



esoc

European Space Operations Centre
Robert-Bosch-Strasse 5
D-64293 Darmstadt
Germany
T +49 (0)6151 900
F +49 (0)6151 90495
www.esa.int

DOCUMENT

NanoSat MO Framework - Software Design Document

Prepared by Cesar Coelho
Reference
Issue
Revision
Date of Issue
Status Draft
Document Type TN
Distribution

1. CONTENTS

2. INTRODUCTION.....	4
3. NANOSAT MO FRAMEWORK.....	5
3.1. Principles.....	5
3.1.1. Core Functionality.....	5
3.1.2. Problem domains.....	7
3.1.3. On-board software portability.....	11
3.2. CCSDS MO Heritage.....	16
3.2.1. MO Core.....	17
3.2.2. Transport Binding and Encoding.....	20
3.2.3. Monitor and Control services.....	21
3.3. Software Management services.....	23
3.4. Platform services.....	27
3.5. NMF Generic Model.....	31
3.6. NMF Composites.....	34
3.6.1. NanoSat MO Monolithic.....	35
3.6.2. NanoSat MO Supervisor.....	36
3.6.3. NanoSat MO Connector.....	38
3.6.4. Ground MO Adapter.....	40
3.6.5. Ground MO Proxy.....	41
3.7. NMF Concepts.....	44
3.7.1. NMF App.....	44
3.7.2. NMF Ground application.....	45
3.7.3. NMF Package.....	46
3.7.4. NMF Mission.....	47
3.7.5. NMF Library.....	48
3.7.6. NMF SDK.....	49
3.8. Generic scenarios.....	50
3.8.1. Simple Scenario.....	50
3.8.2. Multiple Consumers Scenario.....	51
3.8.3. Multiple NMF Apps Scenario.....	51
3.8.4. Proxy Scenario.....	52
3.8.5. Other Envisaged Scenarios.....	52
3.8.6. Complex Scenario Example.....	53
4. NANOSAT MO FRAMEWORK: JAVA IMPLEMENTATION.....	55
4.1. Rationale.....	55
4.2. Capitalizing on existing tools, technologies and software.....	58
4.2.1. Netbeans.....	58
4.2.2. Version Control System.....	59
4.2.3. Continuous Integration.....	60
4.2.4. Dependencies management.....	61
4.2.5. MO software reuse.....	61
4.2.6. Multithreading.....	62
4.3. Implementation.....	64
4.3.1. Implementation Considerations.....	65



4.3.2. COM services65

4.3.3. Common services..... 68

4.3.4. M&C services 69

4.3.5. Platform services implementation 71

4.3.6. Software Management services..... 71

4.3.7. NMF Composites73

4.3.8. Integration of NMF Package concept 75

4.4. Generic Deployment76

4.5. Reuse in other ESA Projects78



2. INTRODUCTION

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 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.

The NanoSat Mission Operations (MO) framework provides a standard on-board software framework that facilitates not only the monitoring and control of the nanosatellite, but also the interaction with its platforms and payload. This is achieved by using the MO services for Monitor and Control services included in the MO service suite and by defining a set of new Platform services, which also follow the MO services architecture.

The MO services are a set of standardized end-to-end services based on a service-oriented architecture which are currently being defined by the Consultative Committee for Space Data Systems (CCSDS) and they are intended to be used for mission operations of future space assets.

The CCSDS standard for Monitor and Control services will allow any consumer to get parameters and aggregations values, to dynamically reconfigure the aggregations, create control checks to verify if the app is operating correctly, receive statistics and check notifications while the application will also be able to raise alerts to the consumers in case something unexpected happens.

3. NANOSAT MO FRAMEWORK

The NanoSat MO Framework (NMF) is introduced and described in detail.

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 other documents.

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.

3.1. Principles

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

3.1.1. 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.

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.

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.

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.

3.1.2. 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.

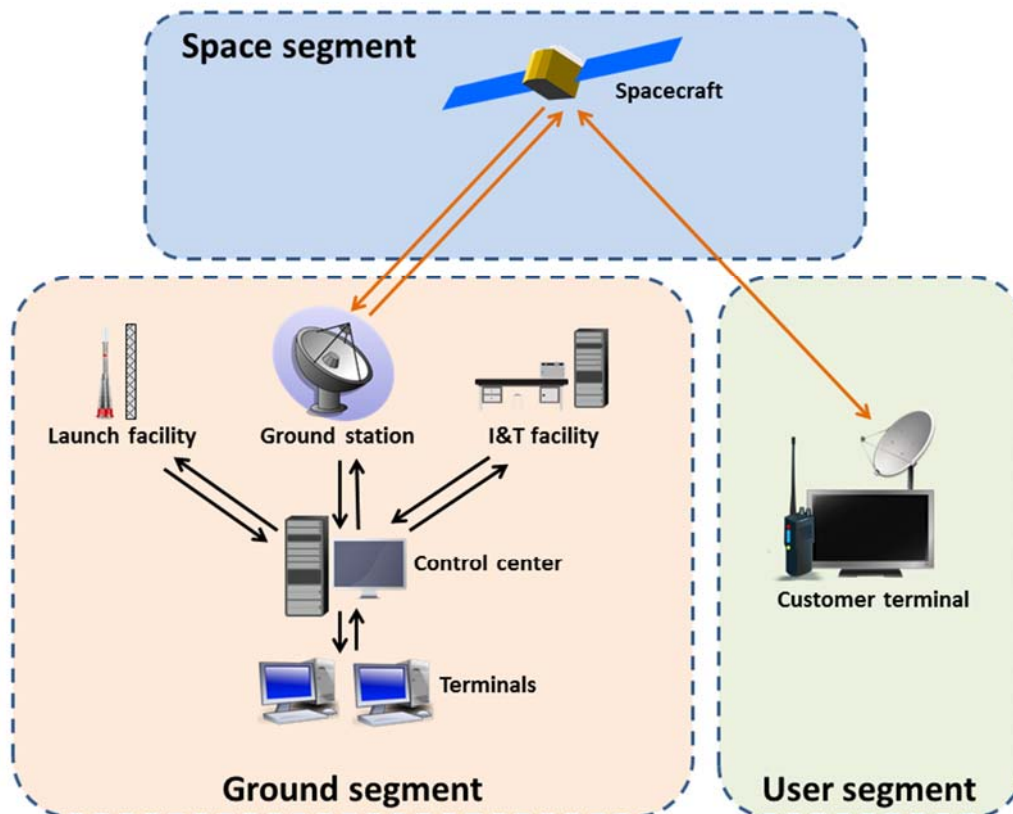


Figure 7: Example of a spacecraft system

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.

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.

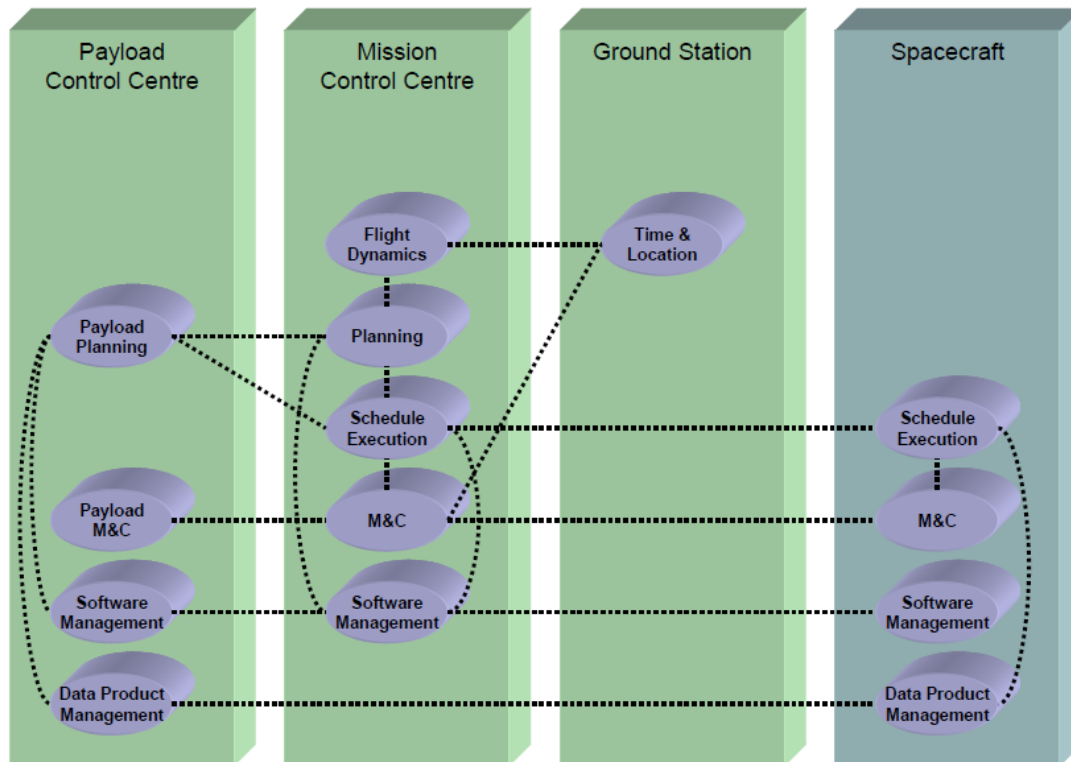


Figure 8: Mission operations activities distributed between a Spacecraft and three separate Ground Segment facilities.

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. 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.

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.

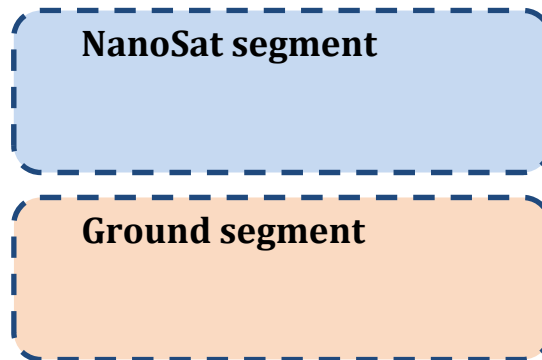


Figure 9: Spacecraft system of the NMF.

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.

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.

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.

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.

3 Problem Domains



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 3.6, 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.

3.1.3. 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.

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.

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.

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.

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.

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.

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.

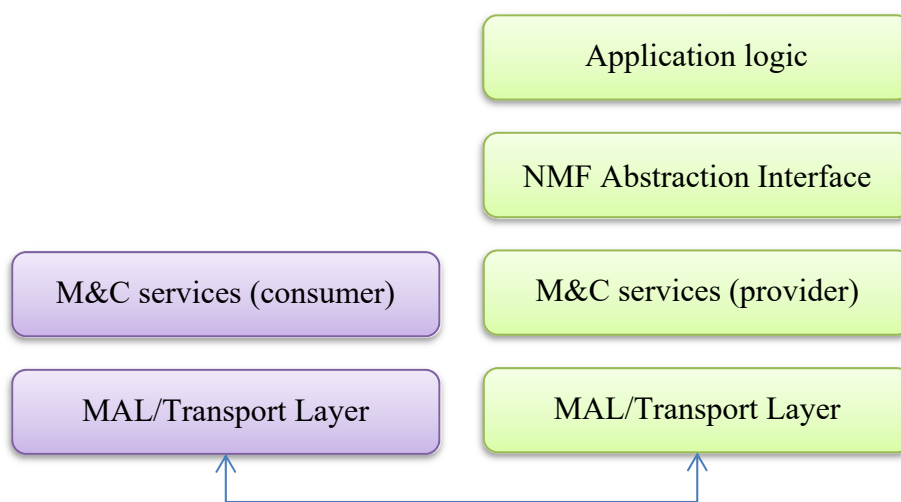


Figure 12: Consumer and Provider side for generic monitoring and control of an application's logic.

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 3.4 explains these services in further detail.

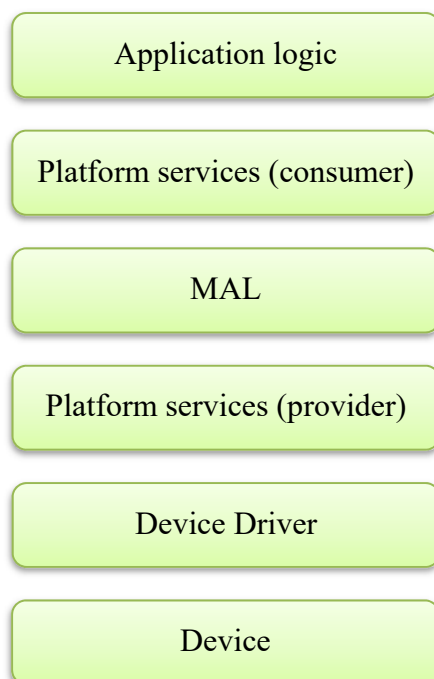


Figure 13: Monitoring and control of a nanosatellite platform’s device using the Platform services with a single process.

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.

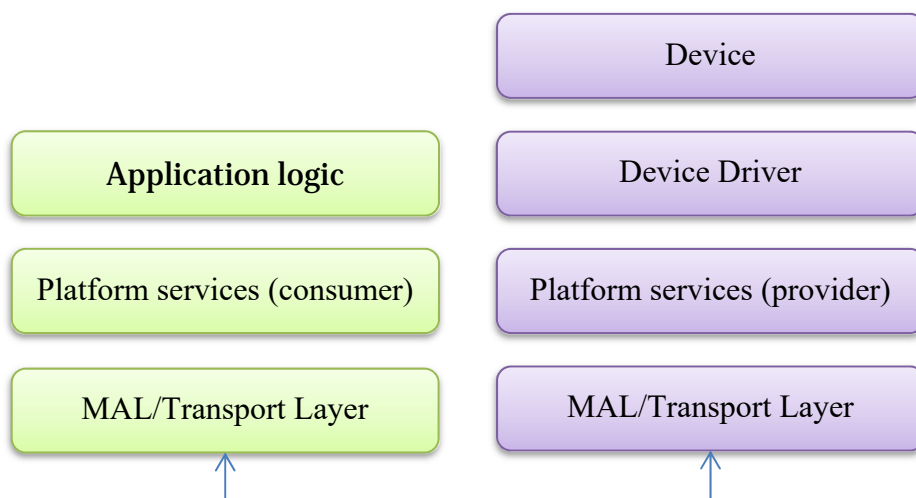


Figure 14: Monitoring and control of a nanosatellite platform's device using the Platform services with two processes.

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 3.6 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.

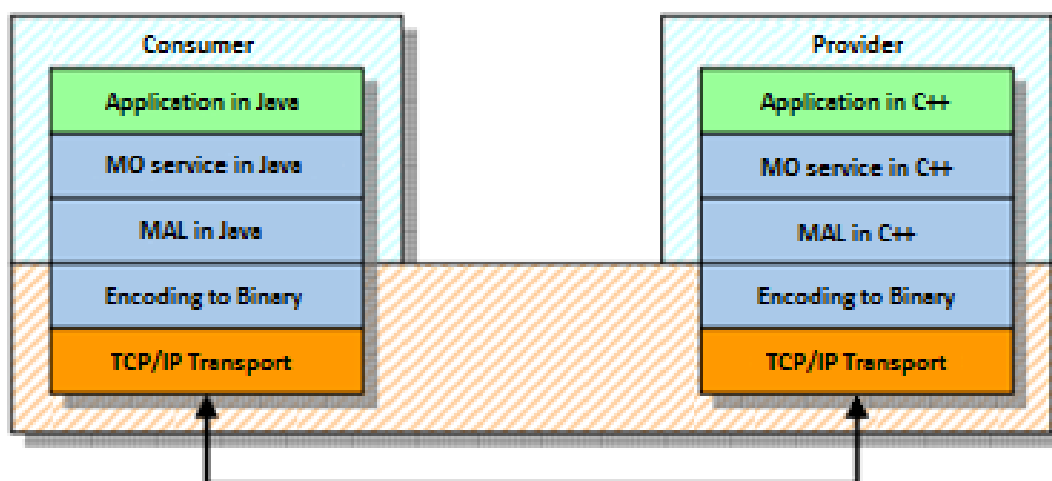


Figure 15: A consumer and a provider developed in different programming languages.



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.

3.2. 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 3.5, 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 3.2.1 while the M&C services are described in sub-section 3.2.3. Figure 16 presents an exploded view of the MO architecture including its sub-layers.

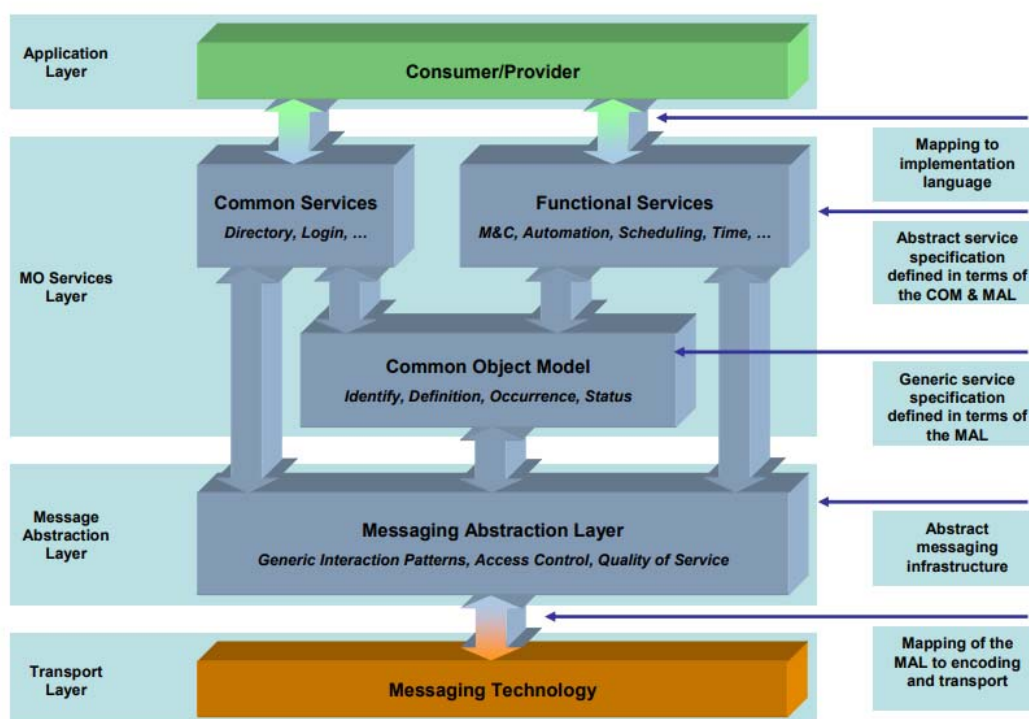




Figure 16: Exploded view of the MO architecture.

Section 3.2.1 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.

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	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.

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.

3.2.1.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.

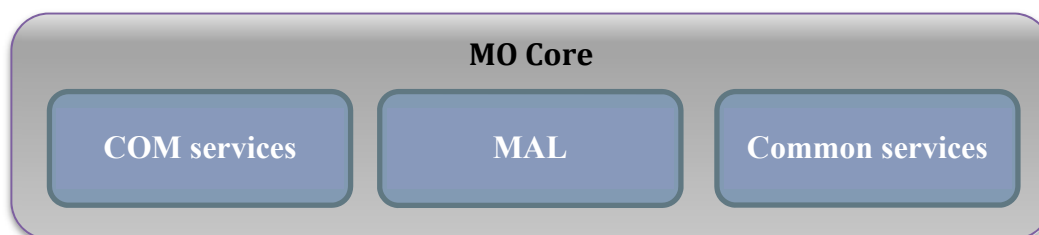


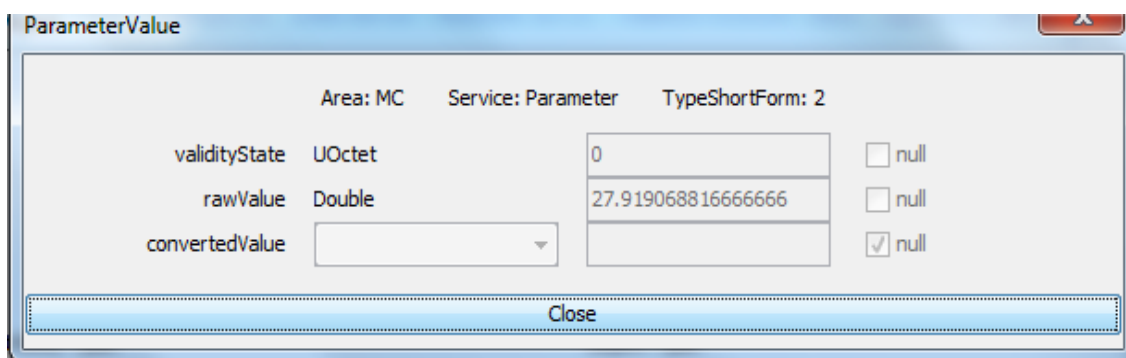
Figure 18: The MO Core with COM services, MAL, and Common service.

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.



Area: MC Service: Parameter TypeShortForm: 2			
validityState	UOctet	0	<input type="checkbox"/> null
rawValue	Double	27.919068816666666	<input type="checkbox"/> null
convertedValue			<input checked="" type="checkbox"/> null

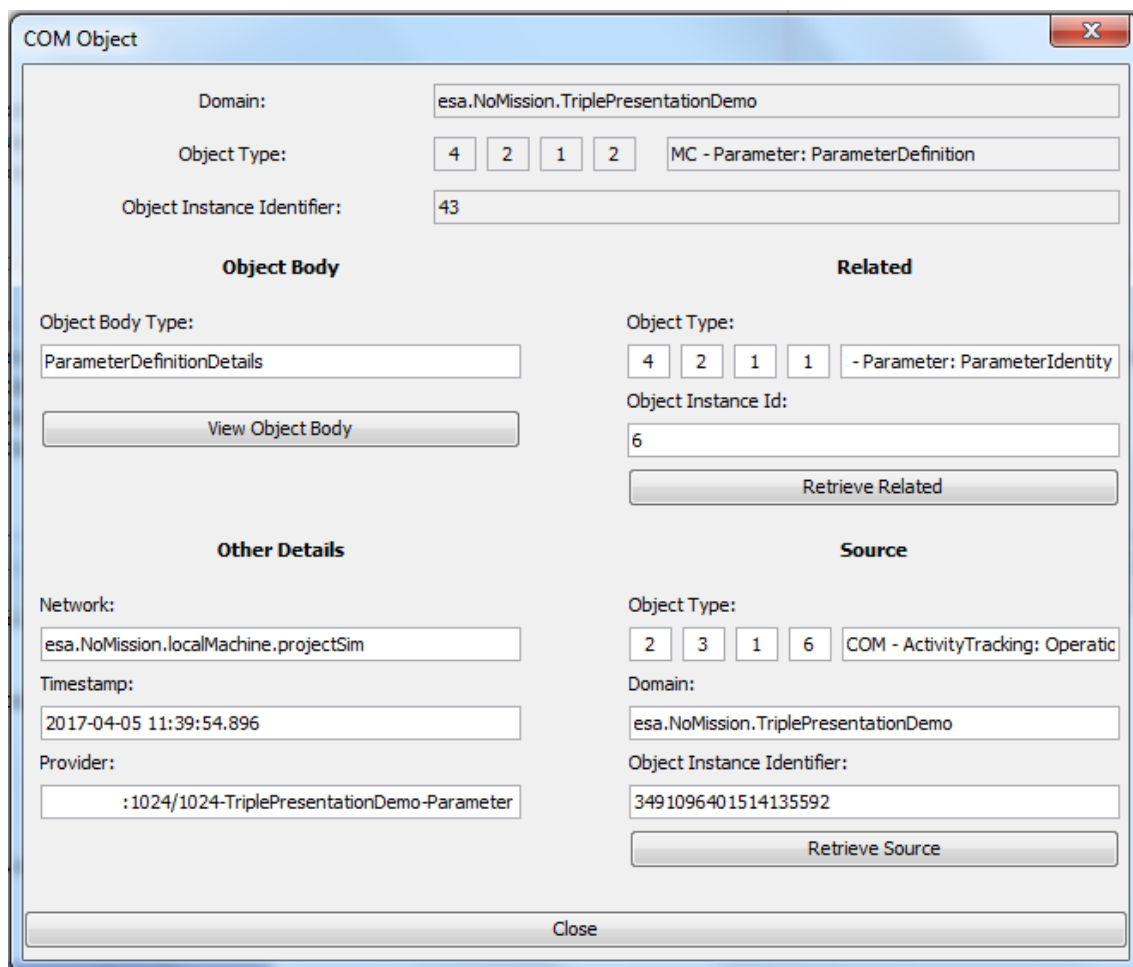
Close

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.

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.

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:

Object Type:

Object Instance Identifier:

Object Body

Object Body Type:

Related

Object Type:

Object Instance Id:

Other Details

Network:

Timestamp:

Provider:

Source

Object Type:

Domain:

Object Instance Identifier:

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:

- 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.

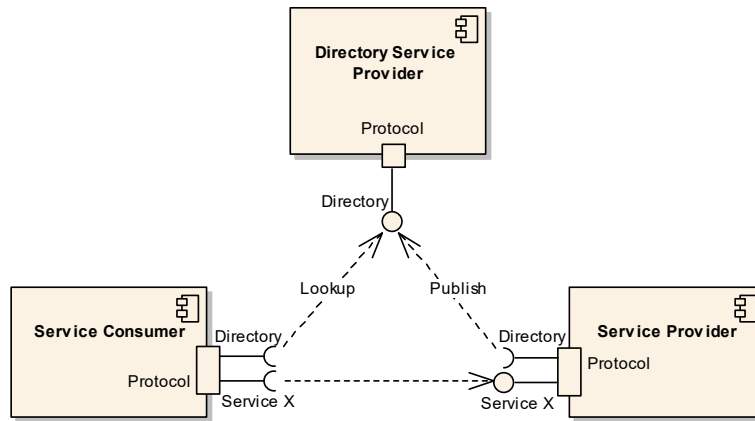


Figure 21: Directory service concept.

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.

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.

3.2.2. 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
- TCP/IP Transport Binding
- 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
- Split Binary Binding
- 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.

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.

3.2.3. *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.

The main services are:

- 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:

- 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.

3.3. 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's 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.

3.4. 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.

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.

3.5. 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.

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.

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.

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.

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 Autonomous ADCS 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.

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 3.3 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 3.7.3.

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.

If a mission needs to use a mission-specific transport, a protocol bridge can be used to accommodate this need. Section 3.6.5 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 3.2.1.

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 3.6. They can produce complete applications that are able to operate in different system scenarios such as the ones described in section 3.8.

3.6. 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.

While section 3.5 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.

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 3.8.

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:

- 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.

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.

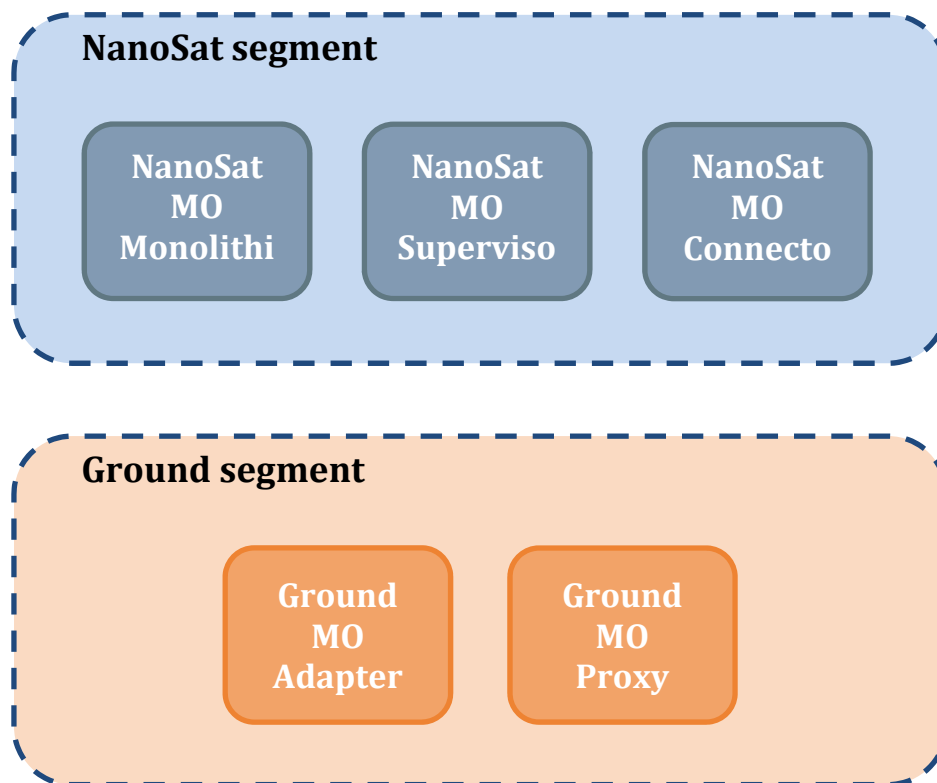


Figure 23: NMF Composites on their respective deployment segment.

3.6.1. 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 3.1.3 includes more details about the deployment of the Platform services in a single process.

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.

3.6.2. *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.

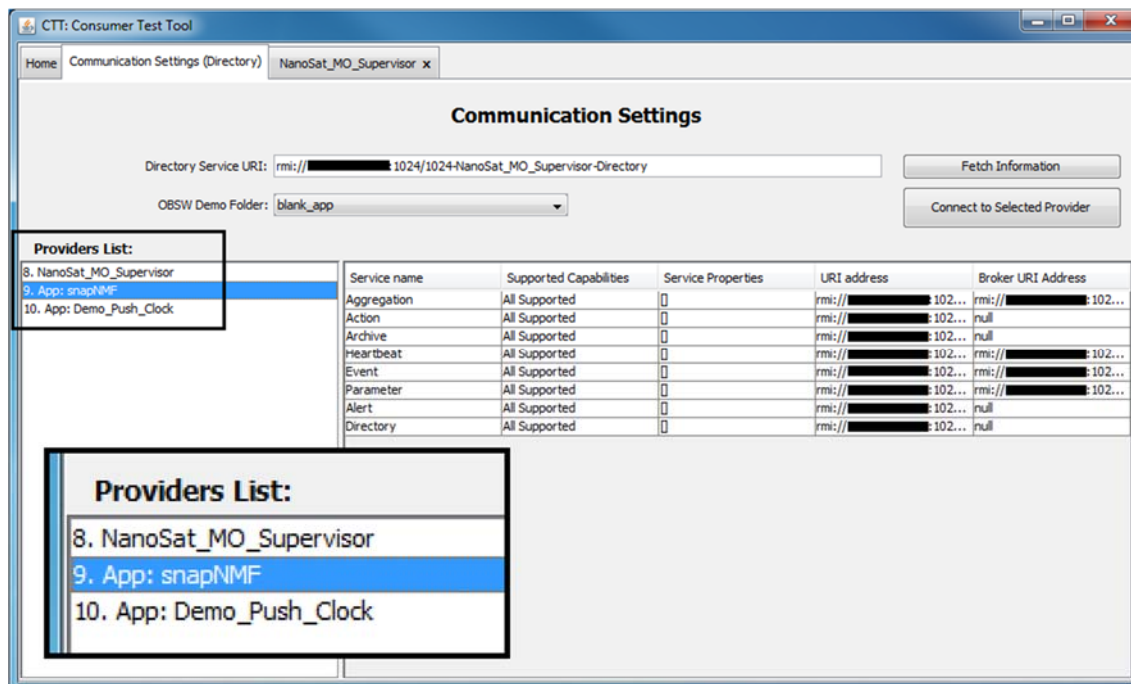


Figure 25: A zoom-in visualization of the 3 providers available on a Central Directory service presented on the Consumer Test Tool.

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.

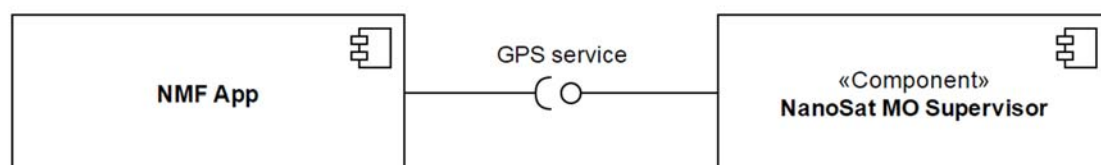


Figure 26: Component diagram showing an NMF App connected to the GPS service of the NanoSat MO Supervisor.

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.

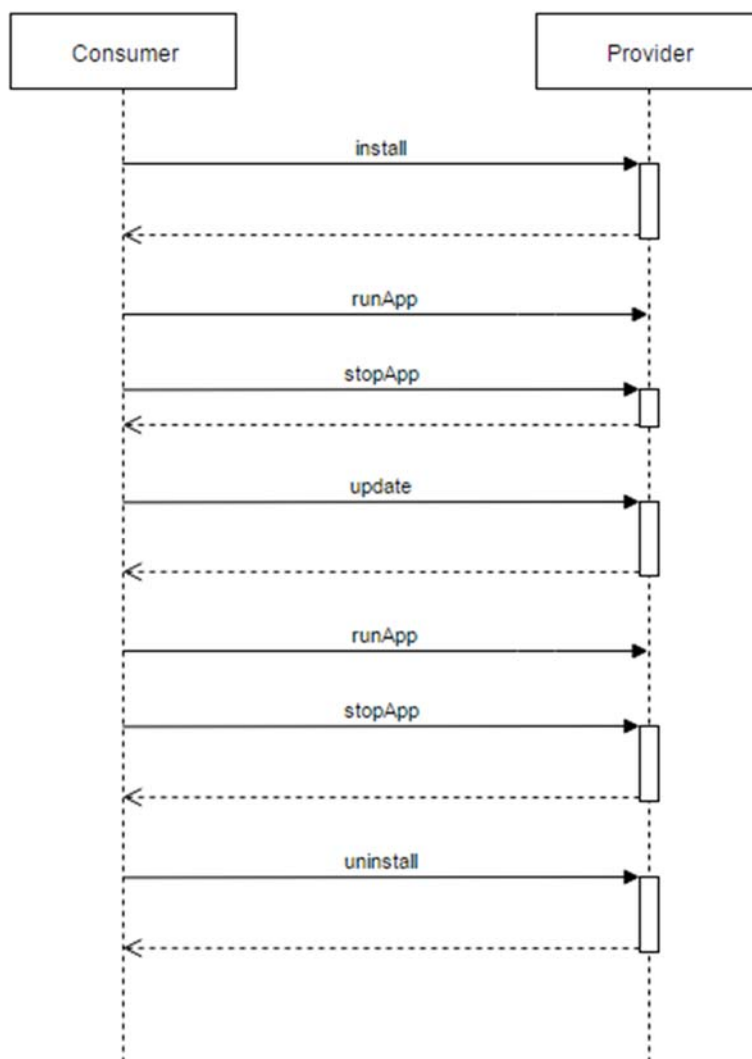


Figure 27: Sequence Diagram of the expected lifecycle of an application.

3.6.3. *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.

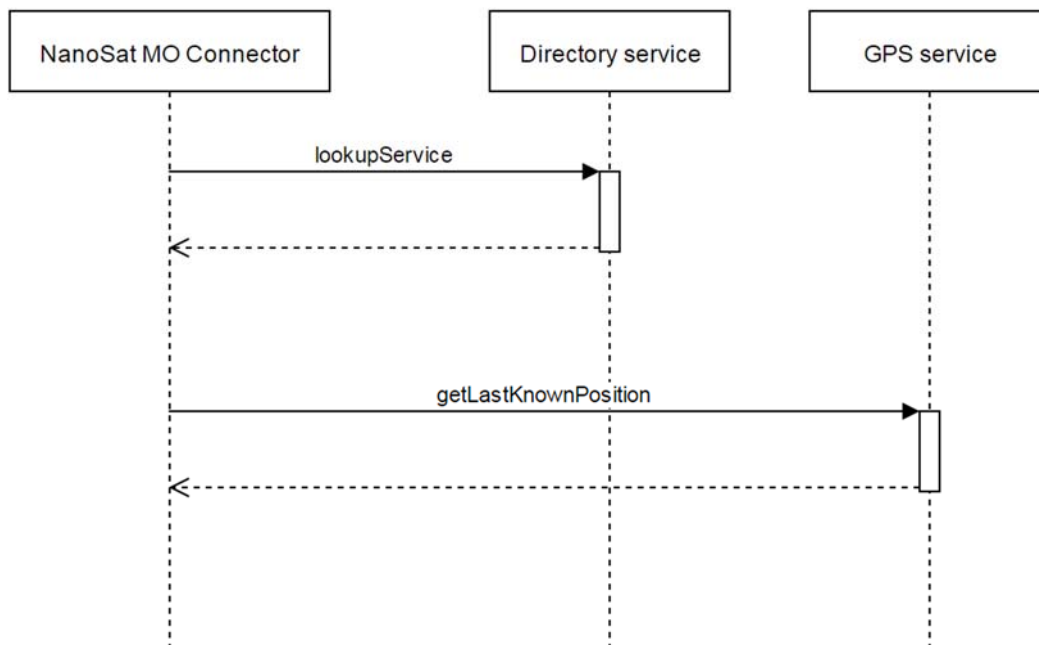


Figure 28: Sequence Diagram of an example showing the NanoSat MO Connector finding the GPS service address and then calling an operation.

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 graceful 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.

3.6.4. *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.

The Ground MO Adapter opens many possibilities for the integration with other ground systems.

3.6.5. *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.

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.

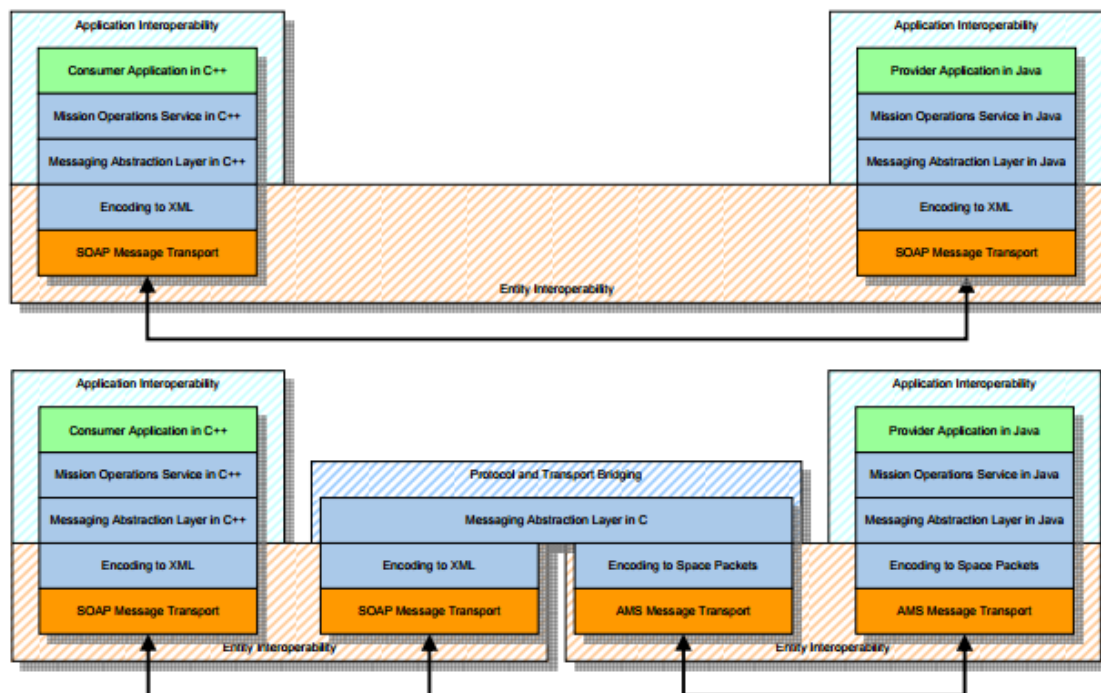


Figure 30: Protocol Bridge Example

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.

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.

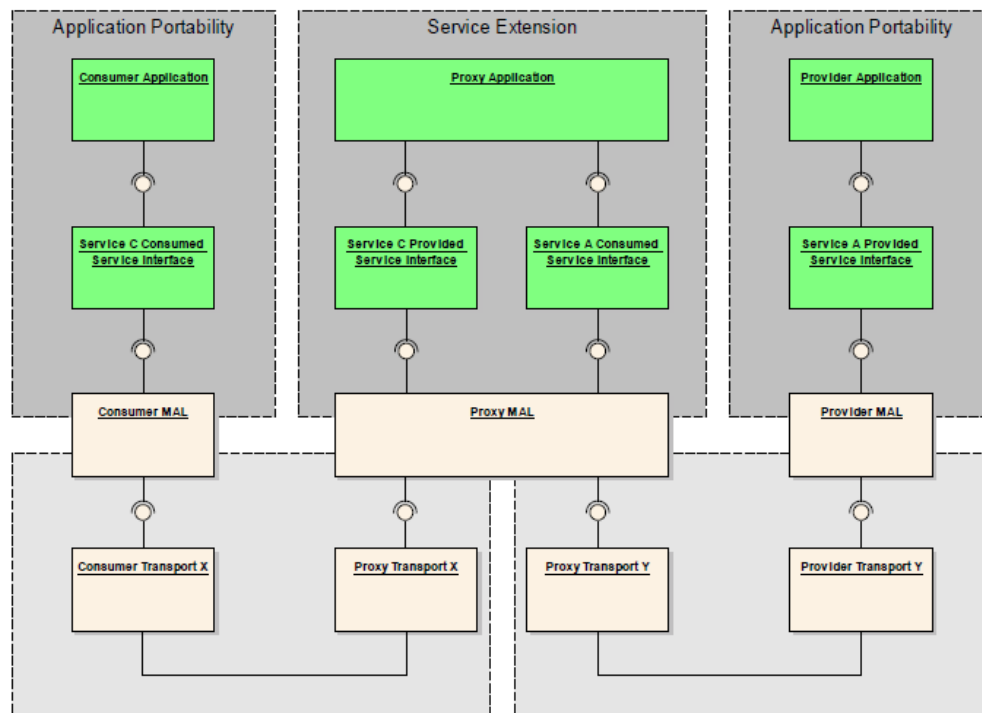


Figure 31: Service Extension Example

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.

3.7. NMF Concepts

This section defines the NMF Concepts of the NanoSat MO Framework. Some of them are interconnected.

3.7.1. 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.

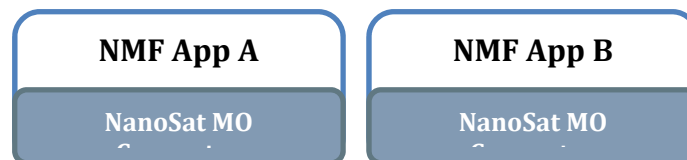


Figure 32: Conceptual visualization of two NMF Apps.

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 3.1.3 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 CCSDS 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 3.7.3.

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.

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.

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.

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 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 3.1.2.

3.7.2. 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.

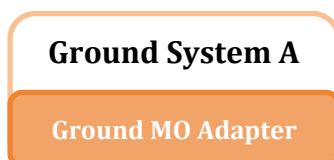


Figure 33: Conceptual visualization of a NMF Ground application.

3.7.3. 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.

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.

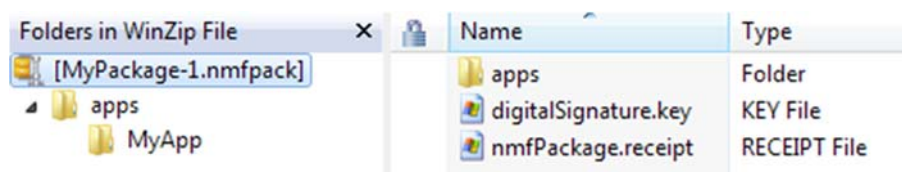


Figure 34: Content of an NMF Package.

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.

```

1 NMFPackageDescriptorVersion=1
2 PackageName=MyPackage
3 PackageVersion=1
4 PackageCreationTimestamp=2017-10-29 09:33:17.130
5 FilePath=apps\MyApp\Demo_All_In_One.jar
6 FileCRC=3733458777
7 FilePath=apps\MyApp\provider.properties
8 FileCRC=2987959643
9 FilePath=apps\MyApp\runAppLin.sh
10 FileCRC=3718171052
11 FilePath=apps\MyApp\runAppWin.bat
12 FileCRC=1530203412
13

```

Figure 35: Example of an NMF Package receipt file.

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. The possibility for an open repository of NMF Packages online would increase the reuse of software among nanosatellite developers.

3.7.4. 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 3.8 presents some generic scenarios that can be used to build something more complex.

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.

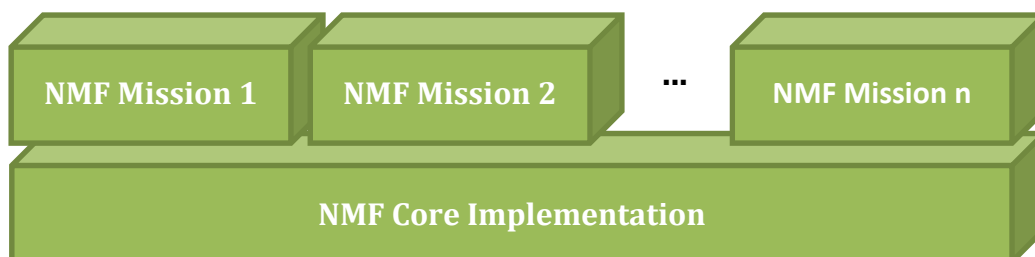


Figure 36: NMF Missions developed on top of an NMF Implementation

As mentioned in section 3.1.2, 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 3.8 describes.

3.7.5. *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.

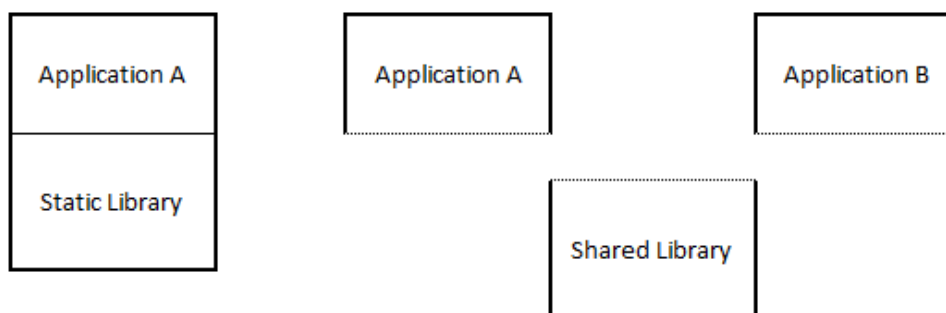


Figure 37: Difference between a static library and a shared library in relation to the application.



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.

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.

3.7.6. 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 SDK execution environment
- Documentation

The NMF SDK is the starting point for a software developer willing to develop applications with the NMF.

3.8. 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 3.8.6.

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.

3.8.1. *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.

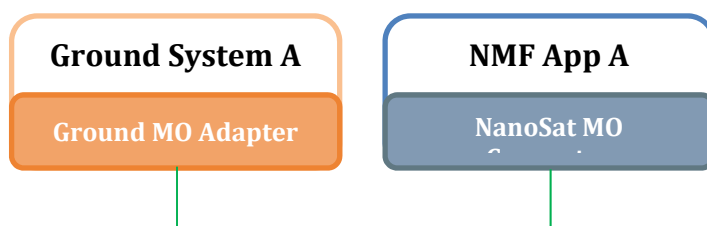


Figure 38: Simple Scenario example.

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.

3.8.2. 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.

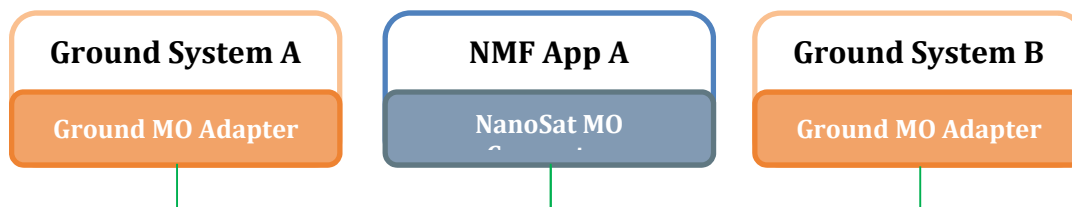


Figure 39: Multiple Consumers Scenario example.

3.8.3. 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.

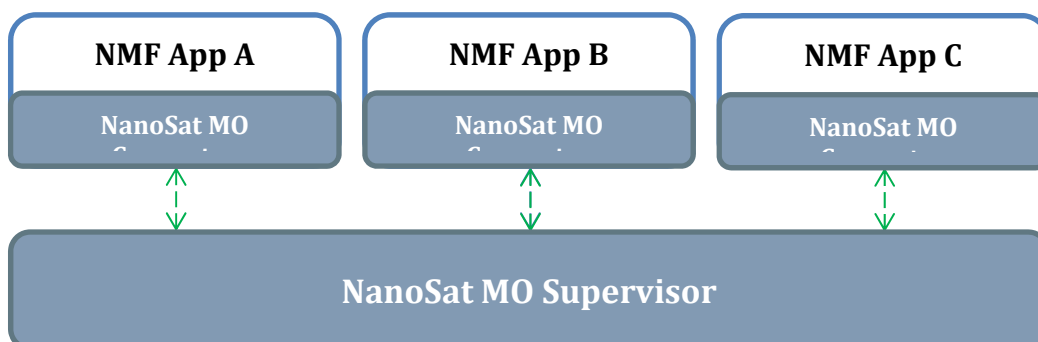


Figure 40: Multiple NMF Apps Scenario example.

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.

3.8.4. *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 41 shows an example.

The purpose and added functionality of having a Ground MO Proxy is presented in section 3.6.5. 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.

3.8.5. *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.

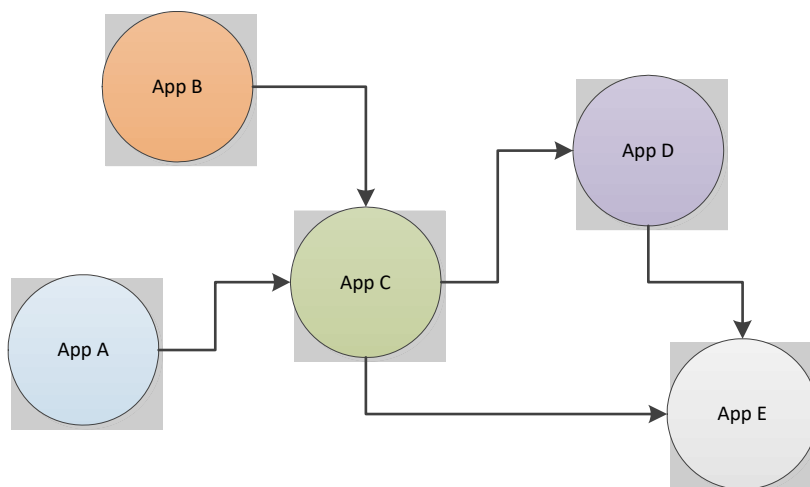


Figure 42: Cascading NMF Apps Scenario.

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.

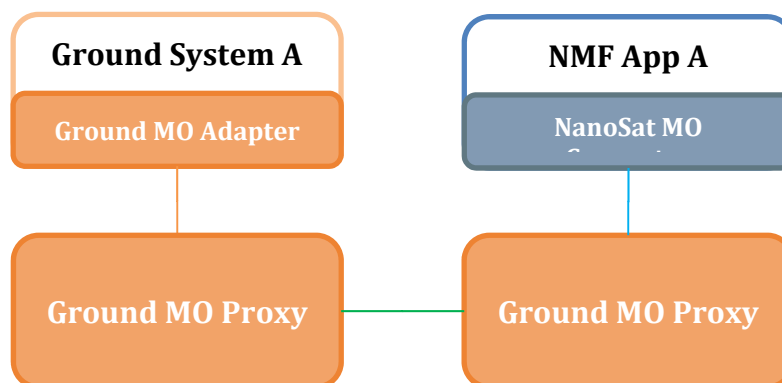


Figure 43: Multiple Ground MO Proxies Scenario.

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.

3.8.6. *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.

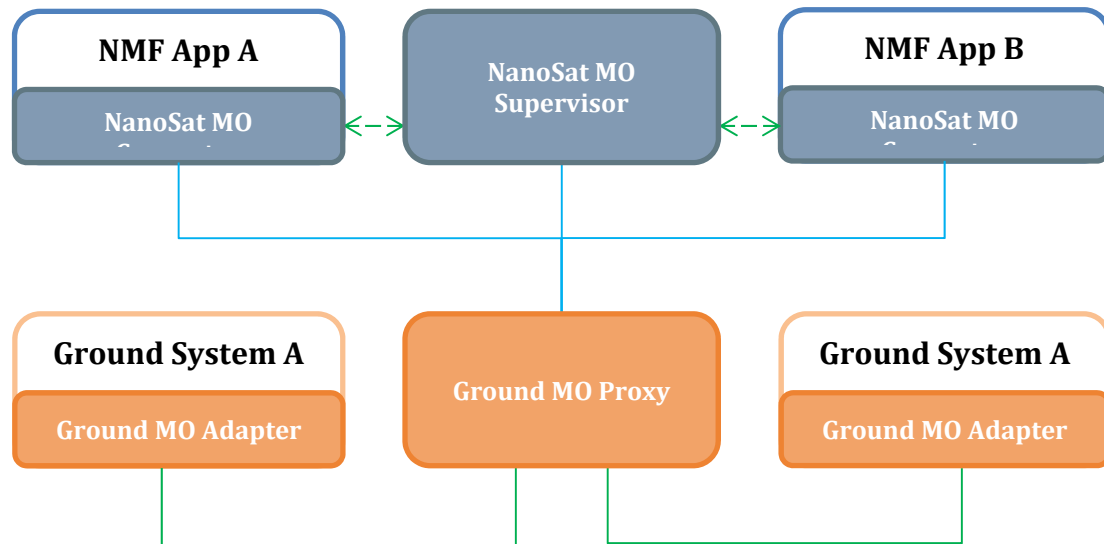


Figure 44: Complex Scenario example.

4. NANOSAT MO FRAMEWORK: JAVA IMPLEMENTATION

In this section, the Java implementation of the NanoSat MO Framework is presented and the most important information is described.

It can be considered as a reference implementation because it provides a concrete interpretation for the “specifications” presented in the previous section 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 section 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 section 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.

4.1. 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.

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.

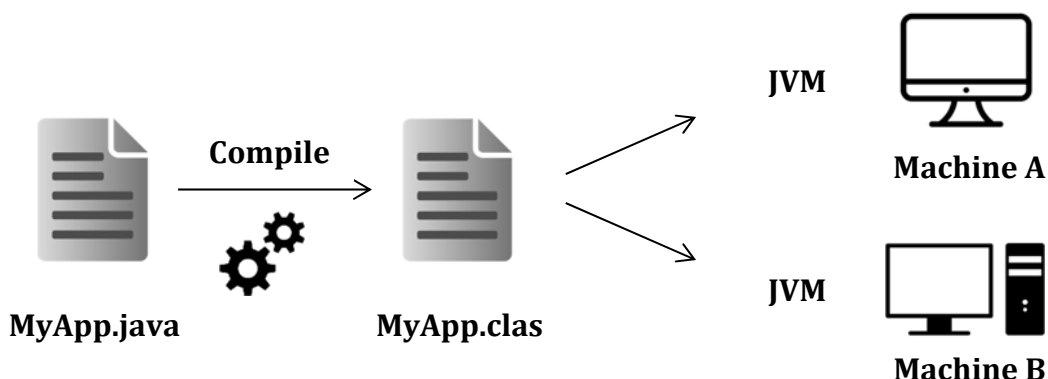


Figure 45: The process of running a Java application in JVMs for two different machines.

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.

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.

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.

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.

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.

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.



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.

4.2. 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.

4.2.1. 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.

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.

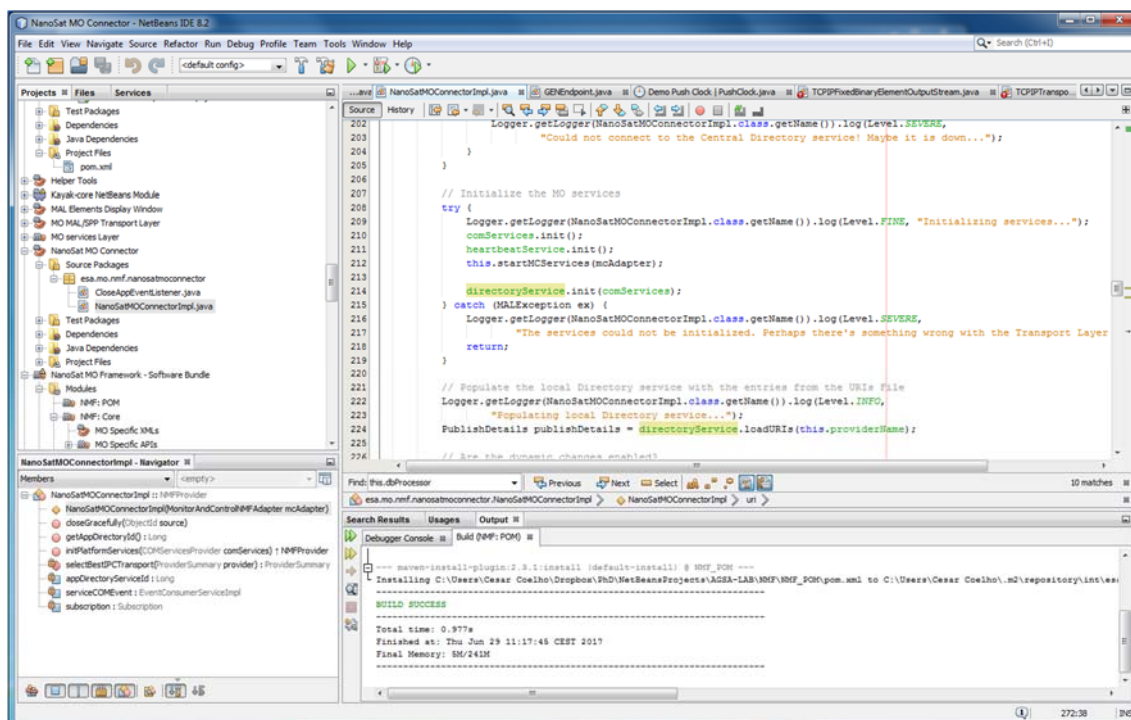


Figure 46: Netbeans IDE screenshot.

4.2.2. 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.

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

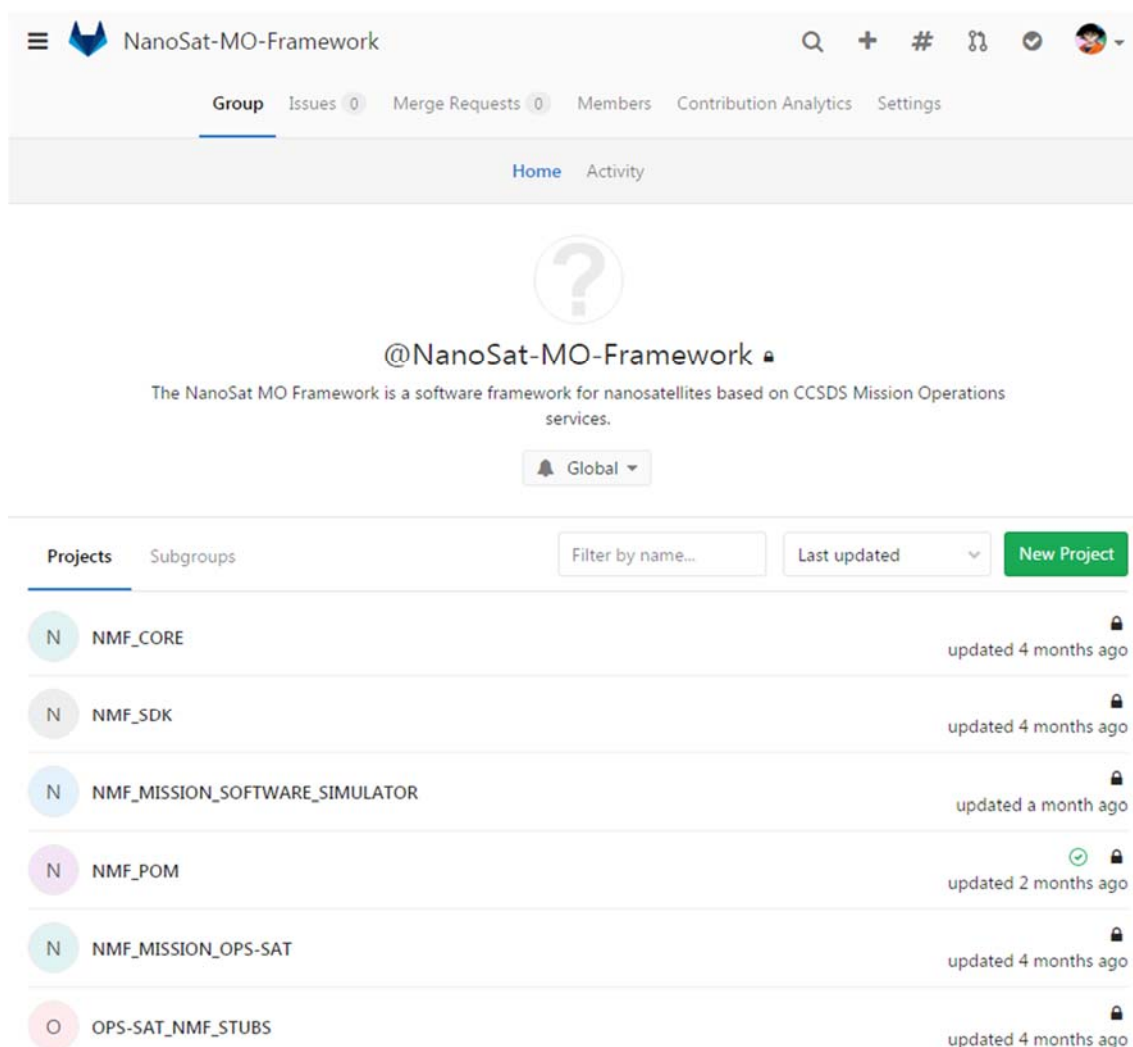


Figure 47: Web-based GitLab displaying the NMF group.

4.2.3. *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.

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.

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.

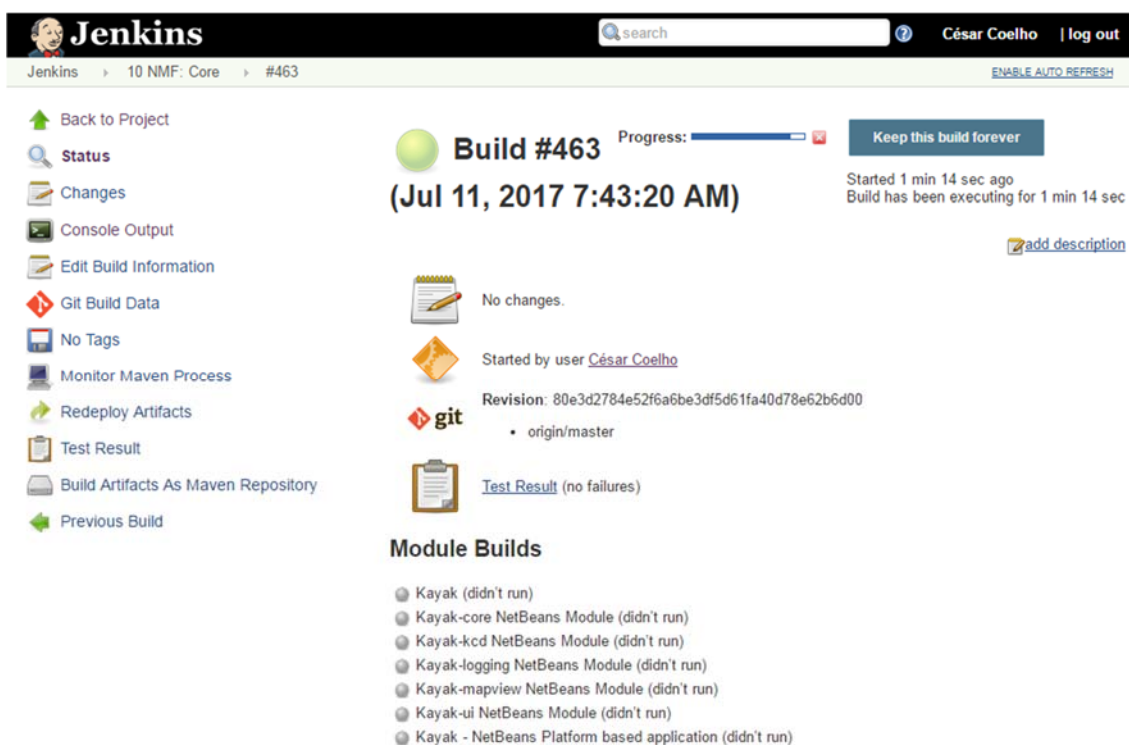


Figure 48: Jenkins during the compilation process of the NMF Core.

4.2.4. *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.

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.

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.

4.2.5. *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.

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.

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.

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.

4.2.6. *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.

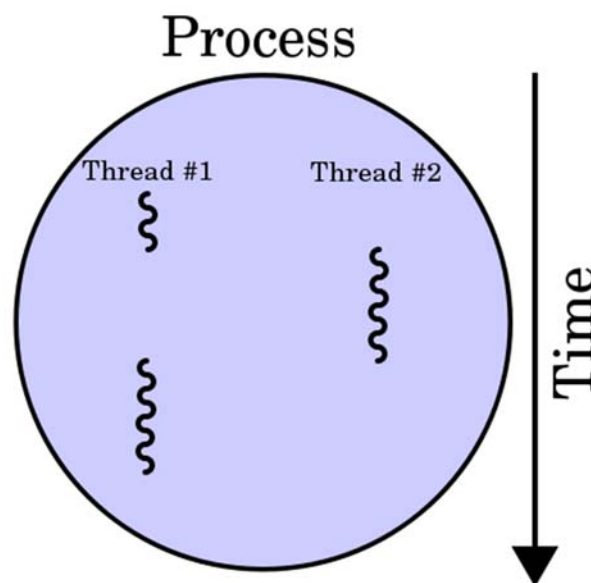


Figure 49: Conceptual visualization of multithreading: A process with two threads of execution, running on a single processor.

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.

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.

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.

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.

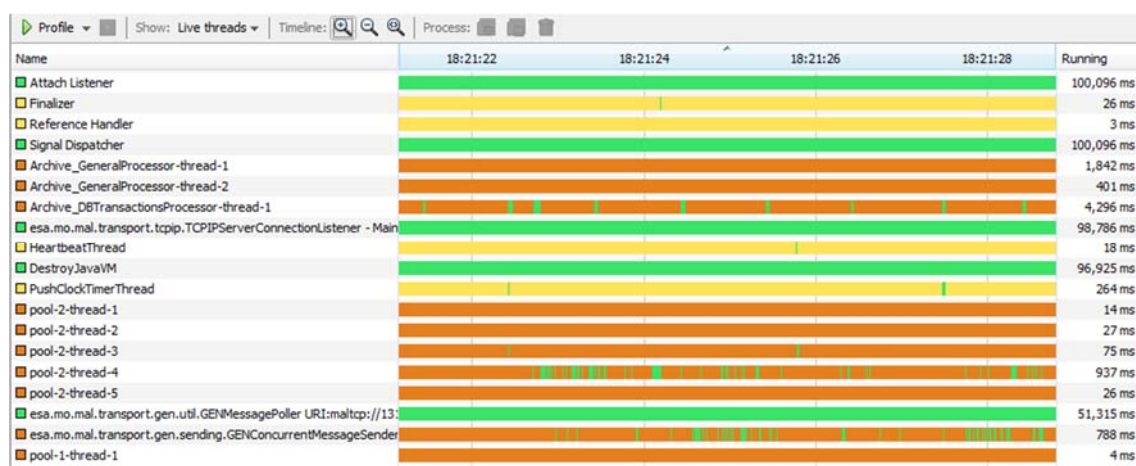


Figure 50: Netbeans profiler displaying parallel threads' running.

4.3. 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 3.1.2 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.

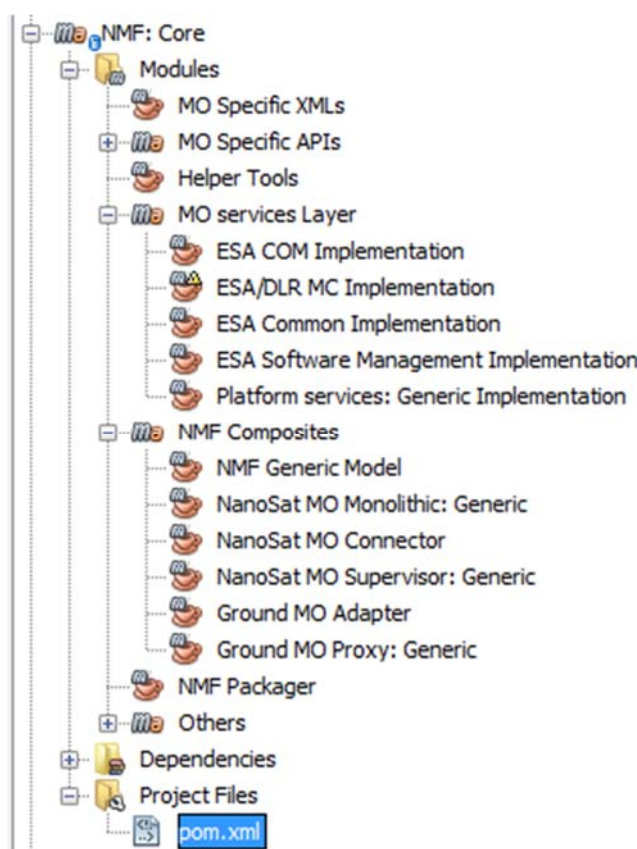


Figure 51: NMF Core “mavenized” multi-module project presented in Netbeans’ Projects window.

4.3.1. 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
- 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. 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.

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:

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.

4.3.2. 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.

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.

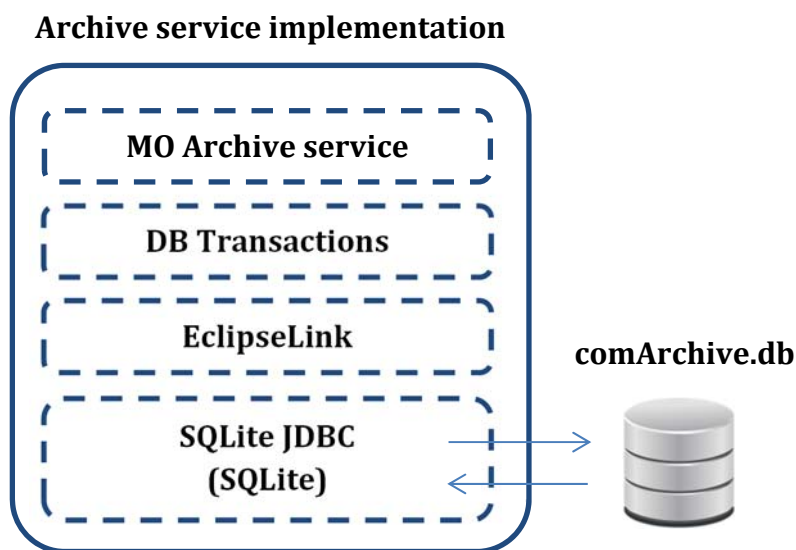
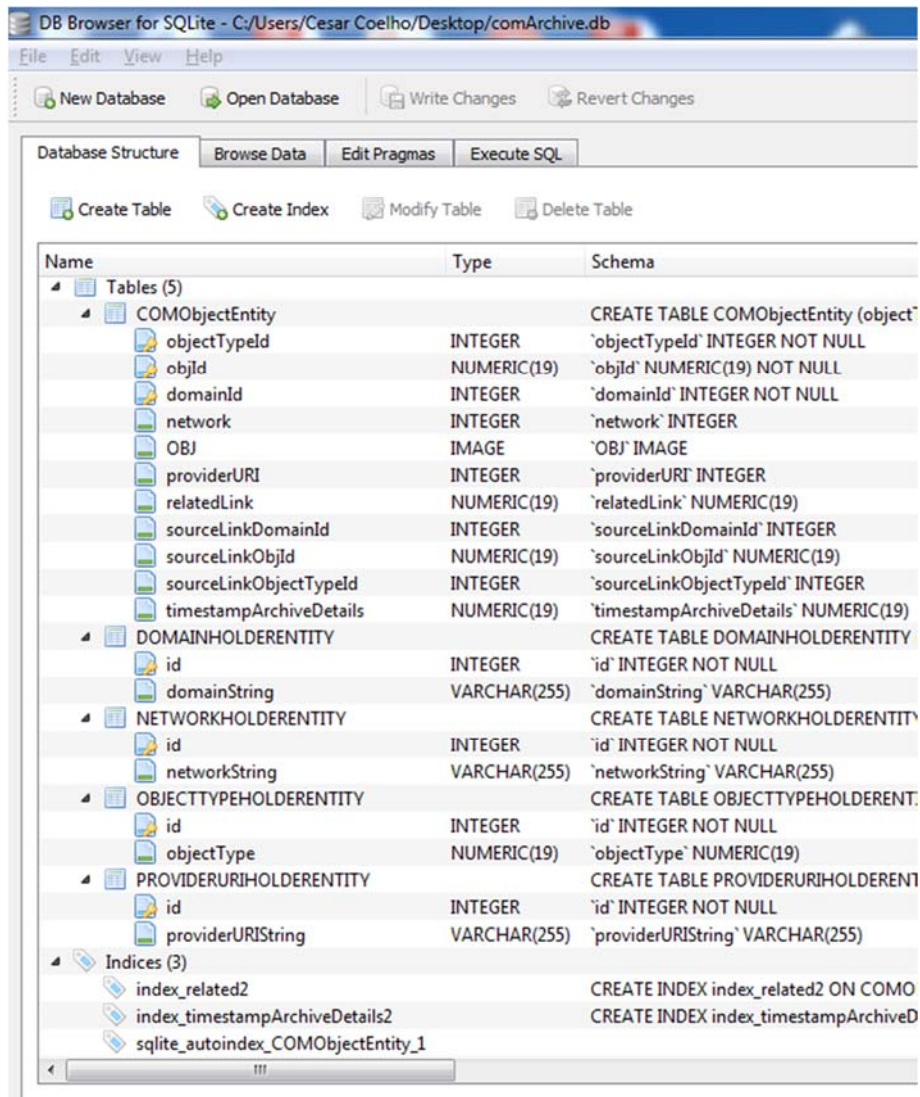


Figure 52: Archive service provider implementation diagram.

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".

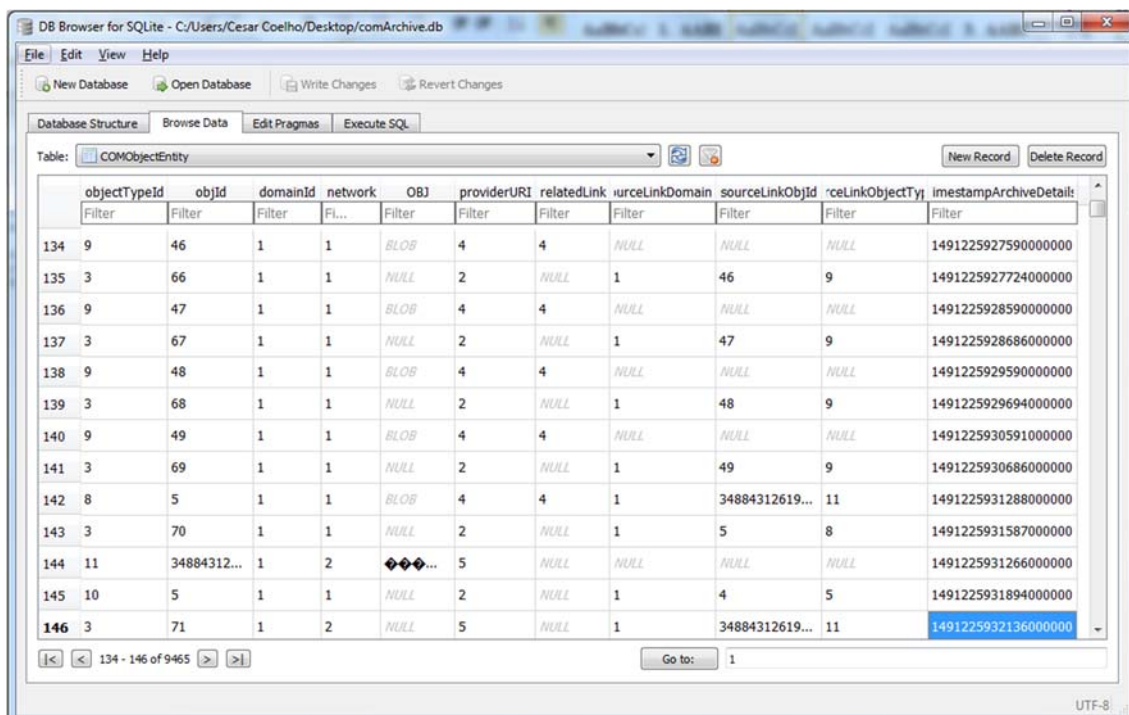
The database structure includes 5 tables and 3 indexes like Figure 53 presents.



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 NETWORKHOLDERENTITY
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.

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.



The screenshot shows the DB Browser for SQLite application. The table 'COMObjectEntity' is selected, and its contents are displayed in a grid. The grid has 12 columns: objectTypeid, objId, domainId, network, OBJ, providerURI, relatedLink, sourceLinkDomain, sourceLinkObjId, rceLinkObjType, and timestampArchiveDetail. The data is paginated, showing records 134 to 146 of 9465. The record with id 146 is highlighted in blue.

	objectTypeid	objId	domainId	network	OBJ	providerURI	relatedLink	sourceLinkDomain	sourceLinkObjId	rceLinkObjType	timestampArchiveDetail
134	9	46	1	1	BLOB	4	4	NULL	NULL	NULL	149122592759000000
135	3	66	1	1	NULL	2	NULL	1	46	9	149122592772400000
136	9	47	1	1	BLOB	4	4	NULL	NULL	NULL	149122592859000000
137	3	67	1	1	NULL	2	NULL	1	47	9	149122592868600000
138	9	48	1	1	BLOB	4	4	NULL	NULL	NULL	149122592959000000
139	3	68	1	1	NULL	2	NULL	1	48	9	149122592969400000
140	9	49	1	1	BLOB	4	4	NULL	NULL	NULL	149122593059100000
141	3	69	1	1	NULL	2	NULL	1	49	9	149122593068600000
142	8	5	1	1	BLOB	4	4	1	34884312619...	11	149122593128800000
143	3	70	1	1	NULL	2	NULL	1	5	8	149122593158700000
144	11	34884312...	1	2	◆◆◆...	5	NULL	NULL	NULL	NULL	149122593126600000
145	10	5	1	1	NULL	2	NULL	1	4	5	149122593189400000
146	3	71	1	2	NULL	5	NULL	1	34884312619...	11	149122593213600000

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

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.

4.3.3. 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.

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.

4.3.4. 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.

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.

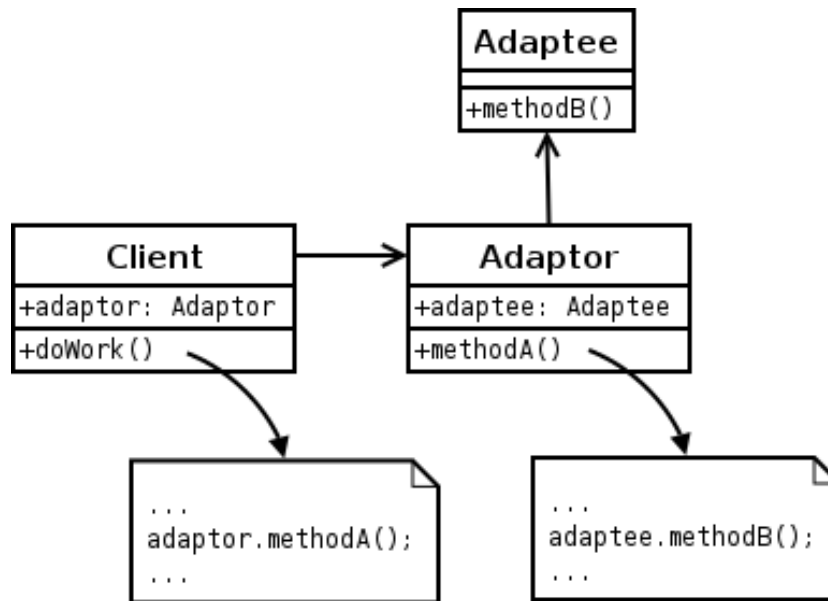


Figure 55: Adapter pattern.

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.

4.3.5. *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.

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.

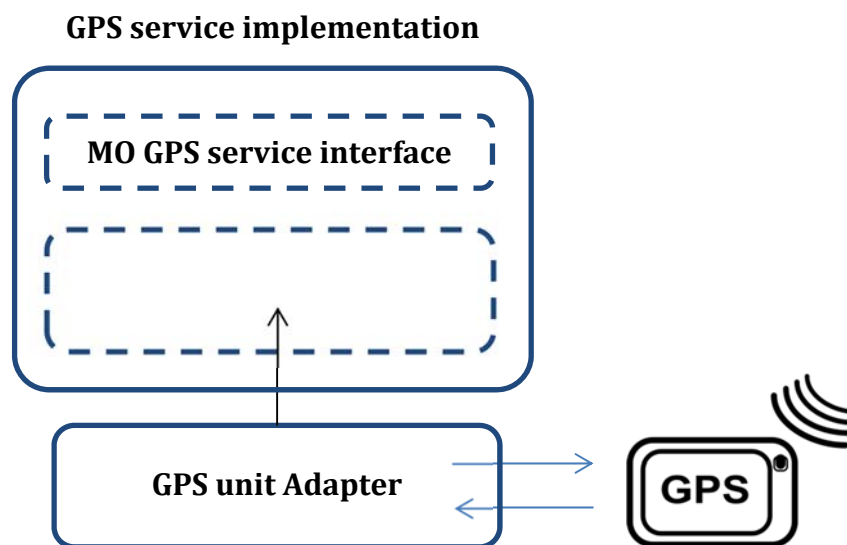


Figure 56: GPS service implementation using the Adapter pattern.

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.

4.3.6. *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.

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 3.7.3.

4.3.7. 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.

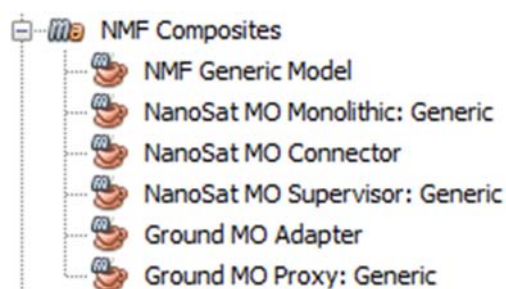


Figure 57: NMF Composites implementation.

The NMF Generic Model implementation is a project that includes all the MO services implementation packages that were presented on section 3 and also includes the MAL implementation and the TCP/IP transport binding implementation as dependencies. Figure 58 presents these dependencies.

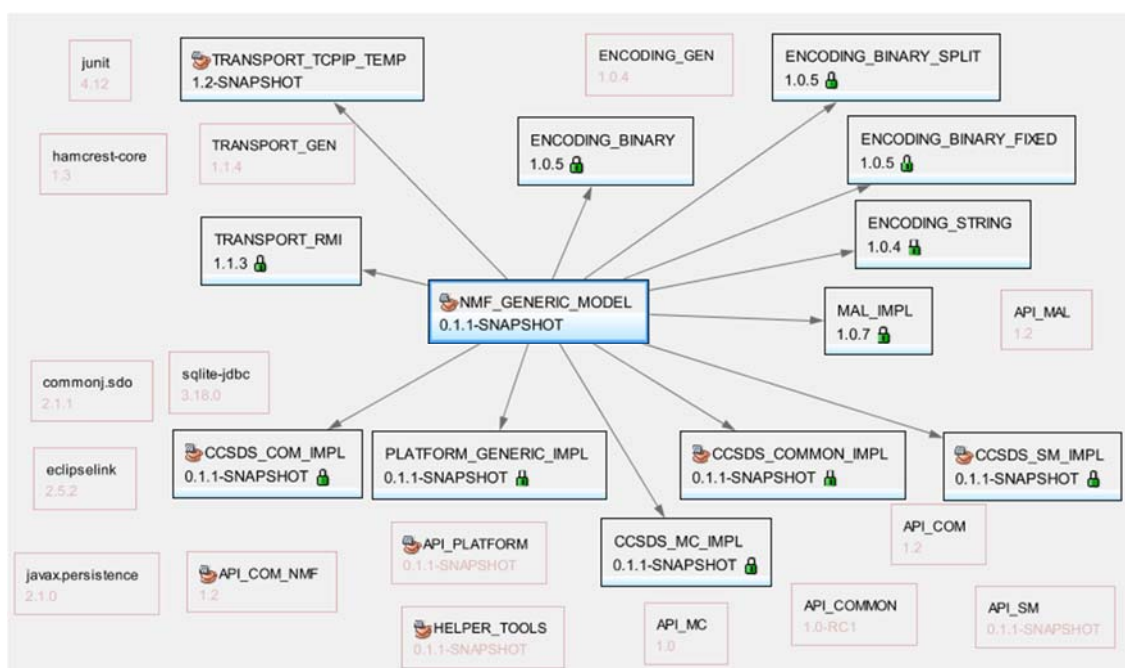


Figure 58: NMF Generic Model implementation dependencies.



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.

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.



4.3.8. *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 section 3.

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.

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.

4.4. Generic Deployment

The deployment of the NMF's Java implementation can be done by using the NMF Library concept explained in section 3.7.5. 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.

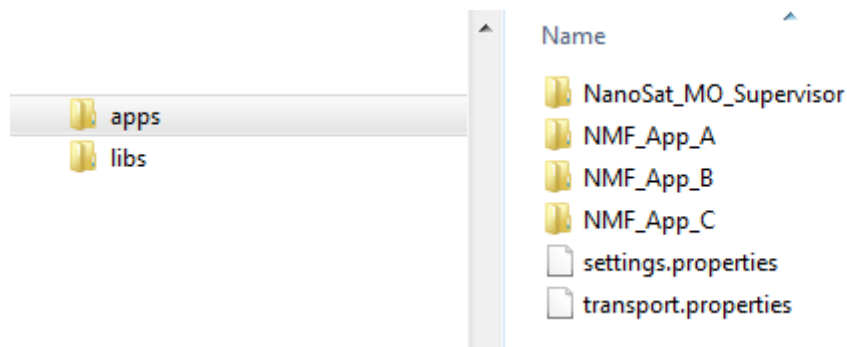


Figure 59: Example of the generic deployment folder structure.

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 SDK environment.

```

1  # NanoSat MO Framework Settings file
2
3  # To form the Network zone
4  helpertools.configurations.MissionName=OPS-SAT
5  helpertools.configurations.NetworkZone=space
6  helpertools.configurations.DeviceName=Playground
7
8  # set the name of the MAL classes to use
9  org.ccsds.moims.mo.mal.factory.class=esa.mo.mal.impl.MALContextFactoryImpl

```

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

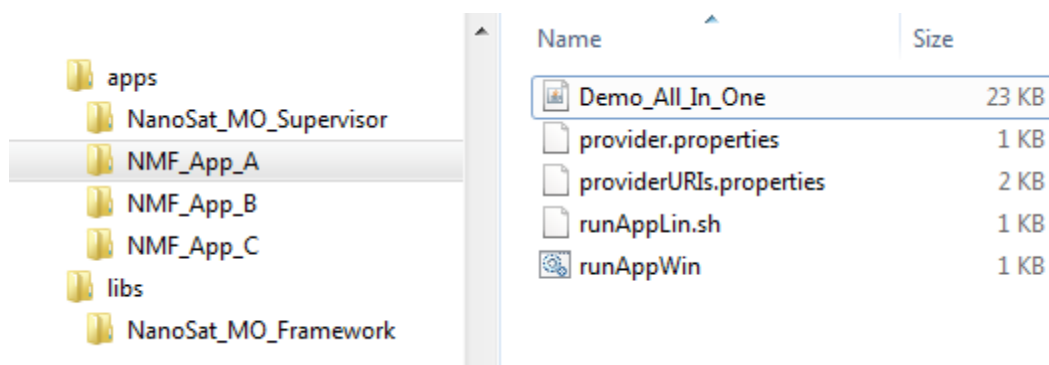


Figure 61: Generic deployment example.

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.

4.5. 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.

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 section.

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.

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.