "EGOS User Desktop A Generic User Interface Framework for Ground Segment Software", 2014. [Online]. Available: http://gsaw.org/wp-content/uploads/2014/03/2014s12b_schutz.pdf. [Accessed: 02- Sep- 2017].

[65] N. Peccia, "The European Ground Systems – Common Core (EGS-CC) Initiative", 2012. [Online]. Available: http://csse.usc.edu/GSAW/gsaw2012/s3/peccia.pdf. [Accessed: 02- Sep- 2017].

[66] M. Sun, T. Wei and J. Lui, "TaintART", Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security - CCS'16, 2016.

[67] "Welcome to NetBeans", Netbeans.org, 2017. [Online]. Available: https://netbeans.org/. [Accessed: 02- Sep- 2017].

[68] "NetBeans NetBeans profiler", Profiler.netbeans.org, 2017. [Online]. Available: https://profiler.netbeans.org. [Accessed: 02- Sep- 2017].

[69] "Git", Git-scm.com, 2017. [Online]. Available: https://git-scm.com/. [Accessed: 02- Sep- 2017].

[70] GitLab, 2017. [Online]. Available: https://about.gitlab.com/about/. [Accessed: 02- Sep- 2017].

[71] "Continuous integration | ThoughtWorks", Thoughtworks.com, 2017. [Online]. Available: https://www.thoughtworks.com/continuous-integration. [Accessed: 02- Sep- 2017].

[72] "Jenkins", Jenkins, 2017. [Online]. Available: https://jenkins.io/. [Accessed: 02- Sep- 2017].

[73] B. Porter, J. Zyl and O. Lamy, "Maven – Welcome to Apache Maven", Maven.apache.org, 2017. [Online]. Available: https://maven.apache.org/. [Accessed: 02- Sep- 2017].

[74] E. Redmond, "Maven – POM Reference", Maven.apache.org, 2017. [Online]. Available: https://maven.apache.org/pom.html. [Accessed: 02- Sep- 2017].

[75] "TheNEXUS", TheNEXUS, 2017. [Online]. Available: http://books.sonatype.com/mvnex-book/reference/multimodule-sect-simple-parent.html. [Accessed: 02- Sep- 2017].

[74] "esa/CCSDS_MO_POM", GitHub, 2017. [Online]. Available: https://github.com/esa/CCSDS_MO_POM. [Accessed: 02- Sep- 2017].

[75] "esa/CCSDS_MO_StubGenerator", GitHub, 2017. [Online]. Available: https://github.com/esa/CCSDS_MO_StubGenerator. [Accessed: 02- Sep- 2017].

[76] "esa/CCSDS_MO_MAL_IMPL", GitHub, 2017. [Online]. Available: https://github.com/esa/CCSDS_MO_MAL_IMPL. [Accessed: 02- Sep- 2017].

[77] L. Das, The x86 microprocessors: Architecture And Programming (8086 To Pentium). New Delhi, India: Dorling Kindersley, 2010.

[78] "Multithreading (computer architecture)", En.wikipedia.org, 2017. [Online]. Available: https://en.wikipedia.org/wiki/Multithreading_(computer_architecture). [Accessed: 02- Sep- 2017].

[79] "What is multi-core processor? - Definition from WhatIs.com", SearchDataCenter, 2017. [Online]. Available: http://searchdatacenter.techtarget.com/definition/multi-core-processor. [Accessed: 27- Mar- 2017].

[80] Aater Suleman. "What makes parallel programming hard?". [Online]. Available: http://www.futurechips.org/tips-for-power-coders/parallel-programming.html. [Accessed: 02- Sep- 2017].

[81] M. Ben-Ari, Principles of Concurrent and Distributed Programming, Second Edition. Addison-Wesley, 2006.

[82] J. Gosling, The Java language specification. Harlow: Prentice Hall, 2013.

[83] "java.util.concurrent (Java Platform SE 7)", Docs.oracle.com, 2017. [Online]. Available: https://docs.oracle.com/javase/7/docs/api/java/util/concurrent/package-summary.html. [Accessed: 02- Sep- 2017].

[84] "Fallacies of Distributed Computing Explained". [Online]. Available: http://www.rgoarchitects.com/Files/fallacies.pdf. [Accessed: 02- Sep- 2017].

[85] "Structured Query Language (SQL)", Docs.microsoft.com, 2017. [Online]. Available: https://docs.microsoft.com/en-us/sql/odbc/reference/structured-query-language-sql. [Accessed: 02- Sep- 2017].

[86] E. Jendrock, The Java EE 6 tutorial. Upper Saddle River [etc.]: Addison-Wesley, 2013. [Online]. Available: http://docs.oracle.com/javaee/6/tutorial/doc/bnbpz.html. [Accessed: 02- Sep- 2017].

[87] "EclipseLink", Eclipse.org, 2017. [Online]. Available: http://www.eclipse.org/eclipselink/. [Accessed: 02- Sep- 2017].

[88] "xerial/sqlite-jdbc", GitHub, 2017. [Online]. Available: https://github.com/xerial/sqlite-jdbc. [Accessed: 02- Sep- 2017].

[89] "SQLite Home Page", Sqlite.org, 2017. [Online]. Available: https://www.sqlite.org/. [Accessed: 02- Sep- 2017].

[90] "DB Browser for SQLite", Sqlitebrowser.org, 2017. [Online]. Available: http://sqlitebrowser.org/. [Accessed: 02- Sep- 2017].

[91] "andreeap/CCSDS-MO-Login-Service-Implementation", GitHub, 2017. [Online]. Available: https://github.com/andreeap/CCSDS-MO-Login-Service-Implementation. [Accessed: 02- Sep- 2017].

[92] ESA Summer Of Code In Space (SOCIS). [Online]. Available: http://sophia.estec.esa.int/socis/. [Accessed: 02- Sep- 2017].

[93] "File:ObjectAdapter.png - Wikimedia Commons", Commons.wikimedia.org, 2017. [Online]. Available: https://commons.wikimedia.org/wiki/File:ObjectAdapter.png. [Accessed: 02- Sep- 2017].

[94] S. Bamfaste, M. Cardone, M. Azkarate, S. Martin, M. Sarkarati, M. Merri, "Introducing CCSDS Mission Operations services to the Meteron Operations Environment", 14th Symposium on Advanced Space Technologies in Robotics and Automation, 2017

[95] ECSS-E-ST-40C, "Software". Space engineering. ESA-ESTEC, 2009.

[96] S. Suteu, "OPS-SAT Software Simulator", Master Thesis, Luleå University of Technology, Department of Computer Science, Electrical and Space Engineering, 2016

[97] "Orekit", Orekit.org, 2017. [Online]. Available: https://www.orekit.org/. [Accessed: 02- Sep- 2017].

[98] CAN Specification. Bosch. 1991. [Online]. Available: http://www.bosch-semiconductors.de/media/ubk_semiconductors/pdf_1/canliteratur/can2spec.pdf. [Accessed: 02- Sep- 2017].

[99] "dschanoeh/socketcand", GitHub, 2017. [Online]. Available: https://github.com/dschanoeh/socketcand. [Accessed: 02- Sep- 2017].

[100] C. Haddow, C. Müller, G. Scotti and T. Ulriksen, "Network Interface System (NIS); ESAs Next Generation CCSDS SLE Interface System", SpaceOps 2006 Conference, 2006.

[101] P. Steele, F. Flentge, J. Schütz, M. Pecchioli, "ESOC New Generation M&C User Interfaces", SpaceOps 2012 Conference, 2012.

[102] "[3.16,202/305] can: c_can: Update D_CAN TX and RX functions to 32 bit - fix Altera Cyclone access - Patchwork", Patchwork.kernel.org, 2017. [Online]. Available: https://patchwork.kernel.org/patch/9279395/. [Accessed: 02- Sep- 2017].

[103] HYPERION TECHNOLOGIES, ST200 Star Tracker - Datasheet. 2016. [Online]. Available: http://hyperiontechnologies.nl/wp-content/uploads/2016/08/HTBST-ST200-V1.0_Flyer.pdf. [Accessed: 02- Sep- 2017].

[104] OEM6® Family, Firmware Reference Manual - Datasheet. 2017. [Online]. Available: https://www.novatel.com/assets/Documents/Manuals/om-20000129.pdf. [Accessed: 02- Sep- 2017].

[105] "Berlin Space Technologies | iADCS", Berlin-space-tech.com, 2017. [Online]. Available: https://www.berlin-space-tech.com/portfolio/iadcs/. [Accessed: 02- Sep- 2017].

[106] C. Coelho, M. Merri, O. Koudelka and M. Sarkarati, "OPS-SAT Experiments' Software Management with the NanoSat MO Framework", AIAA SPACE 2016, 2016.

[107] S. Bamfaste, "Development of a Software Layer for the Integration of Robotic Elements into the METERON Infrastructure using Robotic Services", Master Thesis, Luleå University of Technology, 2016

[108] V. Krishnan, Oracle ADF 11gR2 development beginner's guide. Birmingham, UK: Packt Pub, 2013.

[109] "Libvirt Sandbox An application sandbox toolkit", Sandbox.libvirt.org, 2017. [Online]. Available: http://sandbox.libvirt.org/. [Accessed: 02- Sep- 2017].

[110] "What is Docker", Docker, 2017. [Online]. Available: https://www.docker.com/what-docker. [Accessed: 02- Sep- 2017].

[111] Taesoo Kim and Nickolai Zeldovich. Practical and effective sandboxing for non-root users. In USENIX Annual Technical Conference, pages 139–144, 2013.

[112] "Firejail". [Online]. Available: https://firejail.wordpress.com/. [Accessed: 02- Sep- 2017].

[113] "Yamcs", Yamcs.org, 2017. [Online]. Available: http://www.yamcs.org/. [Accessed: 02- Sep- 2017].

[114] "nasa/openmct", GitHub, 2017. [Online]. Available: https://github.com/nasa/openmct. [Accessed: 02- Sep- 2017].

[115] "Android", Android, 2017. [Online]. Available: https://www.android.com/. [Accessed: 02- Sep- 2017].

[116] "Google Code Archive - Long-term storage for Google Code Project Hosting.", Code.google.com, 2017. [Online]. Available: https://code.google.com/archive/p/s-android/. [Accessed: 02- Sep- 2017].

[117] U. Catalunya, "Android Beyond the Stratosphere — NanoSat Lab — UPC. Universitat Politècnica de Catalunya", Nanosatlab.upc.edu, 2017. [Online]. Available: https://nanosatlab.upc.edu/en/missions-and-projects/android-beyond-the-stratosphere. [Accessed: 02- Sep- 2017].

[118] "Android Beyond the Stratosphere", GitHub, 2017. [Online]. Available: https://github.com/abs-platform. [Accessed: 02- Sep- 2017].

[119] "JUNIPER Project", Juniper-project.org, 2017. [Online]. Available: http://www.juniper-project.org/. [Accessed: 02- Sep- 2017].

[120] I. Gray, Y. Chan, J. Garside, N. Audsley, A. Wellings, "Transparent hardware synthesis of Java for predictable large-scale distributed systems". Second International Workshop on FPGAs for Software Programmers (FSP 2015), 2015.

[121] Spacecraft Onboard Interface Services--XML Specification for Electronic Data Sheets. Draft Recommendation for Space Data System Standards (Red Book), CCSDS 876.0-R-2. Washington, D.C.: CCSDS, June 2016.

[122] CCSDS File Delivery Protocol (CFDP), Issue 4, Recommendation for Space Data System Standards (Blue Book), CCSDS 727.0-B-4, Washington, D.C.: CCSDS, January 2007.

[123] CCSDS Bundle Protocol Specification, Issue 1, Recommendation for Space Data System Standards (Blue Book), CCSDS 734.2-B-1, Washington, D.C.: CCSDS, September 2015.

[124] Space Packet Protocol, Issue 1. Recommendation for Space Data System Standards (Blue Book), CCSDS 133.0-B-1. Washington, D.C.: CCSDS, September 2003.

[125] R. Thompson, "MOIMS Services – Inputs to SEA Reference Architecture", CCSDS Technical Meetings, Cleveland, USA. 2016.

[126] "Interpretation of Time and FineTime values should be specified · Issue #10 · SamCooper/JAVA_SPEC_RIDS", GitHub, 2017. [Online]. Available: https://github.com/SamCooper/JAVA_SPEC_RIDS/issues/10. [Accessed: 02- Sep- 2017].

[127] Dan Vergano, National Geographic. "NASA Declares End to Deep Impact Comet Mission", News.nationalgeographic.com, 2013. [Online]. Available: http://news.nationalgeographic.com/news/2013/09/130920-deep-impact-ends-comet-mission-nasa-jpl/. [Accessed: 02- Sep- 2017].

[128] Telerobotic Operations, Report Concerning Space Data System Standards (Green Book), CCSDS 540.0-G-1, Washington, D.C.: CCSDS, November 2016.

[129] "Spacesuit Simulator - Österreichisches Weltraum Forum (ÖWF)", Österreichisches Weltraum Forum (ÖWF), 2017. [Online]. Available: http://oewf.org/en/polares-science/aouda-spacesuit-simulator/. [Accessed: 02- Sep- 2017].

[130] Mission Operations Message Abstraction Layer. Issue 2. Recommendation for Space Data System Standards (Blue Book), CCSDS 521.0-B-2. Washington, D.C.: CCSDS, March 2013.

[131] "Final Report: CCSDS MO Services, CCSDS SOIS, and SAVOIR for Future Spacecraft", BAL-P-MOSS-D09, ESA, May 2016.