



POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

VIPER: An Automated Tool for Flight Operations Procedures Validation

TESI DI LAUREA MAGISTRALE IN
SPACE ENGINEERING - INGEGNERIA SPAZIALE

Author: **Francesco Monari**

Student ID: 977193

Advisor: Prof.ssa Michèle Roberta Lavagna

Co-advisors: Ing. Edoardo Bruno, Ing. Biagio Cotugno

Academic Year: 2023-24

La strada delle stelle è aperta
Sergej Pavlovič Korolëv

Abstract

Advances in space technology have significantly increased the use of microsatellites by space agencies and private companies, especially for Earth Observation (EO) missions in low Earth orbit (LEO). These spacecrafts are characterised by intrinsic advantages with respect to more traditional satellites such as a relevant size reduction, a shorter development time and significantly lower costs, allowing a larger microsatellites production. The outcome necessitates an increment of automation and standardisation processes for both the satellite production line and the entire operational time.

Flight Operations Procedures (FOP) play an essential role inside this mission operation context. Such step-by-step detailed documents define the sequence of space systems in-orbit operations and they are designed to control the S/C actions under both nominal and off-nominal conditions. Although their validation and execution have usually been performed under manual control during the System Validation Test (SVT) campaign, many standardisation efforts have been made to accelerate their generation and to reduce the possibility of the human-factor in potentially major errors.

In this context, the presented Thesis aims to develop an automated tool to validate the Flight Operations Procedures produced by Argotec to directly handle and control their microsatellites from the Mission Control Centre. More in details, the tool receives the FOP in its standard procedure language and, after an initial consistency check on the procedure correctness, it structures the FOP to interface it with the Mission Control System (MCS). Once the S/C model, or its corresponding Satellite Simulator, returns the elaborated telecommands and telemetry parameters to the MCS, the tool processes the answer and continues the FOP validation, or interrupts the validation process if any error is detected. A final report containing the procedure validation status, together with the list of the possible warnings and errors, is provided to the user at the end of the process to appreciate the test results. Furthermore, the tool design describes not only a valid solution for a specific procedure, but rather a robust structure capable of validating FOPs belonging to different EO mission scenarios.

Keywords: Flight Operations Procedure, Validation, Automated Tool, SVT, FOP.

Abstract in lingua italiana

I progressi tecnologici in ambito spaziale hanno portato ad un notevole aumento dell'uso di microsattelliti nei programmi di missione delle agenzie spaziali e delle aziende private. La loro applicazione è sempre più comune ed in particolare è cresciuta sensibilmente nel contesto dell'Osservazione Terrestre in orbite basse terrestri (LEO). I vantaggi legati all'utilizzo dei microsattelliti, rispetto alle versioni più tradizionali di spacecraft, riguardano una significativa riduzione delle dimensioni, un tempo di sviluppo più breve e costi di produzione più contenuti. Tutto ciò ha innescato un aumento dei processi di automazione e di standardizzazione sia nella linea di produzione del prodotto, sia nell'ambito operativo dello stesso satellite.

Proprio nel contesto delle operazioni di missione, le Flight Operations Procedures (FOP) giocano un ruolo essenziale. Questi documenti definiscono passo dopo passo le sequenze delle operazioni dei sistemi spaziali durante la loro vita in orbita, in modo da poter controllare il satellite in condizioni nominali ed, eventualmente, di contingenza. Sebbene i test riservati a tali procedure siano stati storicamente effettuati in modo manuale durante le campagne di validazione del satellite (SVT, System Validation Test), notevoli progressi riguardanti la standardizzazione di questi processi sono stati effettuati, con l'intento di ridurre le probabilità di errori di natura umana.

All'interno di questo ambito, il lavoro di Tesi qui presentato mira allo sviluppo di uno strumento automatizzato per la validazione delle Flight Operations Procedures. In particolar modo, questo tool prevede di validare le procedure prodotte da Argotec per gestire e controllare i propri microsattelliti direttamente dal Centro di Controllo Missione aziendale. Più specificamente, un tool di questo tipo deve ricevere come input la FOP da validare nel suo linguaggio procedurale standard e, dopo un controllo preliminare sulla struttura e sui contenuti della stessa, deve interfacciarla con il Mission Control Software (MCS). Una volta ottenuti i telecomandi ed i parametri di telemetria elaborati dal MCS e provenienti direttamente dal satellite, o eventualmente dal suo simulatore, lo strumento di validazione è in grado di processare le risposte registrate e completare la validazione della FOP. In caso qualsiasi errore venga riscontrato durante l'esecuzione del test, è necessario che il processo di validazione si interrompa. Un report finale contenente lo stato di validazione

della procedura, insieme alla lista degli eventuali messaggi di incongruenza e di errore, viene infine generato e fornito all'utente in modo da poter visionare e constatare i risultati del test. Inoltre, il tool in grado di garantire tutte queste funzionalità non è limitato alla validazione di una specifica procedura per una singola missione, bensì presenta una struttura solida al punto da permettere la validazione delle FOP appartenenti a diverse missioni di Osservazione Terrestre.

Parole chiave: Procedure di Operazioni di Volo, Flight Operations Procedures, Validazione, Tool Automatizzato, SVT, FOP

Contents

Abstract	iii
Abstract in lingua italiana	v
Contents	vii
List of Acronyms and Abbreviations	xi
1 Introduction	1
1.1 Context of Work	1
1.2 Objectives of the Thesis	2
1.3 Flight Operations Procedures	3
1.4 Flight Operations Plan	5
1.5 Flight Operations Segment Architecture	6
2 State of the Art	11
2.1 FOP Validation During the SVT Campaign	11
2.1.1 Limits of FOP Validation Process	12
2.2 Automation in FOP Validation Process	12
2.2.1 Introduction to FOP Automated Validation Tools	12
2.2.2 Analysis of Existing FOP Validation Tools	15
2.3 Automated Validation Advantages	22
2.4 Driving Factors for the Design of a New FOP Automated Validation Tool .	23
3 FOP Standard Language and Structure	25
3.1 Machine-Readable Languages Trade-Off	25
3.2 PLUTO General Architecture	29
3.3 FOP Step Definition	31
3.4 PLUTO Commands	32
3.5 FOP Parameters Identification	34

3.6	PLUTO FOP Standard	34
4	VIPER Design Definition	41
4.1	Delineation of VIPER Requirements	41
4.1.1	Functional Requirements	43
4.1.2	Design Requirements	45
4.2	VIPER Architecture	47
4.2.1	VIPER Operational Modes	49
5	VIPER Implementation	53
5.1	PLUTO - FOP Preliminary Check	54
5.2	FOP Transcription in a Structured JSON File	55
5.3	MIB Consistency Check	61
5.4	VIPER - Mission Control System Interface	63
5.5	Analysis and Handling of Satellite Simulator Responses	67
5.6	Validation Report Generation	70
6	VIPER Test and Validation	75
6.1	Test Description and Passing Criteria	75
6.2	VIPER Requirements Review of Design	76
6.3	Test FOPs Preparation	78
6.3.1	TMTC Configuration Change	78
6.3.2	Operational Mode Change	80
6.3.3	House Keeping Report Request	82
6.3.4	On-Board Schedule Update	84
6.4	Test Set-Up	87
6.5	Test Results Analysis	88
7	Conclusions	97
7.1	VIPER Future Developments	98
	Bibliography	101
A	Appendix A	105
B	Appendix B	111
B.1	VIPER Test FOPs	112

B.2 VIPER Test Results	119
List of Figures	123
List of Tables	125
Acknowledgements	127

List of Acronyms and Abbreviations

APID	Application ID
ARVALIS	Automated Rule-Based Validation System
ASI	Italian Space Agency
CCL	OS/COMET Control Language
DI	Data Interface
DB	Database
ECA	Event Condition Action
ECSS	European Cooperation for Space Standardization
EGSE	Electrical Ground Support Equipment
ENG	Engineering Value
EO	Earth Observation
ESA	European Space Agency
ESOC	European Space Operation Centre
FCT	Flight Control Team
FDCA	Flight Dynamics and Collision Avoidance
FDS	Flight Dynamics System
FOP	Flight Operations Procedure
FOS	Flight Operations Segment
GS	Ground Station
GSN	Ground Station Network
ISO	International Standards Organisation
ISS	International Space Station
JPL	Jet Propulsion Laboratory
LEO	Low Earth Orbit
LEOP	Launch and Early Orbit Phase
MCC	Mission Control Centre
MCS	Mission Control Software
MIB	Mission Information Base
MP	Mission Planning

MQTT	Message Queuing Telemetry Transfer
NASA	National Aeronautics and Space Administration
OBC	On-Board Computer
OPMODE	Operational Mode
OPSVAL	Model Base Validation System
OSW	On-board Software
OTS	Off-The-Shelf
PA	Procedure Automation
PDAS	Payload Data Archive System
PI	Packet Identification
PL	Payload
PLEXIL	Plan Execution Interchange Language
PLUTO	Procedure Language for Users in Test and Operations
PRL	Procedure Representation Language
PTV	Pre Transmission Verification
PUS	Packet Utilization Service
RAW	Raw Value
ROD	Review Of Design
RX	Reception Radio Configuration
S/C	Spacecraft
SM	Spacecraft Management
SOM	Spacecraft Operations Manager
SPACON	Spacecraft Controller
SPELL	Satellite Procedure Execution Language and Library
SSM	Space System Model
STOL	Spacecraft Test and Operations Language
SVT	System Validation Test
TC	Telecommand
TCP	Transmission Control Protocol
TM	Telemetry
TX_RX	Transmission and Reception Radio Configuration
UUID	Universal Unique Identifier
VIPER	Validation Instrument for Procedure Evaluation and Review
VML	Virtual Machine Language
VSC	Visual Studio Code

1 | Introduction

1.1. Context of Work

Since the beginning of the satellite era, space missions dedicated to Earth Observation (EO) have been numerous and of great relevance. These missions have provided us with precious information about our planet and its environment throughout the last 60 years. Currently, satellites devoted to EO effectively monitor changes in climate, track natural disasters, and collect data about the Earth's surface, atmosphere, and oceans. The European Space Agency (ESA) is widely recognised for its expertise in studying the terrestrial ecosystem and the IRIDE constellation represents one of its latest contributions to the EO field. [13]



Figure 1.1: IRIDE: European Earth Observation satellite constellation

This ESA program promoted by the Italian Space Agency (ASI) and the Italian Government is a system of Low Earth Orbit (LEO) spacecrafts (S/C), considered in fact as a constellation of constellations due to the different nature of the satellites instruments and data acquisition technologies. The specific purpose of the mission is to provide an end-to-end system to foresee, withstand and study environmental events such as fires and hydrogeological disruptions over Italy, rather than monitoring critical infrastructures, the

air quality, the weather conditions and providing global coverage of the Italian continental territories and coastlines. [14] Space programs like IRIDE owe their success to the proper design of space operations as they encompass the necessary plannings and executions to maintain the spacecrafts functionalities and to achieve the mission objectives.

Moreover, these actions must be structured and organised to account for the short duration of LEO mission visibility windows, typically lasting a maximum of 10 minutes, during which telecommands (TC) and S/C telemetries are exchanged with the satellites. To effectively regulate the mission operations, it is necessary to edit specific procedures identified as Flight Operations Procedures (FOP). As part of the mission design, FOPs must undergo a test and validation campaign to correctly perform a preliminary analysis of their contents and implement them within the Mission Control System to demonstrate their effectiveness and correctness.

The work reported in this Thesis is thus focused on the development of a new tool for the automated validation of FOPs, to investigate if this kind of instrument can effectively represent a real alternative to the manually performed FOP validation process in EO missions context. The aforementioned is known as VIPER (Validation Instrument for Procedure Evaluation and Review) and its design was carried out during an internship program at Argotec S.r.L., one of the space companies to which ESA, ASI and the Italian Government have commissioned the production of a substantial part of the entire IRIDE satellite fleet.

1.2. Objectives of the Thesis

Going deeper into the objectives that the Thesis aims to fulfil, they have been identified as follows:

- Carry out a preliminary in-depth study of FOPs, their context of application during the operations of a space mission and Argotec Flight Operations Segment architecture, as it is intended to actively integrate VIPER within this operational framework.
- Conducting an analysis to verify the current state of automation in the field of procedure processing and automated validation tools that have already been developed, in order to exploit the advantages observed and, possibly, improve the bottlenecks reported by previous studies on these topics.
- Develop a new standard for writing procedures in a machine-readable language, thus enabling the interpretation of a FOP by a purely automated tool. The creation of

this new standard will bring a major update to the current way in which FOPs are produced and written in Argotec.

- Define all the necessary requirements for the development of VIPER, starting with a detailed analysis of the actual needs of the Flight Control Team, that must leads to the identification of mandatory functional and design requirements.
- Actually write the source code of the tool, by producing all the relevant Python scripts and functions requested for its execution. More specifically, defining its operating principles, its interfaces, and the generation of a sufficiently clear and compact output to allow an external operator to appreciate the result of the validation conducted. The latter involves processing the FOP with the Mission Control Software and the Satellite Simulator already developed by the company.
- Design and conduct an effective validation test campaign, in order to demonstrate the proper functioning of the tool and compatibility with the other systems within the Flight Operation Segment.

1.3. Flight Operations Procedures

Flight Operations Procedures are specific documents that shall contain all the planned activities to be carried out along the mission timeline in order to ensure both the safe satellite operational life conditions and the mission objectives accomplishment. In their development environment, procedures rely on the knowledge and heritage of previous space missions, along with detail studies on the new specific mission scenario. These step-by-step documents outline the operational sequences of space systems, detailing the telemetry (TM) requests and telecommands (TC) involved.

TM data provides fundamental information concerning the real-time spacecraft status and performances, including parameters such as the satellite system health and operational modes. TMs are transmitted by the spacecraft itself to the Mission Control Centre (MCC) through radio frequencies in different bands depending on the data necessities: these are collected by the Ground Stations (GS) included in the ground segment and transmitted in turn to the Mission Control System. [26] In order to achieve the most efficient transmission possible, TM data are collected in packets: each single unit of them contains several TM parameters within it, organised according to mission specifications and control requirements. It is therefore possible to request the download of a single parameter or, alternatively of the whole TM packet. The so called “Housekeeping TM” is a case of TM packet produced and transmitted automatically to ground in a continuous

flow to properly monitor the S/C health, while specific subsystem parameters or packets shall be requested, mostly on need, from ground control.



Figure 1.2: NASA Johnson Space Center's Mission Control Centre

The process of sending instructions to control the spacecraft operations including its trajectory, attitude manoeuvres, payload activation and data acquisition is instead allowed by the telecommands. Their transmission to the satellite follows the opposite path with respect to the TM downlink, being generated in the MCC, transmitted to the GS and then send to the spacecraft. TCs can be operated during real-time routines if the S/C tracking passes allow immediate executions, or they can be organised in stacks to be sent to the On-board Computer (OBC) and then executed with a time-tag logic. The execution of time-tagged TC is commonly indicated as “sequencing”. [16]

The FOP can be categorised into nominal and contingency ones: more in details, the nominal procedures are structured with a specific sequence of actions to be followed compatible with the nominal timeline characteristic events. In contrast, contingency procedures are designed to be followed during off-nominal conditions, to properly recover from spacecraft and ground segment related anomalies or, at least, mitigate the possible negative effects. An anomaly can be registered when the TM measurements report different parameters values from the nominal ones or even when an expected action is not fully completed due to unknown malfunctioning. A typical FOP structure foresees a definition of the procedure purpose and on its input and constraints (if present), followed by a first check on the essential preconditions to be verified and by the real essence of the document, namely the actions description to reach the required output. In the latter, all the necessary TCs

and TMs shall be inserted. A procedure may include specific commands to recall the execution of other different procedures when the nominal condition is no longer verified and the need to follow a contingency procedure arises. Moreover, the language adopted shall be clear and unambiguous, in order to minimise the probability of committing mistakes during the execution.

This kind of documents do not only involve the transmission and reception of TCs and TMs, but they can be exploited to automate interactions with ground segment systems too. The application of this concept could lead, for example, to automatise the S/C data collection, archiving and distribution in order to make them more easily available during the procedure's elaboration, rather than generate specific TM reports automatically from the reception of raw TM parameters from the satellite. Before being inserted in the Flight Operations Plan, procedures must undergo a revision and validation process entrusted to the Flight Control Team (FCT) members under the responsibility of the Spacecraft Operations Manager (SOM): FOPs are in fact edited by the FCT together with the members of different Units involved during the satellite design and assembly. This process is intended to include those who are informed of all technical specifications and potential vulnerabilities of the spacecraft at a subsystem level, obtaining in the meantime optimised procedures for their applications during routine operations.

1.4. Flight Operations Plan

The Flight Operations Plan identifies the list of all the nominal and contingency procedures defined by the Flight Control Team that will be adopted during the mission. As stated before, each FOP included in the Plan shall be validated during the System Validation and Test (SVT) campaign.

The need to support the verification of procedures is necessary to ensure safe and reliable flight operations throughout the entire mission, such as to grant the compatibility of them with the current satellite hardware and software, but also with the actual Mission Control Centre configurations: once this passage is completed, they will be eventually published in the Flight Operation Plan. [29]

Moreover, in case of constellation missions, the Flight Operations Plan is designed to contain different instances of procedures for each spacecraft, as certain technical specifications may not be common to the entire fleet. Since FOPs constitute the building blocks of all the command schedules and timelines, all the mission phases operations shall be covered by the Flight Operations Plan procedures, which typically include the following passages:

- **Launch and Early Orbit Phase (LEOP)**, the first phase that comprehends the launch, the separation of the S/C from the launcher, the boot-up and detumbling subphase, the deployment of equipment and, eventually, the initial acquisition of TMs.
- **Commissioning Phase**, namely from the first TM provision that confirms the health status of the S/C until the in-orbit operational configuration is reached.
- **Routine Phase**, when the nominal operations activities are performed: usually this phase concerns the exploitation of the mission objective(s).
- **End of Life Phase**, including a last check on the satellite status, a TC transmission with the necessary actions to safely terminate the S/C operational life and, finally, the passivation of S/C subsystems.

1.5. Flight Operations Segment Architecture

FOPs are a crucial figure of merit in mission operations and must be designed to be compatible with the Flight Operations Segment (FOS). This architecture is designed to support a satellite, or even an entire constellation, from the LEOP phase to the end of the mission. To maintain proper communication with the S/C, the FOS involves the coordination of Ground Stations and the reception, processing and dissemination of data to derive the most relevant mission insights. The FOS framework comprehends activity of procedures integration, payload operations, satellites command and control: post-mission activities are carried out by the FOS too.

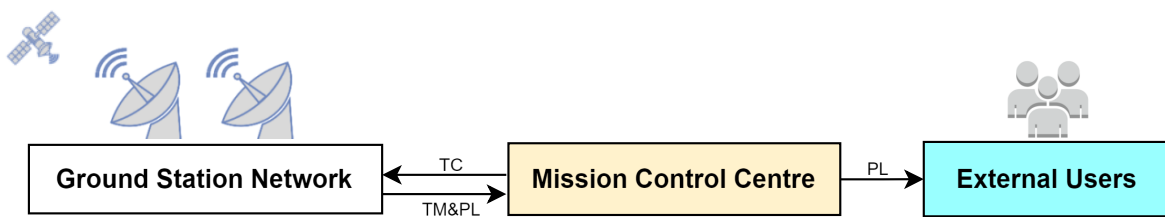


Figure 1.3: High Level FOS Architecture

As it is possible to appreciate in Figure 1.3, a typical FOS structure must comprehend the following basic components:

- **Ground Stations Network (GSN)**, which includes the GS segment spread around the world to assure the optimal link with the S/C, necessary to transmit and receive TCs and TMs.

- **Mission Control Centre (MCC)**, connected to the GSN and responsible for all the control and monitoring functions (S/C housekeeping, handling of the mission phases, mission planning, flight dynamics, collision avoidance, storage of TM and PL data, etc.).

Strictly speaking the external users are not part of the FOS, but they are connected to specific interfaces with the MCC to retrieve the requested information, typically the payload data. Although the architecture of a ground segment may seem simple at a high level, it is actually quite complex when considering the detailed functions, tools, and interfaces involved. Starting from this general point of view, it is possible to deepen the more detailed FOS architecture developed by Argotec in the context of an EO mission. This particular investigation is necessary to study the FOS main characteristics and fully understand the FOP validation process logic: moreover, analysing the Argotec proposal is crucial because it represents the environment in which the FOP validation tool will be developed and tested. A FOS schematic view is shown in Figure 1.4 where the same fundamental blocks of Figure 1.3 are present, with a MCC deconstructed representation. The FOS principle of working can be expressed through an explanation of the single architecture components:

- **GSN**: The network already described has separate channels for the TC provision sent by the MCC, for telemetries relative to the S/C housekeeping status and for the payload data.
- **Satellite Simulator**: During the SVT campaign that the FOS undergoes, it is not possible to communicate with the satellite through the GSN. To reproduce the S/C responses, the MCC is connected to a FlatSat. This particular 1:1 representation of the satellite allows the FOP and FOS validations at both hardware and software level in a relevant mission operations environment. It characterises the specific satellite or mission, so that the monitoring & control software tools can be used to interpret the telemetry received from a satellite and check values are within limits. Moreover, it is possible to generate the commands expected by the satellite and check their execution progress.
- **MCS**: The functionalities of the Mission Control System software are designed to allow the processing and visualisation of telemetry and telecommands. The MCS is linked to the GSN on one side, while the spacecraft controllers have a direct access to this system through their workstation on the other side. The software also includes the Mission Information Base (MIB), a satellite database containing the layout and monitoring and control rules for satellite telemetry and telecommands.

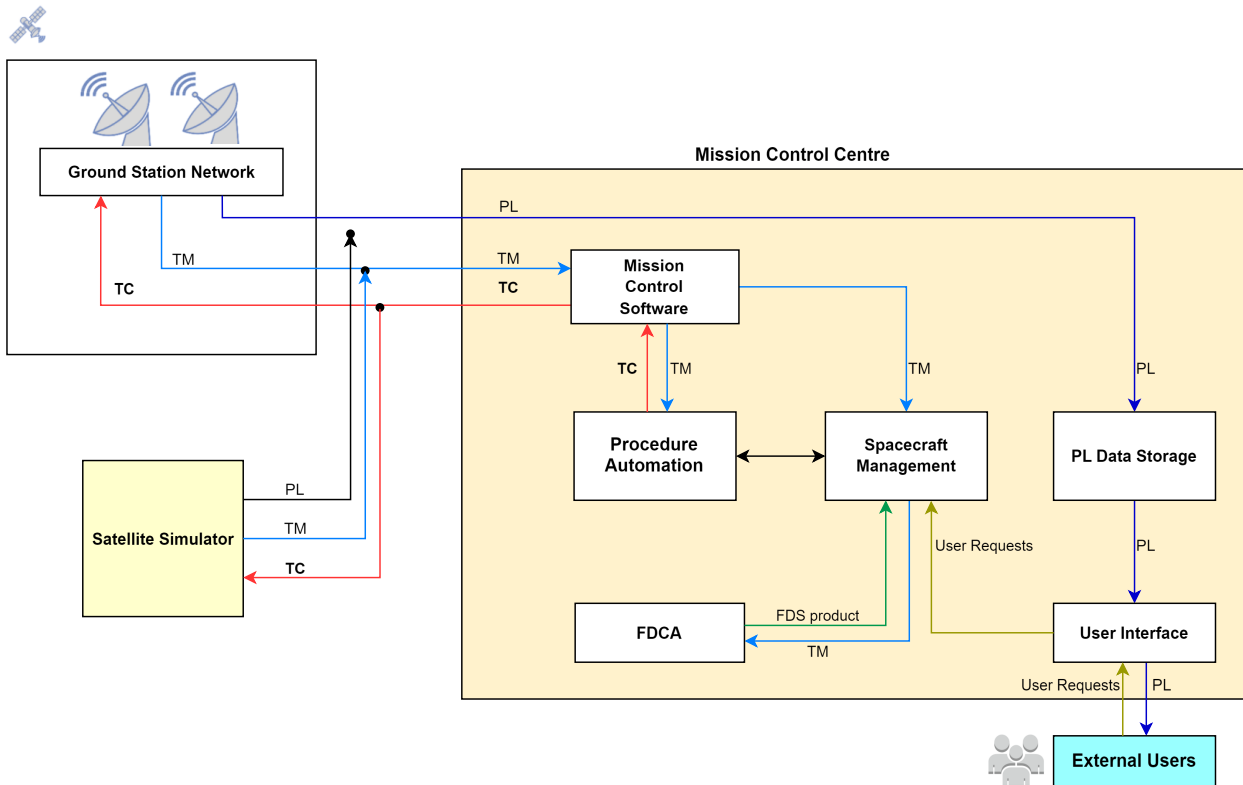


Figure 1.4: FOS Architecture Comprehending a Satellite Simulator and a Deconstructed MCC

These files effectively characterise the specific satellite or mission. The MCS TC framework is capable of generating the commands, release them and verify their on-board reception and, eventually, their execution status. To accurately interpret the received telemetry and generate commands to achieve mission objectives, the MCS is used to establish a link between the Mission Database and the FOS operator. Deepening on the TM aspect, the MCS receives the packet data from the GSN, translates the results into engineering notation and checks for consistency with the integrated TC and TM database: finally, it processes and displays them. This type of software application is able to calibrate the TCs and check the TMs limits of the received packet by implementing the correct verification criteria. The monitoring & control software tools can be used to interpret the received telemetry and generate the appropriate expected commands. A similar software architecture and database is present on board the S/C to interpret TCs and generate TMs in accordance with the data processed on ground.

- **SM:** The Spacecraft Management comprehends a Telemetry Viewer Interface to easily interpret the received data and the Mission Planning functions (MP). The

latter is responsible for the import of orbital events, GS passes and requests coming from the external users regarding specific desired S/C operations. Once elaborated all these possible requirements and constraints, the SM shall produce the mission timeline, namely a schedule of activities for both the ground operations and their uplink to the satellite.

- **FD/CA:** The Flight Dynamics and Collision Avoidance section is in charge of all tasks related to orbit determination, propagation and manoeuvres computation: specific manoeuvres may result necessary in the event of a possible risk for in orbit collision. After the elaboration of the TM parameters coming from the satellite, the FDS products are directly delivered to the SM block for the timeline creation.
- **Procedure Automation:** The FOS Procedure Automation component involves three main elements: the procedure preparation block, the Flight Operations Plan, and the FOP schedule execution block. Nominal and/or contingency procedures are directly written in the preparation block, registered in the PA and, once validated, incorporated into the FOP plan. Considering the telemetry from the MCS and the SM timeline requests as inputs, one or more FOPs belonging to the plan can be operated in the schedule automation programme either in real time or through a time-tagged stack. This execution process can be started from the PA user interface accessible to the Flight Control Team. The schedule automation capability is an asset for handling the repeated LEO missions passes over the GS.
- **PDAS:** The Payload Data Archive System store all the PL data in a dedicated database following a structured logic. This unit is able to properly communicate with the DI and the external users respectively, namely the major payload customers.
- **User Interface:** This interface is applied to transfer in a secure way the payload data over a network, to make them accessible to the users. This transfer protocol offers a high level of security, encrypting the data and regulating the access via authentication credentials and secure ports.

2 | State of the Art

2.1. FOP Validation During the SVT Campaign

As previously stated, the System Validation Test campaign is a mandatory step during the design of a space mission, focused on verifying the performance of the spacecraft by conducting systems integration and performing capability tests in a relevant testing environment. In the context of the Flight Operations, this campaign interests the entire FOS architecture. This method of verification takes place during the qualification phase and implies testing the software and tools directly with the satellite flight model. Although preliminary and earlier testing processes that exploit the Satellite Simulator are strongly encouraged, only the SVT campaign can validate a Flight Operations Procedure.

To determine the appropriate validation process for a FOP, it is essential to specify all the actions required to test a procedure. Both a high-level general analysis and a detailed check of each individual action shall be performed to verify the entire logical sequence of TCs and TMs. Before focusing more on the single step actions consequences, which must not conflict with other telecommands and telemetry requests, it is necessary to check that this logical sequence is correctly reported in the FOP. Later on, through the connection with the Mission Database, it shall be verified that all the TCs and TMs cited by the FOP have a real correspondence with the ones effectively inserted in the DB. The procedure shall be then integrated within the MCS to provide the necessary requests and data provisions elaboration. [21] To replicate the spacecraft behaviour in response to the commands, a Satellite Simulator must be integrated and directly connected to the MCS: the analysis of the telemetry coming from the FlatSat and the relevant TC sent during the simulation run, together with the aforementioned equipment set-up, ensure the FOP validation in a relevant testing environment. During the SVT campaign, the software must be instead connected to the satellite flight model, thus validating the operational procedures with the highest degree of fidelity.

The described Test method appears to be the most suitable verification process for a FOP validation, because it allows the application of FOS dedicated equipment, instrumentation

and software. This process can demonstrate if the mission FOPs are correctly written, consistent with the stated objectives and if their nominal telemetry parameters are guaranteed. Once these three aspects are verified, a procedure can be considered validated and ready for operational use.

2.1.1. Limits of FOP Validation Process

However, the described procedure validation method presents some drawbacks. First of all, the process is considered quite time consuming because of the large number of verifications that are still performed manually, in conjunction with any further problem solving activities. Moreover, the FOP testing campaign priority level is not univocally defined by a standard reference, and it could be identified as a low priority task during the satellite design and development process. The major consequence of this classification is that not every procedure will be effectively validated in its entirety. [20]

The same result could also arise due to budget limitations for the test campaign and delays accumulated during the project phase. [1] Another constraint of this SVT validation campaign is the limited availability of the flight model for procedure testing. This, combined with the significant amount of manual workload required, results in a narrow FOP validation scope. [22]

2.2. Automation in FOP Validation Process

Argotec had followed the same procedure validation method as described in section 2.1 in the context of past space missions SVT campaigns, relying on proven testing processes to obtain the intended results. Nevertheless, the company standardisation and automation trend are evolving in this field too. The aim of automating the FOP validation with a new dedicated tool is to minimise the drawbacks highlighted in subsection 2.1.1. This choice leads to conduct a detailed analysis on the current state of the art of procedures automated validation systems. Then, it will be possible to understand whether it is feasible to automate this FOP testing process without losing, during the development of a new tool, the specific characteristics, the past experiences and the lessons learnt by Argotec in procedures validation methods.

2.2.1. Introduction to FOP Automated Validation Tools

According to the automation strategies and techniques for flight operations reported in [5], the implementation of these methods inside the design of a space mission corresponds

to the insertion of a “controlled operation of a process by computer software that replace human labour”. This intention is not merely a theoretical aspiration, and it is applicable to the FOP validation context with the potential to reduce the workload of operators by simplifying the process. To illustrate the effectiveness of this concept, a schematic overview of a typical validation method is proposed for both the manual and automated cases.

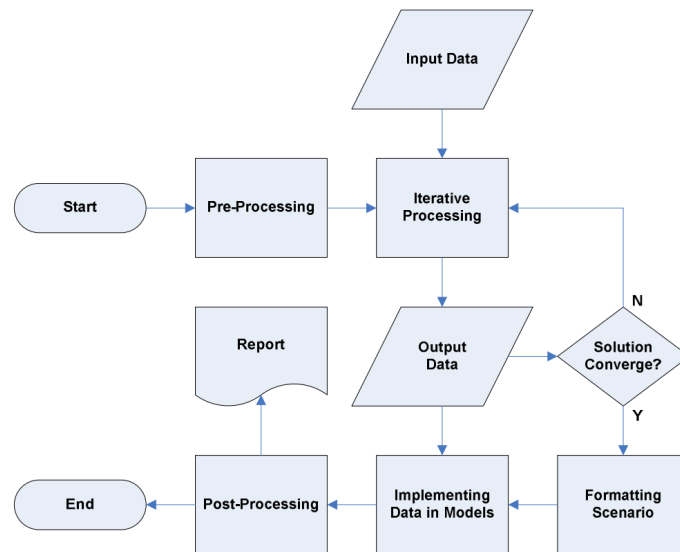


Figure 2.1: FOP Validation method for a manual case [5]

Figure 2.1 illustrates the necessity of a preliminary "pre-processing" stage at the outset of a FOP testing procedure. This stage is focused on ensuring the consistency of the procedure, particularly in terms of its contents. Once the anticipated crosschecks have been performed and the required input data has been provided, the FOP elaboration can begin. Logically, this will lead to the production of output data to analyse and verify with the available models and other FOS tools indications: the results will be post-processed and a final validation report will be produced before the procedure testing loop is fully completed by the operator. With regards to the automated tool case illustrated in Figure 2.2, it is straightforward that the actions required by the FCT members are limited to the production of input data and the initiation of the automated process. All the manual passages mentioned in the first case are now under the control of a tool, which is capable of independently executing them and producing the final validation report. It has been demonstrated that the implementation of automated processes can be a strategic approach, as it does indeed simplify the validation loop. However, in order to verify their validity at a lower level, it is necessary to gain a deeper understanding of the rationale behind these automated processes.

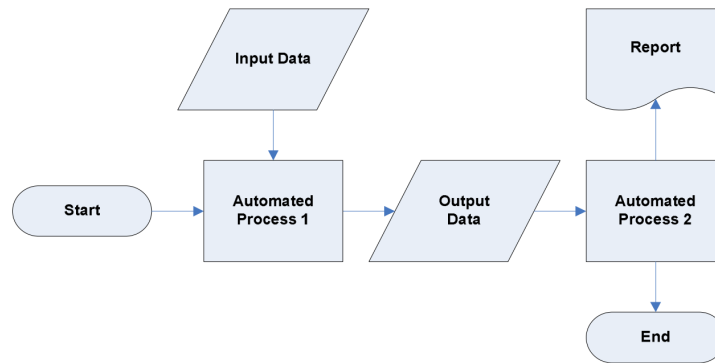


Figure 2.2: FOP validation method for an automated case [5]

A valuable example to study is the Model Based Operation Validation system (OPSVAL) developed by VEGA Space. [20]

This concept of automated validation tool for FOPs has been proposed as a means of reducing the need for human involvement during the verification stages of the procedure. OPSVAL is designed to iterate the FOP with the S/C database and satellite data, as illustrated in Figure 2.3.

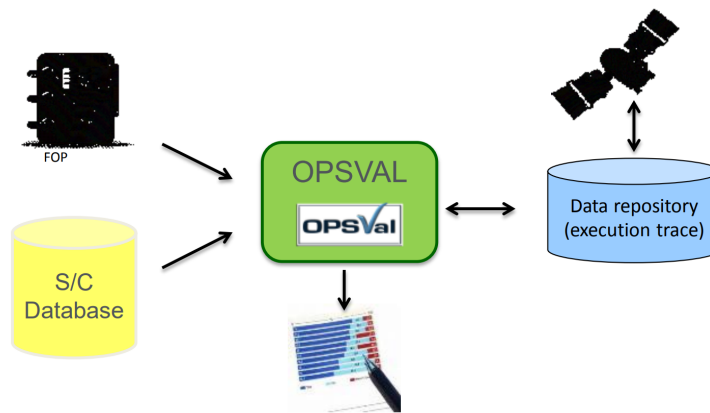


Figure 2.3: OPSVAL Objectives and context of work [20]

The high-level concept of the tool's functioning principle is shown on the left side of Figure 2.4. To provide a case study as comprehensive as possible, another example of procedure evaluation design integrated with automation is reported from [5] in the same figure. The validation process for both conceptual maps appears to be clearly divisible into sub-processes, with the output of one activity serving as the input for the next one. This approach allows for the combination of different activities and their connection with external items such as databases, software, and other characteristic FOS tools. The level of automation applied differs between the two cases: the OPSVAL functioning logic does

not require a human intervention during the validation process, whereas the design of [5] foresees the selection of the most appropriate procedure model to follow.

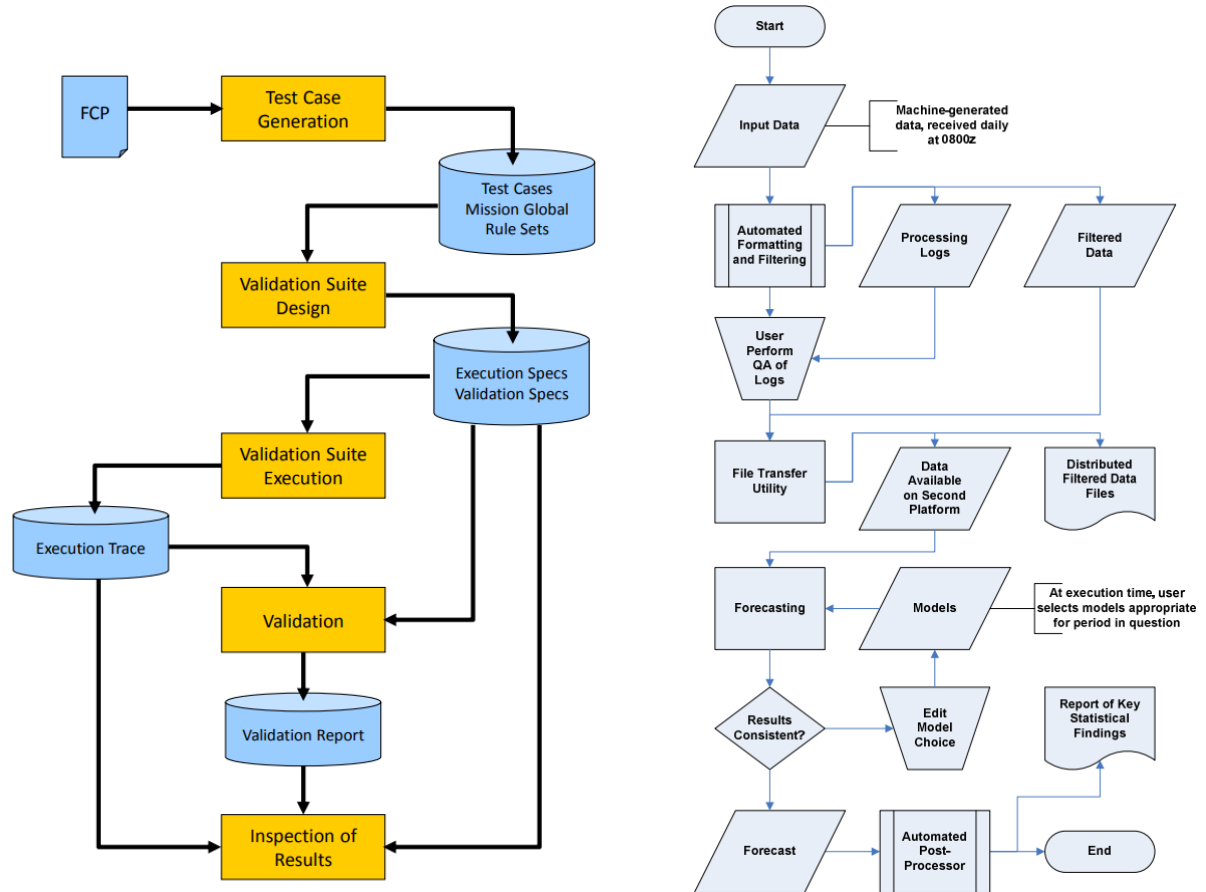


Figure 2.4: Comparison between the OPSVAL Conceptual Map [20] and the procedure evaluation process map of [5]

The selection of these options appears to be the consequence of considerations and requirements that are determined by the specific needs of each project: consequently, it is not possible to assert that one solution is inherently superior to the other. However, this permits a certain degree of flexibility in the utilisation of automation, according to the specific requirements and objectives of the user. In both cases the implementation of automation represents an initial overload of the FCT work, but a sensible reduction of it can be granted during the FOP validation campaigns. [20] [5]

2.2.2. Analysis of Existing FOP Validation Tools

A further analysis at a lower level of detail on the design of automated FOP validation tools is proposed hereafter. The OPSVAL tool, already presented in the previous paragraph, is mainly based on the collection of data produced during the FOP elaboration,

verified according to the procedure characteristics nominal conditions. The process can be subdivided into three primary environments: the test case generation, the validation execution and the inspection of results. [20] As an automated tool, the procedure to validate is required in a machine-readable language regulated by a standard form to allow its evaluation and the generation of the test cases rules. These rules are the well-defined cases to be followed during the procedure validation: they may differ between FOPs because they define and correlate the TC/TM sequences with the necessary checks based on the procedure contents. As previously stated, a FOP must be tested at two levels of correctness to ensure that the information reported in the procedure is consistent with the Mission Database and that the sequences are correctly uplinked and executed. [31] This process is illustrated in Figure 2.5, which shows the specification and definition of validation rules directly derived from the procedure.

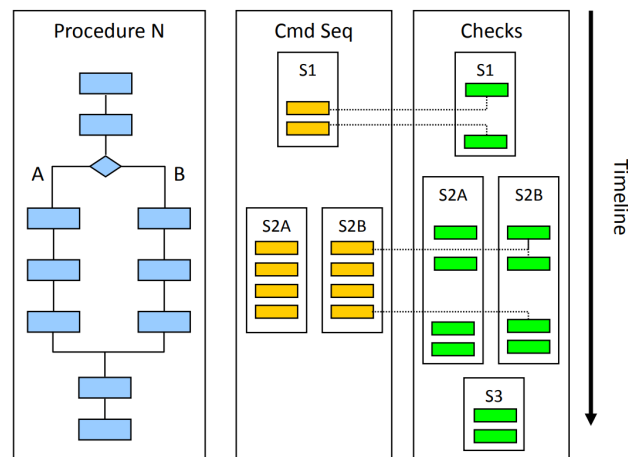


Figure 2.5: Example of OPSVAL test cases definition [20]

It is imperative that the command sequences identified are fully verified and respected before proceeding with the FOP evaluation, unless specific indications are imposed directly from the operator. Once the preliminary checks have been completed, all the procedure contents are implemented in the definition of an execution process to follow in order to complete the validation. As can be appreciated in Figure 2.6, all the TC/TM sequences are arranged in a chronological order to ensure the FOP execution in its relevant test environment. The validation specification represents the set of rules created during the test case editing process, organised according to the execution trace to monitor the outcomes of the procedure elaboration.

The execution environment includes these specifications, the MCS and the Satellite Simulator itself, along with their appropriate interfaces, remembering that the link with the

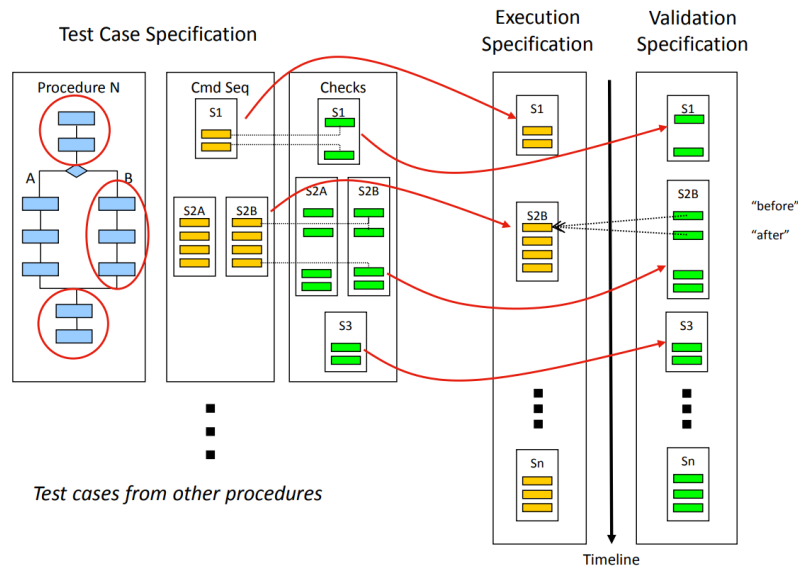


Figure 2.6: OPSVAL Execution and Validation specifications [20]

flight model can be exploited to connect the real S/C during the SVT campaign. All the data produced during the procedure execution are stored in appropriate data archives to assist the offline evaluation of the test results. The term “offline validation” refers to the process of verifying that the stored data is in accordance with the validation specification once the procedure elaboration process is complete. The objective is to ensure that the process is not dependent anymore on the use of the simulator, with regards to the analysed FOP. [20] Finally, a validation report is created to organise the procedure verification outcomes in a human-readable format that is clear and unambiguous. It is crucial to acknowledge that the OPSVAL system does not consider the execution system itself to be a component of the tool. Instead, the validation instrument relies on external procedure elaboration, provided that it is consistent with the validation specification.

The model implemented in this tool represents one of the most important figures of merit in the automated FOP validation field, due to its relative simplicity and, more importantly, its efficiency. Such was the extent of the findings that in 2016 ESA conducted a follow-on study identified as “Automated Rule-based Validation System (ARVALIS)”. The experience gained from the OPSVAL application serves as the study starting point, which had the intent of improving and refining the VEGA Space tool. Furthermore, this concept was not confined to the utilisation of ESOC space missions operational data, such as those employed from ExoMars, but was also employed to validate operational procedures within the context of the ISS Columbus operations. This resulted in a tool applicability extension to a wider range of mission scenarios. [22]

In order to provide a more detailed analysis, it is necessary to distinguish between three categories of operations within the ARVALIS domain:

- **Control statements**, specifically the TC/TM sequences responsible for actively modifying the status of the Satellite Simulator or the S/C itself.
- **Monitoring statements**, which only verify the actual status of the system through the analysis of TM parameters.
- **Timing and synchronisation statements**, as the actions that regulate any time constraint that has to be implemented in the different sequences.

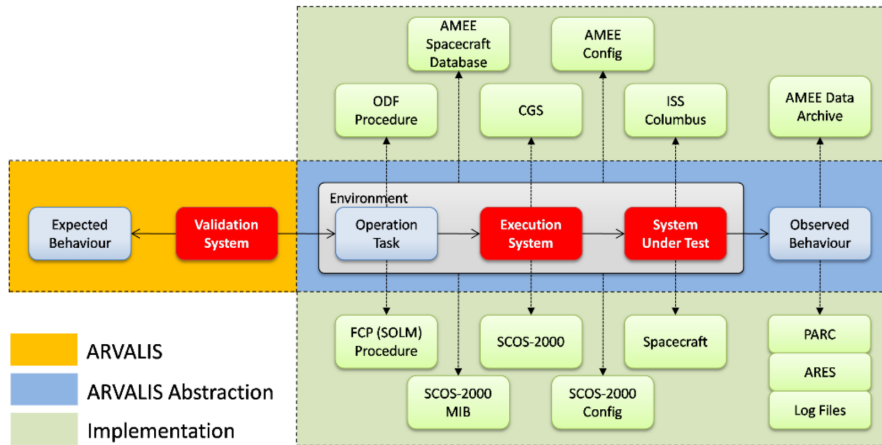


Figure 2.7: ARVALIS FOP Validation Conceptual Map [22]

These operation tasks are integrated as shown in Figure 2.7. It is evident that the validation tool works as a connection between the expected behaviour of the system during the application of the FOP and the actual behaviour observed in the testing environment. The single operation task of the procedure is designed to serve as input to initiate the FOP execution, after an initial consistency check with the MIB as described in [20]. The tool is capable of cooperating with different execution systems through the appropriate interface, thereby providing the necessary satellite data to validate the procedure. With regards to this latter point, it is to be noted that there is a significant difference between this system and OPSVAL. The generation of a FOP validation output is not constrained by an offline examination based on the data stored in dedicated databases, although this function has been maintained: a real-time validation report can be generated by exploiting the automation processes inherent in the tool design. This function has been implemented to drive the execution system in a fully automated manner, by requiring the simultaneous identification of the operation task to be performed, the associated system responses and the corresponding expected nominal parameters to cross-check.

The logic implemented in ARVALIS to accomplish this requirement is based on the Event-Condition-Action rule. [22]

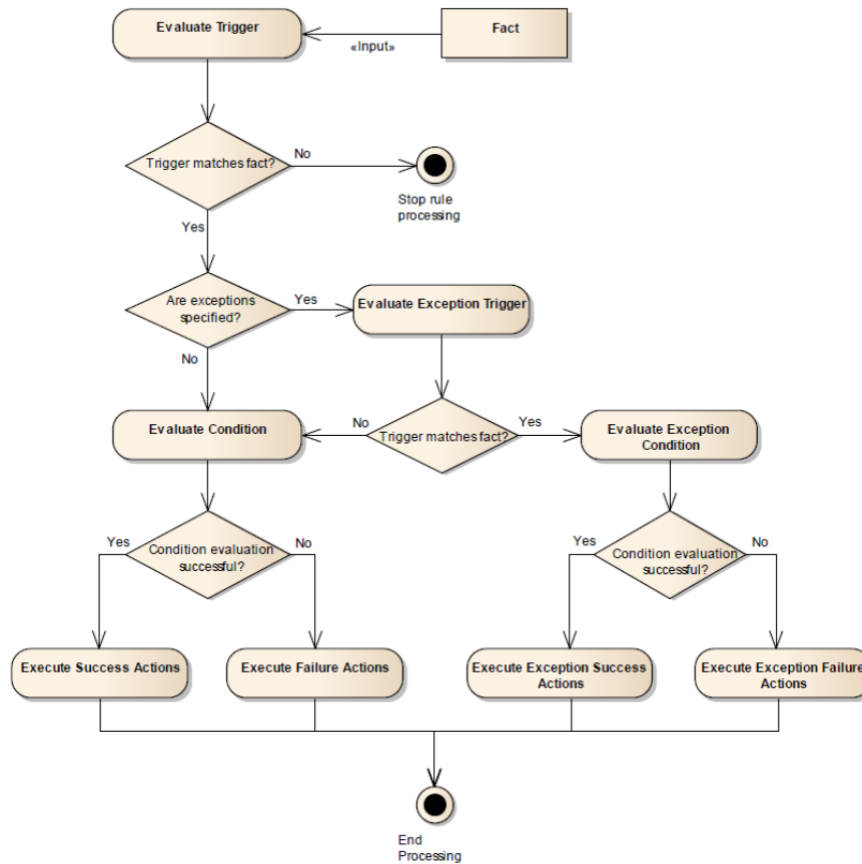


Figure 2.8: ARVALIS ECA Process [22]

The event is associated with a triggering action that can identify the specific operation task. The series of logic tests shown in Figure 2.8 regulate the system responses handling process. This new logic enables the modelling of several actions in response to the obtained feedback, depending on the specific needs of the new tool. For instance, it is possible to connect the FOP to the respective contingency procedure, exit the validation, proceed with different step of the same FOP, etc. . . In any case, the regulation can be expressed in terms of both positive and negative actions for the evaluation of a specific rule. Furthermore, tools projected with the same logic of ARVALIS facilitate interactions with external components by providing dedicated interfaces to multiple data archives, contrary to OPSVAL. This new aspect has the potential to enhance the retrieval of crucial operational data not only during the final execution phases, but throughout the entire validation process.

Eventually, the definition of a more readable FOP validation report was subjected to further considerations and improvements. OPSVAL is also characterised by a ECA in-

spection process to identify when and if any defined rule can be triggered. The concept of reporting to the operator a clear description of the event, if necessary with the correlated error description, is exemplified in Figure 2.9. This example clearly shows the condition of the triggering action, the checks to be verified and the status of the actions performed. At the conclusion of the entire validation process, a summary is generated that comprehensively outlines all the procedures evaluated, along with their respective identified events and descriptions. The graphical representation of this is illustrated in Figure 2.10, where it is possible to find the statistics indications regarding the triggered rules and events along the testing campaign. All the information indicated by these reports are essential to conduct an accurate analysis on the FOP validation tests; however, the reports are disordered and not easy to go through.

Rule G-1-4-TLM-AST50932-STM_12
 Date: 2010-10-19T17:20:53.053+02:00 Procedure: CACE_340 Step: 1 Statement: TLM AST50932

Rule for Log value of this parameter at the generation time of the event: for step 1

When
 Telecommand ASC50586 is successfully uplinked

Checks
 Verification time window: from -10 s to 0 s

Raw value of Parameter "AST50932" is equal to 12 at the end of the time window	AND
(Eng. value of Parameter "AHD00008" is equal to "ON" at the end of the time window	OR
Eng. value of Parameter "AHD00007" is equal to "ON" at the end of the time window)	

Add check Remove check Validate checks

Actions

FAILURE	Error	Value of parameter AST50932 does not match given value constraints
----------------	-------	--

Add action Remove action

Figure 2.9: OPSVAL Rule Description [20]

The reporting function of ARVALIS has been designed to take over all the necessary notions previously produced by OPSVAL but incorporating changes to the format and organisation of the information. Firstly, the Validation Report is generated in several formats commonly well known in work environments, such as PDF and Microsoft Excel Sheets. Moreover, the report is produced for every FOP analysed, identifying the details that characterised the procedure validation process, like the S/C DB version adopted, the start/end time indications, the FOP ID code, etc...

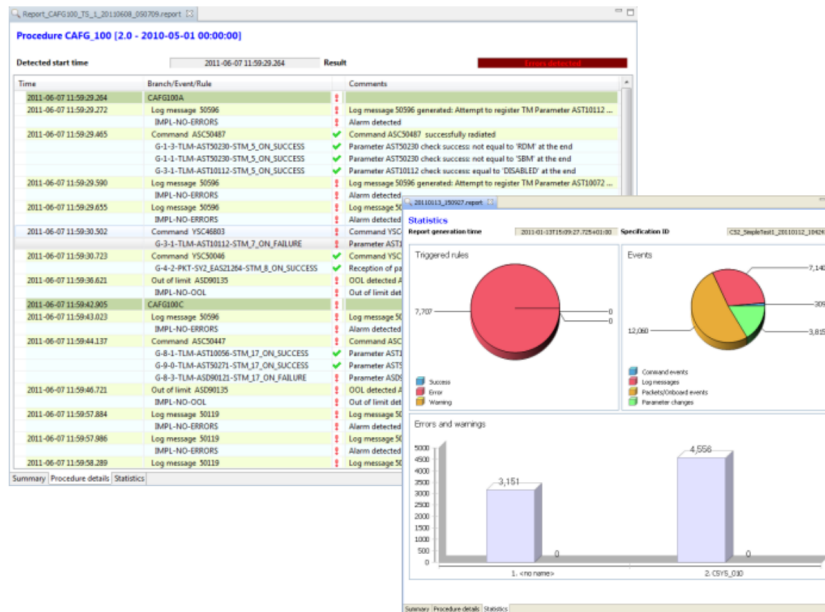


Figure 2.10: OPSVAL Validation Report summaries [20]

Detailed Validation Report



Report_ValSpec_20160209_104842

Creation Time 2016-02-09 10:49:14
Spacecraft DB Version MIB_017
 20140917_TGO_V03.00_EDM
 ROI0
Validation Start Time 2015-01-20 06:00:00 **Validation End Time** 2015-01-20 18:00:00
Validation Specification ValSpec **Last Update** 2016-02-09 10:48:40

Validation specification generated from execution log file test

Consistency Checking Report			
Operation Task ID / Issue	Task Type	Issues / Step	Status
Operation task: GN-FCP-001	MOIS	2	ERROR
Telecommand EGNZ3041 not defined		1	ERROR
Telecommand EGNZ3042 not defined		2	ERROR
Operation task: GN-FCP-290	UNKNOWN	1	ERROR
Operation task GN-FCP-290 not found inside the ARVALIS project DLR_Test			ERROR
Operation task: GN-FCP-501	MOIS	0	OK
Operation task: GN-SVT-601	MOIS	0	OK
Operation task: LL-FCP-210	MOIS	0	OK
Operation task: LL-FCP-250	MOIS	0	OK

Validation Session Report	
Time	Operation Task ID / Fact / Message
2015-01-20 08:20:00	
2015-01-20 09:03:53	Trigger: Generated on-board event 2 by with severity ERROR: FDIR:SEP DETECTION On-board event with critical severity detected. Rule: "NoOnboardEventError" Check successful - No operation (assume condition is satisfied)
2015-01-20 09:54:29	Trigger: Generated on-board event 10 by with severity ERROR: PL:EDM INV MATH OP On-board event with critical severity detected. Rule: "NoOnboardEventError" Check successful - No operation (assume condition is satisfied)

Figure 2.11: ARVALIS Validation Report in PDF format [22]

Then, a list of all the consistency check verified is presented along the corresponding status acknowledgment, followed by the operation tasks that produced an ERROR condition during the validation. In these cases, a brief diagnostic of the failure is given. The result of this new ARVALIS feature represents a clear improvement of the tool reporting function.

2.3. Automated Validation Advantages

As a conclusion of the analysis on the FOP automated validation state of the art, it is worth summing up all the advantages that the design of a new automation tool offers to the FOS Team. This implementation will:

- Lower the typical timeframes for entering and validating the entire FOP plan within the tools already available to the FCT.
- Reduce the operators workload and the number of operators needed.
- Decrease the possibility of the human-factor in potentially major errors during the testing campaign.
- Reduce the associated costs of the validation campaign.
- Standardise the FOP format and design a FOP validation process not specifically tailored and bounded to a single mission, but rather exploiting the standardisation to provide an adaptable tool that operates different mission scenarios.
- Ensure that the operator is verifying the correct version of a procedure, as long as checking that any procedure step is effectively tested in its entirety.
- Facilitate the FOP regression test implementation and handling.
- Exploit replay sessions to assist the FOP elaboration by implementing its execution with the MCS even without the direct Satellite Simulator link.
- Help identifying potential errors and possible software bugs of space and ground segment tools and interfaces connected by exploiting its automated nature.
- Facilitate the proceeding of testing those FOPs that require updating or modification throughout the satellite lifecycle.

This list responds to the needs of Argotec to automatise and standardise the procedure validation and testing processes. Hence, the choice to develop a new FOP Validation tool appears to have been made with the backing of a substantial body of evidence.

2.4. Driving Factors for the Design of a New FOP Automated Validation Tool

It is possible to design validation tools characterised by different working principles, as there is not a unique solution. Usually, the various tools appear to be different from each other not in their fundamental logic, but in architecture details because the design must follow external driving factors [5] like:

- The availability of other specific tools inside the FOS segment that can help the FOP validation.
- The familiarity of the FCT team with the systems and tools on which the validation tool relay. If designing a new instrument implies the training of the entire team on other tools and services, such an approach might prove to be counterproductive.
- The corporate strategy, namely whether in-house software development is favoured and encouraged or whether the acquisition of external tools with greater functional guarantees is preferred.

In the light of what just said, it is clear that one of the core objectives of the proposed validation tool is to serve as a collaborative instrument, capable of functioning in conjunction with other requisite components. However, it is equally important to ensure that the tool is designed in a manner that minimises its dependence on external resources. Proceeding at a lower level of details and considering that the entire FOP validation was previously performed manually by the operators, the automation process must be updated with the relevant changes and needs to cope with the demanding challenges.

Firstly, the FOP shall be written in a machine-readable language and structured in such a way to allow a software to understand and assimilate all the parameter values and limits. Preliminary checks shall be performed on the procedure language and contents, such as the consistency with the Mission Database. The MIB itself must undergo a brief verification by the operator on its actual loaded version. [28] Then, the system must be able to elaborate the entire procedure and autonomously associate the satellite response with the limits indicated to determine whether the single step can be considered validated or not. Once completed the testing on every step, the entire FOP sequence of TC/TM is analysed in its entirety to reach its validation goal. The Satellite Simulator configuration can determine the success or the failure of the test, which implies the necessity to control its status before the validation start. In the end, the procedure evaluation shall be produced within a validation report to consolidate the FOP status, specifying the possible error sources and mitigation actions to follow in case of negative outcome.

3 | FOP Standard Language and Structure

3.1. Machine-Readable Languages Trade-Off

The design of VIPER, the new automated tool that will be integrated into the Argotec FOS environment, must comply with the project drivers presented in section 2.4. A particularly demanding functional constraint concerns the need to deal with FOPs expressed in a machine-readable language. As mentioned above, this requirement has never been faced by the company, which governed the missions carried out so far by exploiting operational procedures in text-only format. It is therefore necessary to carry out an analysis of the machine-readable procedure languages currently available in order to design the optimum new standard for FOP processing.

First steps towards new procedural languages were taken by JPL during the development of NASA Virtual Machine Language (VML), a mission independent human readable script implemented in C. [16] With a high heritage on missions such as Mars Odyssey, Stardust and Genesis, this programming language acts as an interface between the MCS and the satellite's on-board hardware, allowing FOPs to be written in a high-level way that can then be translated into executable instructions. VML is therefore not a machine-readable procedure format, but it does explicitly define the attributes that a FOP must to be interpreted by a computer system. In brief, procedure steps shall be interpreted as modules, which act like libraries of specific functions (defined as sequences of TCs and TMs), blocks, absolute and relative time labels. Different operators shall be considered to handle the actions to be taken, together with the use of conditional loops and event-driven sequencing. In other words, the FOP must be expressed in such a way to enable the creation of a script-like version of it.

Based on this analysis, NASA defined a new Procedure Representation Language (PRL) with the intention of providing a valuable meeting point between human and automated FOP. This option places more emphasis on the procedure editing process, outlining spe-

cific rules for defining and identifying steps, introducing execution branches and verification instructions for evaluating TM parameters against target values. [19] The latter, as suggested by VML, are governed by Boolean expressions and IF/FOR/WHILE cycles.

Other hierarchical languages created to exploit plans for automation, such as the Plan Execution Interchange Language (PLEXIL), are interesting alternatives due to their intrinsic simplicity. In fact, this language is characterised by a tree structure of nodes governed by a conditional logic: each node can be considered as the equivalent of a PRL step, with a set of instructions to regulate the starting point, the execution and the result of the block. [19] Although PLEXIL was successfully implemented for ISS automated operations demonstration, it is an optimal option for planners or translators and not for FOPs. PRL is in fact a more complex language, but, at the same time more suitable for procedures because of its wider range of instructions and specific nesting order. An example of an operations procedure expressed in PRL is shown below:

```

Procedure
Parameters
  id="X" externalType="RPCM"
  parameterType="In"
LocalVariables
  id="Y" externalType="RPC"
ExitModes
  id="exit_success" Message="Procedure exited
  successfully"
  id="exit_verify_failed" Message="Procedure failed on
  verify"
ProcedureTitle
  name="RPCM Power On Reset" number="5.420"
  id="proc_5420"

Step id="step_1" title="Configuring RPCM after powerup"
OrderedBlock id="block_1"
  CommandInstruction id="instr_1"
  Description "cmd [X] Common Clear"
  CommandIdentifier X.RPCMCommonClear
  VerifyInstruction id="instr_2"
  ExitModeReference="exit_verify_fail"
  Description "Verify [X] Power On Reset -- blank"
  Value X.PowerOnReset
  Operator "equal"
  TargetValue "blank"
  GotStep stepRef="step_2"

```

Figure 3.1: FOP Example in PRL [19]

As can be seen in Figure 3.1, the FOP format has a strong similarity with human space-flight operations procedures, because the language has been developed starting from this particular heritage and with the aim of providing a FOP language interpretable both from an automated system and manually by a FCT member. The limitations of human-centric expressiveness led to the analysis of S/C operations languages based purely on Metamodels. As defined in [3], metamodels are a “set of elements which have been heuris-

tically proven to be sufficient to write the vast majority of test and operations procedures”. These metamodels are mapped in several procedure languages that have been successfully implemented in OTS tools, such as:

- OS/COMET®Control Language (CCL)
- Spacecraft Test and Operations Language (STOL)
- Satellite Procedure Execution Language and Library (SPELL)
- Procedure Language for Users in Test and Operations (PLUTO)

These languages have common steps control structures and cycles like IF, WHILE, WAIT, REPEAT, SWITCH and PRECONDITIONS; each language obviously implements them according to its own logic standards.

CCL features a user extensible control language, making it an optimal choice for many commercial Ground Control Centres. [3]

STOL is adopted by many segments for its ability to synchronise TCs with TMs using a language that is relatively easy to learn and use. Another FOP language that allows the writing of simple sequences, but at the same time with a relevant possibility to develop complex procedures without any limits, is SPELL. With respect to CCL and STOL, it provides a better solution in terms of readability, reliability and efficiency. [23]

Moreover, CCL may have some limitations in terms of available libraries and a steep learning curve for users unfamiliar with functional programming languages, and STOL is characterised by problems of integration with external software tools. In the light of these considerations, CCL and STOL can be discarded as languages of choice.

SPELL is based on Python specifications and is characterised by built-in functions for the main actions related to typical operations (acquiring TM parameters, verifying them, sending TCs and acknowledging their execution status, etc...).

Example 1:

```
Send('TC1')
```

Example 2:

```
Send('TC1', Verify=['TM1', eq ,10], AdjLimits=True )
```

Figure 3.2: SPELL Procedure Code View [23]

Figure 3.2 illustrates a FOP TC send instruction example, which highlights the simplicity and compactness of this language.

Eventually, PLUTO is an open standard language published by the European Cooperation for Space Standardisation (ECSS) to operate in the context of space mission procedures. Being associated with an ESA standard, PLUTO is compatible with SCOS-2000 based MCS and it is currently adopted by major organisations such as the Canadian Space Agency, EUMETSAT and ESA itself. [15]

The ECSS-E-ST-70-31 Standard provides the definition of a Space System Model (SSM) structured by different data: i.e. TMs, TCs, ranging, ground segment commands and measurements. This type of data is handled by PLUTO as objects, where each object can represent system elements, reporting data, activities and events. [11]

Such language definition guarantees the adaptability to different SSMs, independently of the application specifics. PLUTO allows a FOP development characterised by single activity calls with nominal and contingency paths, thanks to asynchronous functions and with the use of prefixed and dedicated language constructs. Their implementation within the procedure helps to organise the FOP in steps like PRL and, more importantly, provides a simple language grammar for operators who are not familiar with this language.

Procedure Name	Switch on Gyro5 in Fine Mode
PLUTO script	<pre> procedure preconditions wait until Gyro Temperature > 60 degC end preconditions main initiate and confirm Switch on Gyro Converter; initiate and confirm Switch on Gyro5; initiate and confirm Gyro5 Fine Mode; end main end procedure </pre>

Figure 3.3: FOP Example in PLUTO [11]

Considering all the advantages of PLUTO, it seems logical to select this standard as the FOPs language of choice. [30] One of the main advantages with respect to SPELL concerns the need to update or modify FOP over time and during the mission itself to ensure its compatibility with the S/C systems: SPELL lacks flexibility and is considered to be more time-consuming and error-prone, whereas PLUTO can exploit its standardised commands and constructs to facilitate the process. [15] Another positive aspect is that, although PLUTO is a machine-readable language, it is sufficiently clear to be interpreted by human operators too, while the same cannot be said of SPELL which has the same aspect of a Python code.

Moreover, the MCS software used by Argotec is able to generate several spacecraft languages including PLUTO but not SPELL: this asset can facilitate the development of the necessary interfaces between VIPER and the Mission Control Software.

3.2. PLUTO General Architecture

Remembering that the main activities of a Flight Operations Procedure are the TCs transmission and the request of telemetry packets and parameters, the latter do not only concern the parameter itself, but may also be related to the verification of TC effects. From now on, this specific type of TM will be referred to as a verification or a check. On the other hand, the term 'action' is used in a more general sense to refer to anything that goes beyond the contents already described, for example the creation of flags/events, the sending of warning and error messages, the requests for user inputs, etc. . .

To fully comply with the ECSS standard definition and all PLUTO rules, the ECSS-E-ST-70-32C31 is used as the main reference for the study and as a guide for the procedure editing. In accordance with this document, a PLUTO FOP consists mainly of five blocks: the declaration body, the preconditions, the main body, the watchdog and the confirmation body. Within these blocks, the procedure content must be further subdivided into steps to ensure that the FOP itself is correctly managed by the tool in a hierarchical and sequential order.

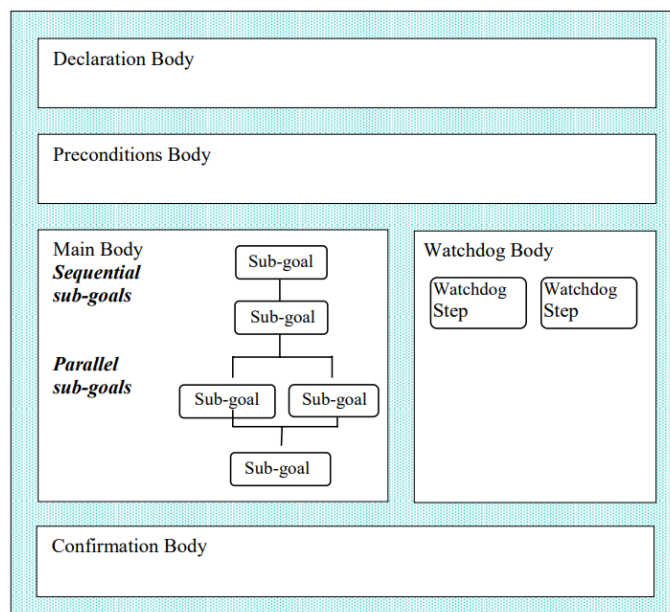


Figure 3.4: PLUTO FOP Structure [11]

The Declaration Body is the first block of the FOP containing the general description of the procedure, in which the final objective to be achieved through the application of the specific FOP is clearly specified. Subsequently, the Preconditions include the first TC/TM sequences which define the initial conditions and control actions that, once verified, ensure the possibility to run the procedure. Within this block, the sequences start to be grouped into different steps. Each single procedure has its own preconditions, but there are some typical checks common to several FOPs: the four main steps are proposed below.

- **Required Inputs:** this is the first step of each FOP, and it is reporting the variables and initial conditions required by the specific procedure, i.e. indications related to the simulator, coefficients to be inserted during the procedure elaboration, TC parameters, etc. . .
- **Radio Configuration Verification:** a step aimed at verifying the correct state of communication between the satellite (or its simulator through the EGSE) and the Mission Control Centre.
- **S/C OPMODE Verification:** a step dedicated to the investigation of the satellite's operating mode at the moment of the FOP execution.
- **"Are You Alive" Test:** step containing the necessary command to verify that the satellite is operating correctly and that it can receive the data sent from the ground, process them and send back the corresponding telemetry data.

Any additional and specific checks of a given FOP must be included within the same block. Otherwise, if one of the four checks described is not necessary and/or results in an error in the FOP, it must be removed. Even if only one of these steps is not verified or presents an off-nominal condition, the execution of the procedure is immediately aborted. The Main Body positioned after the preconditions check represents the primary block containing all the FOP-specific commands and actions that actually specify the procedure. There is no limit to the number of steps required for the procedure within this block.

In contrast to the two blocks described, the presence of the Watchdog Body is not strictly necessary and, consequently, its absence does not result in a PLUTO consistency error. The function of this field is defined as a control action of the Main Body, such that it is performed in parallel with the execution of the main structure. If an anomaly occurs during the execution of the procedure, the function of the watchdog is to intervene with a plan of commands and contingency actions to resolve it completely or, at least, mitigate the effects of the recorded errors. Eventually the Confirmation Body has the function of verifying that all the procedure objectives stated in the Declaration Body have been achieved and thus declares the success or failure of the FOP.

Figure 3.5 illustrates the PLUTO FOP execution logic to summarise the interactions between the various blocks.

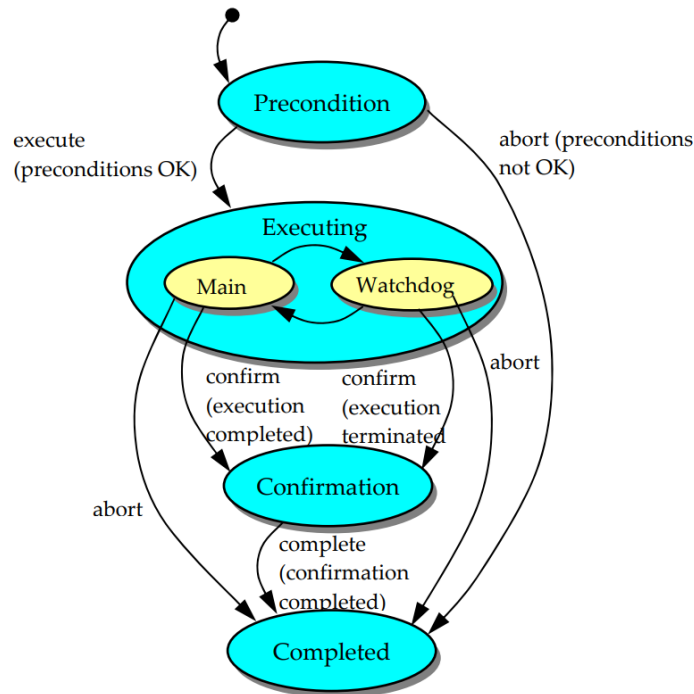


Figure 3.5: PLUTO FOP Execution Stages [11]

3.3. FOP Step Definition

As already mentioned, the architecture of the FOP is based on a block and step structure to help the tool visualise the sequence of procedure actions to be performed. Except for the procedure initialisation commands, preconditions, main, watchdog, their "end" commands and any associated comments, all the content of the procedure must be organised within steps. The definition of this procedure structure was guided by the grammar of the PLUTO language itself which, in addition to its standard commands that will be discussed in more detail in the next paragraph, offers the possibility of creating steps that are managed as real blocks of operations. Each step is formalised with the command *initiate and confirm step* and is terminated with *end step*. It can be structured in a more or less complex manner: within it, a step allows an organisational scheme identical to the FOP general architecture (preconditions, main body, watchdog body, etc...). [6]

At the same time, a step can only contain a single command, telemetry request or action. Seven different types of steps, differentiated by content, can be highlighted and reported hereafter.

- **Definition of Required Inputs:** either individually or through input arrays, with their consequent display representation in order to obtain an operator verification.
- **Request of a TM parameter:** to verify its value by operator verification or by automatic verification of the results obtained.
- **Request of several TM parameters:** same logic as for a single telemetry request, but for a larger number of parameters, as long as they belong to the same TM packet.
- **Request of a TM packet:** this represents the necessity to retrieve a whole TM packet in order to make its parameters available to the tool and to the operator.
- **Sending a single TC:** step defined by the sending of a telecommand to the MCS, which will elaborate and return its acknowledgement status to the tool.
- **Sending a TC and requesting a TM parameter:** possibility of sending a TC and immediately requesting a TM parameter to verify the telemetry correlated to the TC elaboration.
- **Sending a TC and requesting a TM packet:** same concept of the previous case, but distinguished by the request of a complete telemetry packet.

3.4. PLUTO Commands

The commands available in the PLUTO language and their possible interactions are described and fully investigated in the reference document [11]. Nevertheless, the FOP PLUTO model proposed in this chapter represents the standard that has been designed appositely for the VIPER development, in order to present a machine-readable language that is well-defined and structured by means of case associations.

To this end, the entire available dictionary was consulted, but only some specific logical language structures were highlighted and selected: these were considered sufficient to ensure the most complete coverage of FOP cases. The list of the commands adopted is given below, with a brief explanation of the main functionality of each command:

- **PROCEDURE:** the start of a FOP is indicated by the *procedure* command, which must be left as the only line element. The entire procedure must then be specified and, to close the procedure block correctly, *end procedure* must be entered.
- **PRECONDITIONS/MAIN/WATCHDOG:** the start of these blocks is obtained by the respective calling *preconditions*, *main* or *watchdog* commands which must be left as the only line element.

As for the procedure command, the content of each block must be specified and, to close it correctly, the *end preconditions*, *end main* or *end watchdog* command must be entered.

- INITIATE AND CONFIRM: This command allows to initiate a general action and to impose a confirmation alert to be received once its execution has been recorded. It should be noted that the confirmation is not related to the result of the task but to the acceptance and execution of the action.
- INITIATE AND CONFIRM STEP: The same concept as *initiate and confirm*, but it refers only to the initialisation of a step. It therefore allows the step to be separated from the rest of the procedure and requires the *end step* command.
- DECLARE: the command allows to create a variable or constant.
- GET VALUE: The *get value* command is used to extract a specific TM parameter from a particular telemetry packet or set of parameters.
- IF: The *if* loop follows the logic common to other programming languages, namely it represents a loop governed by the truth of the logical condition imposed. After the *if* condition, in the same formulation line, the keyword *then* must be added as the last word in the same line of formulation. To conclude the loop, *end if* must be inserted.
- WITH: This keyword allows additional specifications and inputs to be attached to a command or action. It also requires *end with* to close the loop. It is strictly linked to the *arguments* keyword in order to specify the parameters required for a function or command.
- ARRAY: the command allows to create an array. Each array counts specific components, specified as constants or variables. To end the construction, *end array* must be entered.
- RECORD: the *record* command registers constants or variables, usually within an array. It requires the *end record* command to close it.
- RAISE EVENT: This command is used to create an event associated with a condition defined after a previous telemetry check. The creation of this event can trigger the initiation of other commands or actions.
- ASK USER: this command indicates an interface action between the operator and the procedure. Namely, the FOP requests an input from the operator to associate with a specific variable or condition before continuing the execution of the procedure.

- **INFORM USER:** differently from ASK USER, this command function has the scope to inform the user by displaying an information string. The sentence to be printed must be specified between “ ”.
- **LOG:** this command can be used to request the printing of a specific constant value, a variable or a string. If the content described after its call occurs between ’ ’, it will be printed and reported to the operator as a sentence.

3.5. FOP Parameters Identification

Before proceeding with the definition of the PLUTO FOP standard for the VIPER application, it is necessary to introduce the peculiarities of some key elements of the procedure. According to the PUS standard, each telecommand shall be identified through a TC name and three numbers, namely the type, the subtype and the application ID. [7] The three-digit numbers identification avoids ambiguity when calling up a single command, since several TCs may have the same name and the same type and subtype numbers: in such cases, the APID is essential for correct identification.

Another specification related to the TC send step concerns the parameters “execution time” and the “ackFlag”. The time indication is required whenever a timetagged telecommand is sent on board to populate the S/C schedule and must be expressed in ISO time format. The ackFlag instead represents an indication on the acknowledgement status of the telecommand. The value of this parameter is represented by a number between 0 and 15, which indicates the possible steps of the action relative progression (i.e., the action triggered by the TC has been verified, accepted, completed, etc).

Telemetry packets have a similar identification to the telecommand, again specified by a packet name, type, subtype and APID, but with the insertion of two additional numbers. The latter are referring to the first and second “flexible” Packet Identification numbers, expressed as PI1 and PI2: different packets can in fact share all the other cited values, but not PI1 and PI2. Regarding the single TM instead, the parameter name is sufficient to uniquely identify the information.

3.6. PLUTO FOP Standard

The PLUTO commands proposed in the previous paragraph were selected for their functionality in creating a standard FOP language that can be easily interpreted by VIPER, while at the same time being consistent with ECSS guidelines. The following command list provides the most important references for writing a procedure in accordance with

the new PLUTO Standard generated ad hoc for the FOP validation. Each indication is accompanied by instructions on how to correctly express the designed logic.

PLUTO FOP General Structure: the selected procedure provides a specific block structure containing the initialisation and closing of all the PLUTO principal blocks, namely the *procedure*, *preconditions*, *main* and, possibly, *watchdog* commands. This list order must be respected.

```

procedure
  preconditions
    [...]
  end preconditions
  main
    [...]
  end main
  watchdog
    [...]
  end watchdog
end procedure

```

Listing 3.1: PLUTO FOP General Structure Example

Step Creation: this passage concerns the use of the formulation *initiate and confirm step*, which is followed by the name intended to be assigned to the step. This name must not contain any spaces, as it will be collected as a single element by VIPER. Furthermore, within the step, all actions must be entered according to the seven types specified in section 3.3.

```

initiate and confirm step Safe_Mode_Insertion
  [...]
end step

```

Listing 3.2: Step Creation Example

Required Input identification: the data required to validate the FOP can be identified as either a single parameter or as an array. To create the variable containing this information, the command *declare* must be entered followed only by the name of the input, which should be written without spaces as a single element. The value or the string specification of the variable must be indicated later in the FOP. In the case of an array input, the *declare* command must be used to create the input in the same manner as the creation of the single input, but this time followed by *with* and *array*. At this point, it is

mandatory to utilise the *record* command on the subsequent line, followed by the assignment of the variable name. The formation of the single component must be repeated for each new element, with each instance requiring the assignment of a new line.

```
declare Start_time
-----
declare Vector with array
  record Vx end record
  record Vx end record
  record Vx end record
end array
end with
end step
```

Listing 3.3: Required Input identification Example

TC Send: a TC send uses the *initiate and confirm* command. The correct formula for reading the command involves entering the abbreviation "TC" immediately after the command, followed by the keyword *of* and, consequently, the telecommand ID code. The latter is made up of the abbreviation TC, the type and sub-type specification divided respectively by underscore and followed by *of* again: a further last number indicates by the TC apid digit. The numeric values (type, subtype and apid) must be expressed by means of three-digit numbering.¹

```
initiate and confirm TC of TC_001_002 of 003
```

Listing 3.4: TC Send Example

TC Send with Specific Parameters: the concept is analogous to the "TC send" with the exception that additional information is included in the telecommand. Following the TC send, the *with* command must be entered, followed by the *arguments* keyword. Each line below shall indicate the name of the parameter to be added, ":@" and the value to be associated with the variable (digit or string). In case of a string parameter it is necessary to indicate it as a single word. Unless differently specified, the value indicated is taken as input of an engineering parameter (ENG), but a RAW format parameter can also be specified. In this case, after the equality operator, the two-letter abbreviation representing the nature of the parameter must be entered in capital letters, followed by the desired RAW value. The available acronyms are SH (Short), LO (Long), US (Unsigned Short),

¹The type, subtype and apid numbers indicated, respectively 001, 002, 003, are not meaningful, they are reported only to appreciate the standard layout

UL (Unsigned Long), FL (Floating Point), CH (Char), BO (Boolean), OC (Octet String), ST (String), BS (Binary String), TI (Time).

```
initiate and confirm TC of TC_001_002 of 003
  with arguments
    Temperature_set := 20,
    Temperature_set := LO 293.15
  end with
```

Listing 3.5: TC Send with Specific Parameters Example

TC Send with Directives: the same functioning process of previous TC points is employed, with the addition of a new feature that allows for the incorporation of telecommand-specific properties. These specifics include the TC execution time and/or the *ackFlag* value. The two data elements must be inserted within a *with* loop, which is marked by the keyword *directives* and must be entered after the *with arguments* loop, if present. In the event that the value of the *ackFlag* is not explicitly expressed, a value of 11 will be considered by default. In the absence of any prior declaration, the two variables must be initialised.

```
initiate and confirm TC of TC_001_002 of 003
  with directives
    ackFlag := 9,
    exe_time := 2013-03-02T00:00:00.000
  end with
```

Listing 3.6: TC Send with Directives Example

TM Parameter Request: to indicate the request for a specific telemetry parameter, it is necessary to give a name to the variable associated to the searched data. This variable is then matched through the "==" operator to the *get value* command, which is followed by the telemetry name within ". It is also necessary to specify in which telemetry packet the value is contained, and this is indicated with the *of* keyword followed by the packet code, showing type and subtype.

```
TM_04 := get value TM_04 of TM_001_002;
```

Listing 3.7: TM Parameter Request Example

TM Packet Request: to indicate the retrieval of an entire telemetry packet, it is instead necessary to provide not only the type and subtype of the packet, but also the apid, PII

and PI2 digits. This ensures that the downloaded packet is uniquely identified. The request consists of a *verify packet of* command, followed by the same identifier of the single TM parameter request, simply lengthened by the formula *of* and the three more specific digits. It should be noted that these latter digits are to be separated by underscores.

```
verify packet of TM_001_002 of 003_004_005;
```

Listing 3.8: TM Packet Request Example

Display Representation: the “inform user” command enables the return of a string contained between quotes to the display. For the sake of standardisation, it is necessary to use this command when it is not followed by the user’s request for input, once the actual display print has been visioned.

```
inform user 'TM 0C004 OFF-NOMINAL, S/C IN SAFE MODE'
```

Listing 3.9: Display Representation Example

Display Representation followed by an Operator Input: the representation may refer to both the required inputs or the telemetry values. The required command is *log*, followed by the name of the object to be printed, which must be provided as a single name with no spaces.

```
log Start_Time
ask user
```

Listing 3.10: Display Representation with Operator Input Example

Operator Input Request: operators can be interrogated for an input action prior to the transmission of a TC or after a telemetry check. Again, the name must be entered without the inclusion of spaces.

```
ask user New_Mode
```

Listing 3.11: Operator Input Request Example

Automate Nominal/Off Nominal TM Parameter Check: This type of verification involves the acquisition of one or more telemetry parameters and the implementation of the *if* loop. The standard structure therefore involves the TM request as previously explained and the *if* command followed by the telemetry value identifier, the *is* keyword and the respective logical operator, depending on the request to be evaluated. Five

different operators can be implemented: equality (=), inequality (! =), less than (<), greater than (>) and the belonging to a certain range of values. These are followed by the reference object of the analysis, given within ' ': in the case of the range operator, the reference object shall be inserted between the two "<".

The automatic structure of the VIPER control provides for an analysis capable of exploiting the *if* loop and thus creating an event associated with a non-nominal value if the reported indication is actually found. The warning string to be reported to the user shall be indicated in the next line through the *inform user* command.

The final step consists of creating an event with *raise event*. Here, after the PLUTO command, the name associated with the event must be entered, again without spaces. Moreover, the automated TM check can be associated with a contingency. In this case, the absence of the event creation command is justified by the treatment of the non-nominal condition directly in the same procedure step. The TC to be sent will necessarily have to be of the typology with an additional parameter, to allow the user input to be associated as a parameter of the command.

```

if TM_04 is != Safe_Mode then
    inform user 'The S/C is not in safe mode'
    raise event SC_not_safe
end if
-----
if TM_04 is != Firing_Mode then
    inform user 'The S/C shall exit Manoeuvre Mode'
    ask user new_OPMODE
    declare OPMODE
    initiate and confirm TC of TC_001_002 of 003;
        with arguments
            OPMODE := new_OPMODE
        end with
end if

```

Listing 3.12: Automate TM Parameter Check Example

TC and TM Delay Time: Prior to the transmission of a TC send or TM request, it is possible to indicate the delay information that is intended to be applied when sending the command and/or the telemetry demand. This is achieved by specifying the PLUTO command *wait for* together with the name of the variable and the time to be effectively waited, expressed in seconds.

```

declare delay_TM
delay_TM = 120
wait for delay_TM

```

Listing 3.13: TC and TM Delay Time Example

A summary table of the principal PLUTO commands adopted within the procedure model is provided, together with the respective calling actions. These are presented alongside a column of additional notes recalling their main use within the FOP standard.

FOP Function	Initial PLUTO Command	Final PLUTO Command	Notes
Create a step	initiate and confirm step	end step	Followed by the step name
TC send	initiate and confirm	-	Followed by “TC” and the type, subtype and apid values
TM parameter	get value	-	Followed by the parameter name and the TM packet type and subtype
TM packet	verify packet	-	Followed by the TM packet type, subtype, apid, PI1 and PI2 values
Declare a variable	declare	-	Single and array required inputs initialisation
If cycle	if [...] then	end if	Raise event logic, preconditions check, TM check
With function	with	end with	TC parameter specification
Array function	array	end array	Required inputs array creation
Components record	record	end record	Array components registration
Raise of an event	raise event	-	Event creation after an if cycle
COntfirm by the Operator	ask user	-	Single command
Operator input	ask user	-	Command followed by the input to insert
Inform operator	inform user	-	Information string display
Log	log	-	Information string display followed by the ask user command
Time delay	wait for	-	TC and TM time delay

Table 3.1: PLUTO FOP Standard Commands

4 | VIPER Design Definition

After the description of the PLUTO language selected for the new Argotec FOP standard, the actual VIPER design will be presented. The path followed for its development considers the notions acquired with the high-level functional analysis of procedure validation tools, already presented in chapter 2, as the project starting point. These concepts, together with the needs of the company, are translated into specific requirements that the VIPER design must fulfil. Their definition and understanding will eventually lead to the final tool architecture definition.

4.1. Delineation of VIPER Requirements

In accordance with the guidance set out in ECSS-E-ST-10-06C [12] and the Expanded Guidance for NASA Systems Engineering [25], the process of writing requirements must adhere to specific criteria. Primarily, they must be unique, unambiguous and described in quantifiable terms. This is of particular importance as it allows for an objective verification of their fulfilment. Furthermore, technical requirements must be backwards-traceable, in order to facilitate the tracing of each requirement back to its originating source.

Requirements can be grouped in two main categories: mandatory and recommendations. As the term itself suggests, mandatory requirements are those that must be satisfied to meet the design specification: they are distinguished from the others by the use of the word *shall* in their specification. In contrast, requirements that may improve overall system performance, but are not necessary to achieve the main objectives, are identified by the key word *should*. Moreover, requirements can be further subdivided into two additional categories.

- **Functional Requirements**, which define what the system shall do, including the functionalities and actions it must be capable of performing to meet the user needs
- **Design Requirements**, which specify how the system must be implemented to accomplish all the functional and non-functional specifications. They represent the technical guidelines for the tool development

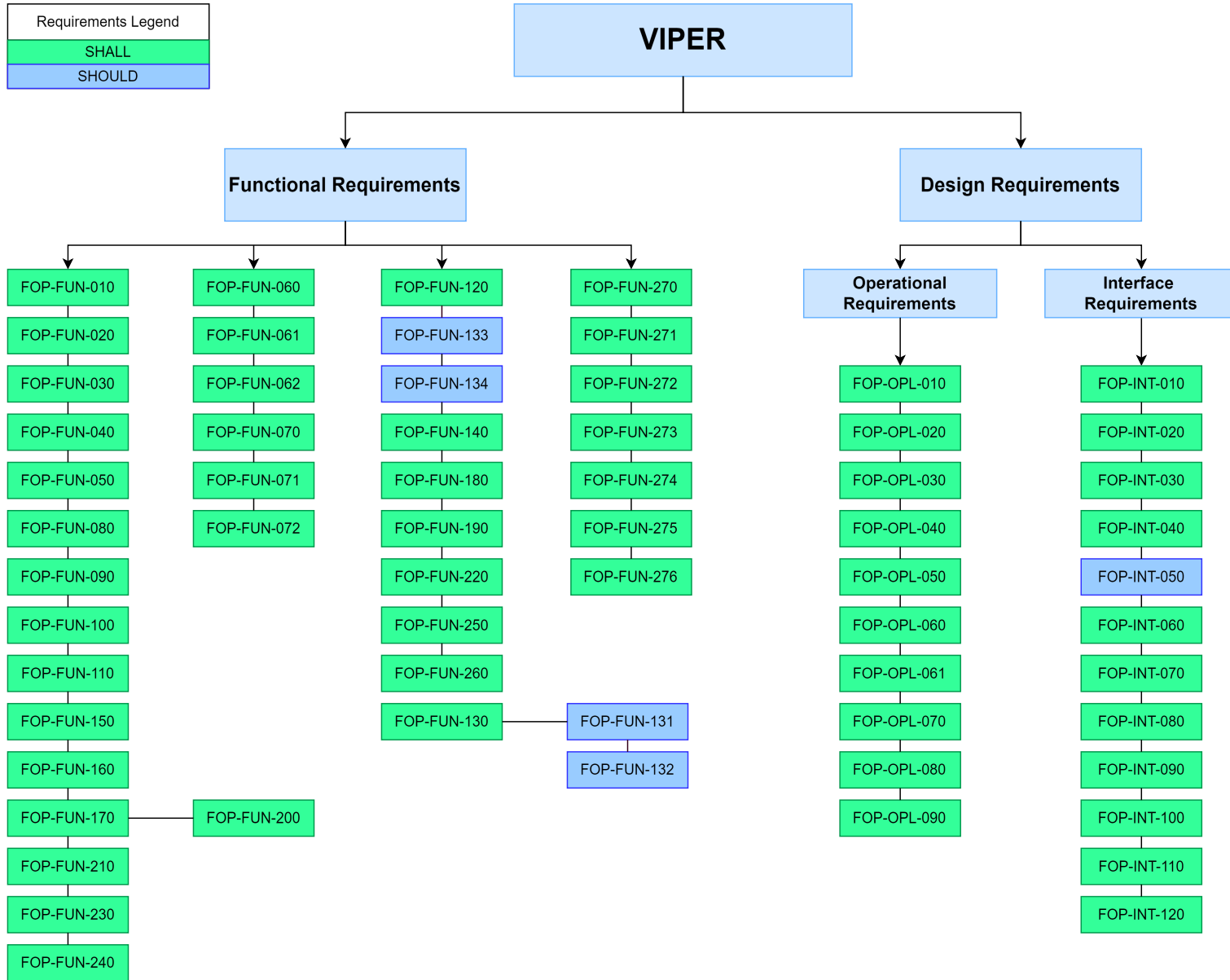


Figure 4.1: VIPER Requirements

These categories are complementary and they are both fundamental in the development of a comprehensive and functional software tool. Requirements are not only merely suggestions: they must be verified throughout the design phase and during the testing campaign. In order to accomplish the validation, each requirement shall be formed by a reference index, the main statement and the indication on how it will be verified. The latter verification methods are identified between four possible alternatives: Inspection, Analysis, Review of Design and Test. For the specific VIPER application, only the "Review of Design" and the "Test" methodologies are implemented, because largely sufficient. Figure 4.1 illustrates the VIPER requirements, clearly distinguishing between "mandatory" and "nice to have" ones. For completeness, the totality of these requirements is reported in Appendix A, coupled with the associated verification method.

4.1.1. Functional Requirements

By definition, the functional requirements are inherently linked to the functional analysis performed on the validation tool, focusing on what has to be accomplished by VIPER. Figure 4.1 presents a comprehensive categorisation of these functional requirements, which can be broadly divided into four distinct groups based on their content and context of application. The initial list of functional requirements represents most of the overall requirements. The most general requirement serves to elucidate the principal objective of VIPER, namely the capability to validate FOPs [FOP-FUN-010].

Starting from this indication, the requirements are progressively delineated in increasingly specific terms. A close examination of the statement reveals that it is indeed possible to reconstruct all the functionalities that the tool is designed to provide. The initial aspects to be addressed concern the PLUTO FOP, which will serve as input to VIPER.

The tool must be capable of processing the entirety of the procedure steps in their hierarchical order, ensuring the correct execution of the FOP. [FOP-FUN-040/050]

In consideration of all the PLUTO procedure properties addressed in chapter 3, the tool shall request confirmation from the operator regarding the correctness of the required inputs reported. At this point, VIPER shall verify and cross-check that the PLUTO structure and grammar are correctly respected. In order to gain a deeper understanding of the tool operational context, Figure 4.2 illustrates the FOS units that will interface with the validation tool.

The cross-check with respect to the MIB is essential to VIPER in order to maintain consistency between the procedure TM parameters, TM packets and TCs and those present in the Mission Database. Once the entire FOP content has been verified, the procedure

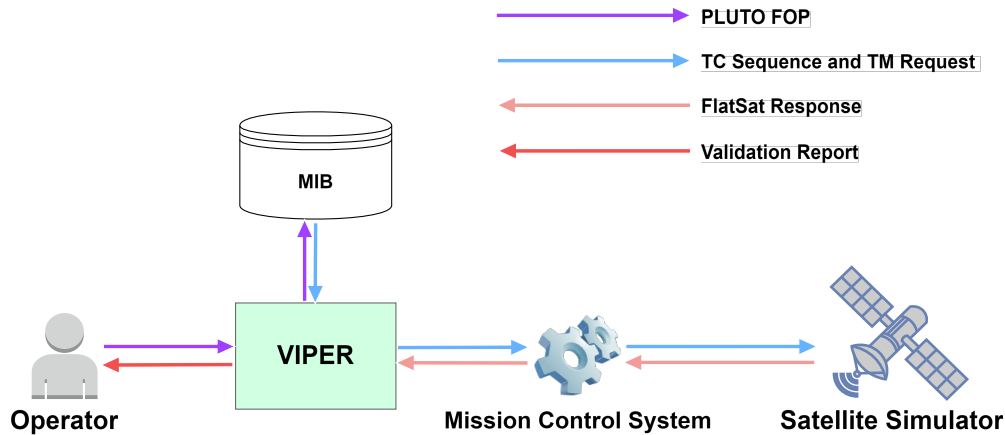


Figure 4.2: VIPER and FOS units interfaced

shall be executed sending the telecommands and requesting the telemetries directly to the MCS. The software will be in fact connected to the Satellite Simulator in order to forward requests and record the responses obtained from their processing, acting as an interface to and from VIPER. Several requirements interest then the comparison between the responses obtained and the nominal TM parameters indications coming from the initial FOP. The outcome of this analysis will be presented in the final Validation Report.

The second group comprises those requirements that differentiate between an ERROR and a WARNING condition [FOP-FUN-060/070]. These two conditions are fundamental to the validation process, as they enable the identification of whether an off-nominal condition represents a critical showstopper or merely a point of concern. Furthermore, the VIPER system must report the triggering condition of these events to the operator, thereby enabling the operator to continue or interrupt the validation process in the event of a WARNING. Conversely, an ERROR association must be correlated to a forced interruption in order to prevent any disruption to the automated functioning of the tool. [FOP-FUN-072]

The requirements list that in Figure 4.1 begins with FOP-FUN-120 and ends with FOP-FUN-260 is strictly correlated to the second group, as it defines which cases are associated with a WARNING or an ERROR.

- **WARNING Status:** it indicates when the TM parameter expressed in the FOP does not match the Satellite Simulator corresponding response and when a TC is accepted by the satellite but not correctly completed.
- **ERROR Status:** associated to a wrong PLUTO structure in the procedure, which does not allow the validation process to be continued autonomously by the tool.

Alternatively, it may be attributed to wrong FOP contents, which can be identified as a discrepancy between the TM and TC parameters with respect to the MIB. Moreover, TC, TM parameters and TM packets may be correctly indicated but not sent, not received or not elaborated by the MCS software. These actions are catalogued as errors, just as the missing reception of the requested responses from the Satellite Simulator.

The final cluster of requirements (from FOP-FUN-270 to FOP-FUN-276) expresses the indications and guidelines related to the Validation Report. In these requirements it is possible to find what shall be contained inside the report in terms of contents and the attribution of the FOP Validated/Not Validated status rules. Such conditions are relatively straightforward; the presence of at least one warning or one error compromises the procedure validation. Conversely, if all the VIPER tests on the FOP are successfully overcome, the procedure can be considered validated. In any case, the report is intended to assist the operator to easily identify the source of any off-nominal parameter and its location. In this respect, the VIPER Validation Report is designed to meet these needs.

4.1.2. Design Requirements

The design requirements are intended to define the way in which all the functionalities described will be performed and granted, dealing with more technical aspects of the tool design. During the VIPER project definition, the design requirements were grouped into two different categories: operational and interface requirements.

In accordance with the specifications set forth by [33], operational requirements serve as the primary source of constraints on the actual design, specifying the tool technical characteristics. A clear illustration of this can be observed in the unambiguous identification of TCs and TMs. Telecommands and telemetries are uniquely catalogued in the Mission Database through their Type-Subtype-APID numbers, the TM packets with Type-Subtype-APID-PI1-PI2 numbers, and the single TM parameters through their names, as previously explained in section 3.5. It is so imperative that VIPER correctly identifies and deals with telemetries and telecommands reported in such a manner. [FOP-OPL-010/020/030]

Furthermore, the tool shall be capable of validating FOPs that differ from one another. This functionality implies that VIPER must be designed to treat them in the same way, ensuring that the validation is carried out in the same manner. VIPER shall also produce the same outcome if an input FOP is analysed more than once with different operational modes. [FOP-OPL-090] Operational requirements pertain to the validation process per-

formance of FOPs too. The application of VIPER is intended to assist and reduce the operator workload, but the introduction of automation shall represent an efficient alternative. In this context, the functioning of the tool shall not result in an increase in the computational time required by other script running concurrently within the MCS, as it should maintain a consistent computational effort. [FOP-OPL-070]

The interface requirements are instead dedicated to a more detailed specification on how VIPER must interact with other systems. This begins with the operator, who will insert the FOP as input: the tool must be then able to interpret the PLUTO procedure language and translate the FOP contents in a format readable by other Argotec MCC elements. This will ensure efficient compatibility with the FOS. [FOP-INT-020/030/040] To ensure the fulfilment of these requirements, the selection of the scripting language for the tool turned out to be Python. As illustrated in Figure 4.3, the collection of MIB data is possible through the submission of query requests to the Mission Database. [FOP-INT-050]

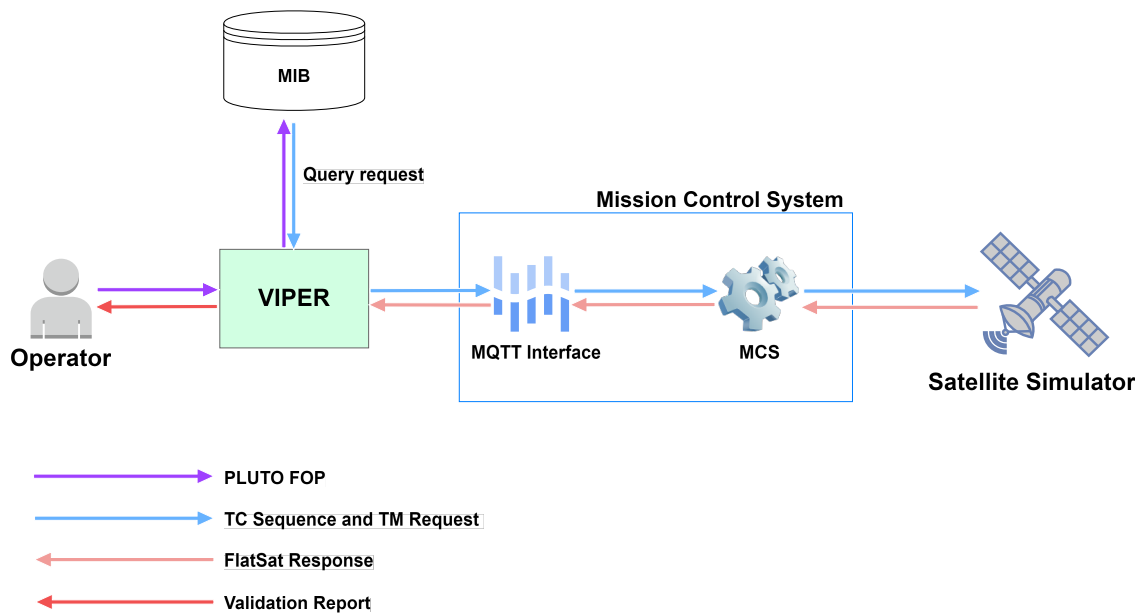


Figure 4.3: VIPER Interfaces

With regards to the connection between VIPER and MCS, the tool will exploit an MQTT interface that has already been developed by Argotec for the purpose of connecting the Mission Control Software and the FOS Procedure Automation software. [FOP-INT-080] Through this interface, the tool will be able to receive the TCs execution status and the telemetry coming from the Satellite Simulator. Eventually, the results of their elaboration must be returned to the operator in a clear and human readable format. In this case, the Validation Report will serve as an offline document containing all the interactions

between VIPER and the simulator. However, a live display of these interactions is also planned during the FOP execution. [FOP-INT-120]

4.2. VIPER Architecture

The tool architecture has been designed to meet all the functional, operational and interface requirements discussed in the previous paragraphs. To facilitate its comprehension, Figure 4.4 presents the detailed VIPER architecture scheme.

Here, the FOP is received as an input and is subjected to a preliminary verification on the procedure PLUTO language. The check consists in a comprehensive analysis of its structure, its grammar and its consistency with the PLUTO Standard defined for VIPER.

Remembering that VIPER is programmed in Python, it is easily compatible with the JSON file format. [17] In principle, this feature can be exploited to allocate the PLUTO FOP contents in such structured file, offering broad compatibility with all elements of the FOS. The various facets of the procedure could be easily accessible and manageable thanks to the JSON internal structure of lists and dictionaries. As this new file is universally readable and modifiable across different systems and platforms, it increases the interoperability and flexibility of the entire VIPER system.

Therefore, once the procedure is considered "well written", it is necessary to draft its contents in the corresponding FOP JSON format with an appropriate function. Following this translation, the FOP consistency check against the MIB is performed through a Query request logic: the data verified with the Mission Database concerns the names and specific properties of TCs, TM parameters and TM Packets. If one of the preliminary PLUTO – Mission Database Checks identifies an error, the process is immediately transferred to the Validation Report Generator, which notifies the operator with the detected error event.

Otherwise, the procedure validation continues, implementing an adaptation of the FOP from its initial JSON draft to another JSON adaptation in order to correctly insert the FOP into the MCS MQTT interface. The VIPER system is utilising an interface that was developed by Argotec prior to the design of the automated tool. ¹

This step may appear tedious and repetitive at first glance, but it allows for the avoidance of a new software interface design. The latter would undoubtedly be a more complex and time-consuming approach than adapting an already validated FOS tool.

¹The MQTT interface was already developed by Argotec FCT at the moment of the VIPER design

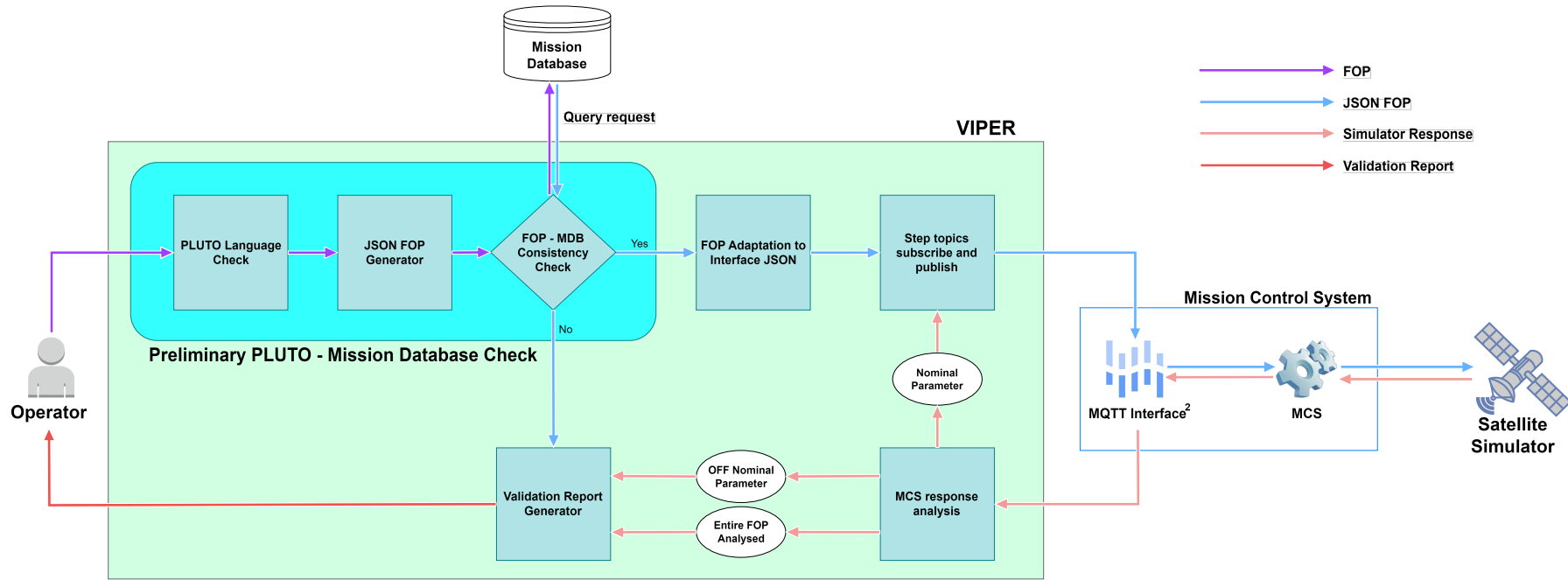


Figure 4.4: VIPER Architecture

The aforementioned interface enables the user to construct the TCs and TM provisions in a step-by-step manner, with the expectation of receiving the Satellite Simulator responses and notifications. These answers are then evaluated by comparing them against the “Accept” and “Complete” execution status expected, in the case of TCs, and the expected nominal values reported inside the original FOP, in the case of TM parameters.

The automated process of responses analysis creates a loop with the interface block responsible of prepare and elaborate the different FOP step contents: in presence of nominal received parameters, the block proceeds to the following procedure step evaluation. The described component is indicated in Figure 4.4 with the name “Step Topics Subscribe and Publish”. On the other hand, the exit from this loop is triggered by the occurrence of off-nominal parameters, which initiates the generation of the Validation Report and its subsequent transmission to the operator. This final step is automatically initiated in the exact same manner when the entire FOP has been executed and analysed.

4.2.1. VIPER Operational Modes

The correct functioning of VIPER is regulated by specific operational modes, which exploit its architecture and working principle to provide the operator with the most user-friendly validation tool. Six different modes have been identified, among which two modes ensure a comprehensive FOP analysis, while the remaining four are not sufficient to conclude the elaboration of the procedure, but need to be specifically combined with each other.

The **full** mode is the most general and it can be considered as the reference mode. The required inputs are the name of the procedure to validate, the name of the satellite on which the FOP will be tested and a specific folder destination to save the documents produced by VIPER as output of the validation process. With this information, the mode grants the execution of the FOP along the entire tool architecture, as previously explained. In case any WARNING will be identified by VIPER during this execution, the message string will be saved and reported in the final Validation Report without pausing the FOP analysis.

The **full_input** mode is essentially equal to the *full* one, both in terms of required inputs and principle of working. The main difference is bounded to the WARNING cases handling. Indeed, any warning message is not automatically saved and recorded, but is reported to the operator in the form of a display alert. Proceeding in this way, the FOP validation is stopped to wait for a user response regarding the continuation or forced termination of the process. Providing this possibility has been identified as an essential capability of VIPER as the tool itself is intended to validate procedures, but is not able

to replace human input in assessing whether having received a non-nominal value can be considered as a showstopper or as a non-compromising condition for validation.

The **pluto_mib** is the first of the four modes that needs to be combined with other modes. Its functioning principle is illustrated in Figure 4.5, where it is possible to appreciate that the mode objective is to execute the PLUTO-Mission Database Preliminary Check. Inserting the same inputs of the previous modes (FOP name, satellite name and folder path), the procedure PLUTO structure and its contents are verified and crosschecked with the MIB, which leads to the generation of a first preliminary Validation Report.

Such document will in fact indicates only if the PLUTO Standard has been respected and if any inconsistency with the database has arisen.

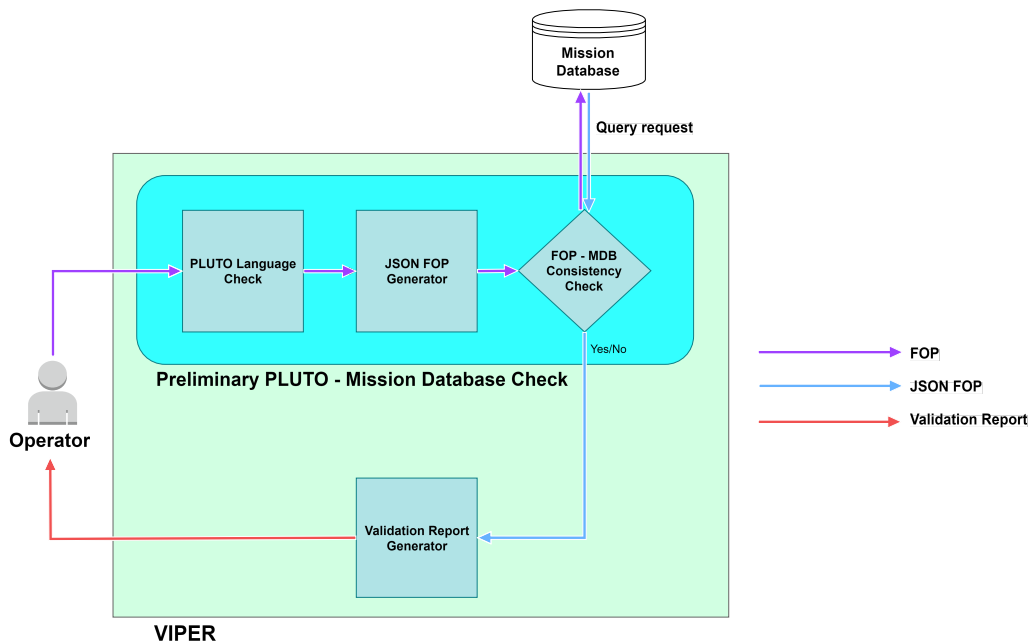


Figure 4.5: VIPER pluto_mib Mode Working Principle

Although this mode appears to be merely a restriction to the *full* mode, it has demonstrated its value during the design of the tool. This is because it allows the correctness of the procedure writing to be analysed without the use of the Satellite Simulator, a resource that is not always available when needed in complex spacecraft development contexts.

The remaining three modes are different evolutions of the **mcs_interface** operational mode. As previously stated, in order to complete the FOP validation process these modes must be coupled with *pluto_mib*, which actually works as the mode starting point. To facilitate the calling function, the same inputs requested from the other modes must be inserted. By exploiting the folder path indication, VIPER can in fact autonomously retrieve the JSON format FOP produced after the consistency check with the MIB and

proceed with the JSON adaptation required by the MCS interface. The principle of working is graphically reported in Figure 4.6. Once the Satellite Simulator data are obtained, the tool continues the analysis on the procedure as foreseen by the VIPER design.

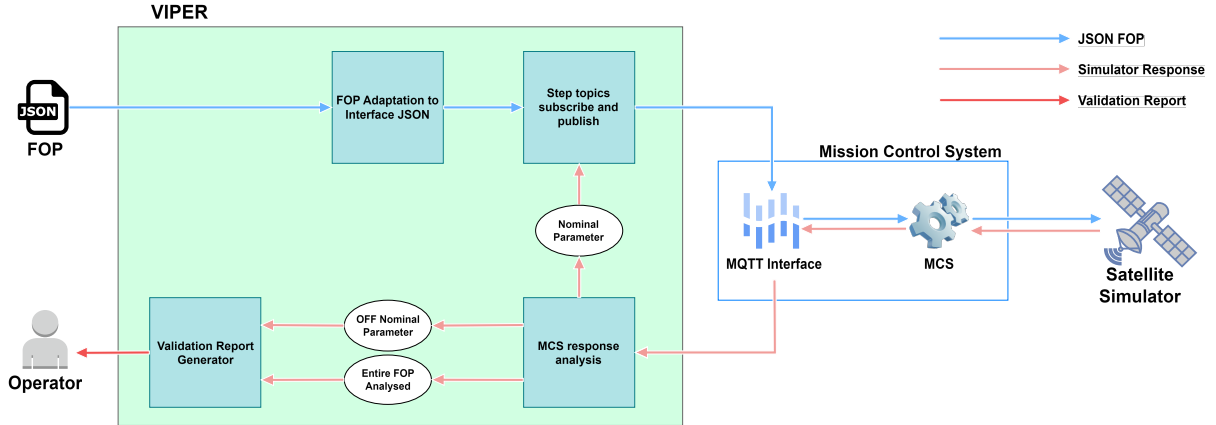


Figure 4.6: VIPER `mcs_interface` Mode Working Principle

At this stage it is worth to introduce the difference between the three interface modes, which consists only in different logics of MCS responses analysis and handling.

- `mcs_interface`**: this mode is representative of the nominal workflow, saving any possible WARNING message and reporting it in the Validation Report. The combination of `pluto_mib` and `mcs_interface` provides the same processing logic as the `full` mode.
- `mcs_interface_input`**: the operational principle is identical to `mcs_interface`, but together with the `pluto_mib` application, aims to reproduce the `full_input` mode results. Every possible WARNING message will be displayed to the operator, while pausing the validation.
- `mcs_interface_debug`**: the implementation of this mode has been thought to assist the operator during debugging actions, in case previous FOP validation processes have been completed with a negative outcome. The same properties of `mcs_interface_input` are implemented in this mode, adding an operator input each time a procedure step is completed during the FOP elaboration. This allows to easily identify which part of the procedure is interested by an error.

As specified by the operational requirement FOP-OPL-090 (Appendix A), the application of different operational modes for the same FOP, shall produce the same procedure validation status. Eventually, a summary of the VIPER operational modes names, together with a brief description of their functioning principles, is provided in Table 4.1.

Operational Mode Name	Description
pluto_mib	Preliminary PLUTO-MIB consistency check
mcs_interface	FOP Execution with MCS/Satellite Simulator and Validation Report generation
mcs_interface_input	FOP Execution with MCS/Satellite Simulator and Validation Report generation, requiring operator input for any warning message
mcs_interface_debug	FOP Execution with MCS/Satellite Simulator and Validation Report generation, requiring operator inputs at every step evaluation and for any warning message
full	Preliminary PLUTO-MIB Consistency Check, FOP Execution with MCS/Satellite Simulator and Validation Report generation
full_input	Preliminary PLUTO-MIB Consistency Check, FOP Execution with MCS/Satellite Simulator and Validation Report generation, requiring operator input for any warning message

Table 4.1: VIPER Operational Modes Summary

5 | VIPER Implementation

This chapter presents a more detailed overview of the VIPER architecture, analysing each part of the tool structure in terms of its functionality and the rationale behind the development choices made. In order to conduct this analysis, it is necessary to start from the widest possible point of view, namely investigating how the functional structure of the tool has been designed. The operational framework is constructed upon the execution of a main script, which, upon analysing the input data, is capable of associating the various required operational modes with a series of functions defined within the tool. This process requires that the necessary inputs, already anticipated during the description of VIPER operational modes, must be provided in a Python terminal following a specific logic. The required inputs are:

- The **main script name**, indicated as `Final_Main.py`.
- The **operational mode**.
- The **satellite name**, expressed between quotation marks and anticipated by the command `-sn`.
- The **FOP name**, expressed in `.pluto` format and anticipated by the command `-fn`.
- The **FOP destination path**, used to save the tool outcomes in a specific folder. This path must be anticipated by the command `-fp`: such information can be associated with a default value to avoid its continuous insertion. However, it can be modified at any time.

An illustrative example of the appropriate command to insert into the Python terminal considering the default path destination is hereafter proposed:

```
python Final_main.py full -sn "SAT_01" -fn VIPER_FOP.pluto
```

Listing 5.1: Example of VIPER Python Terminal Command

It is only when inputs are expressed in this manner that they can be analysed by VIPER. Consequently, all of them are immediately checked. If even a single input is reported

incorrectly, the tool will trigger a flag, which will in turn result in the generation of a negative validation report and the consequent interruption of the validation of the procedure. Conversely, VIPER begins to process the FOP.

The Python functions depicted in Figure 5.1 have been created with the objective of ensuring the correct evaluation of the procedure and have been made accessible to the main script, which effectively utilises them in accordance with the required specifications.

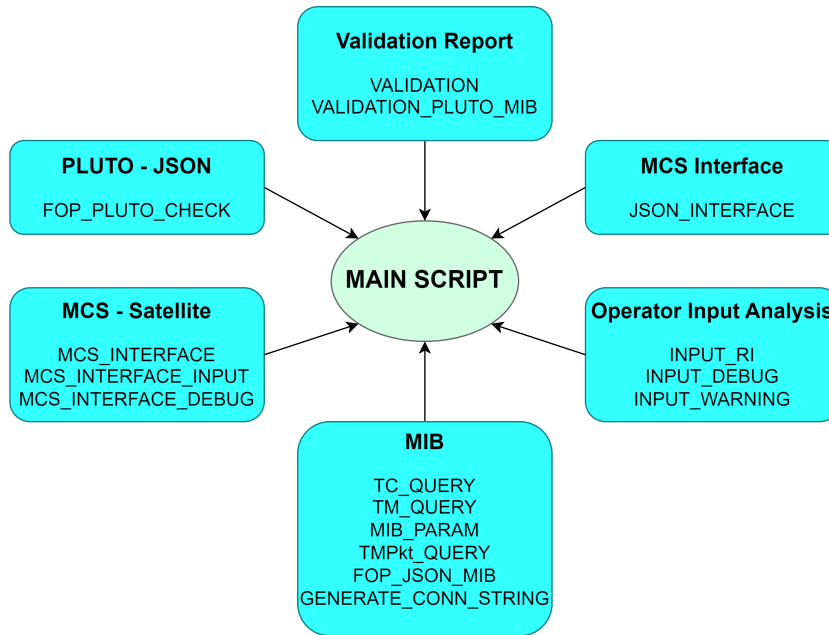


Figure 5.1: VIPER Python Functions

The subsequent sections will address each step of the tool operation in detail, outlining the effective execution of the identified procedures, the outputs of these phases, and the potential alternative approaches to achieving the objective that may be defined by requests for different operational modes.

5.1. PLUTO - FOP Preliminary Check

The PLUTO FOP Preliminary Check is the first VIPER block that directly treats the procedure contents. Inside the tool algorithm, the analysis of the PLUTO structure is entirely executed through the initial part of the function `FOP_PLUTO_CHECK`. Here, the FOP is opened in its `.pluto` format and analysed element per element, considering every procedure line as an element. A structural check on the PLUTO constructs can be performed by identifying the adopted standard PLUTO keywords, as previously discussed in section 3.4. Whenever needed, the element line is split into single words, facilitating a more precise comparison with the adopted PLUTO Standard. This strategy is typically

adopted when the correct line has been identified and there is the need to verify the consistency of a word or a number, such as to confirm that it has been inserted in the correct position along the line or, more widely, inside the entire FOP. Performing such analysis enables VIPER to identify two types of errors: those related to an incorrect PLUTO FOP general structure and those involving mistakes in the construction of precise PLUTO commands. In more detail, the first category of errors includes:

- "Procedure" inconsistency
- "Preconditions" inconsistency
- "Main" inconsistency
- "Watchdog" inconsistency

Complementarily, the verification on the specific commands regards the following possible errors:

- "With" inconsistency
- "If" inconsistency
- "If and Then" inconsistency
- "Step" inconsistency
- "Record" inconsistency
- "Log" inconsistency
- "Initiate and confirm" inconsistency
- "End" inconsistency

In the event that any of the aforementioned errors are identified, a specific dictionary will be populated with a flag indication and an error message, which will then be provided to the operator in the Validation Report.

5.2. FOP Transcription in a Structured JSON File

The decision to translate the PLUTO FOP into a JSON file is supported by the numerous advantages of this format. Primarily, it is characterised by excellent readability, both for humans and for machines, due to its text format nature. Furthermore, the interoperability of this kind of file allows it to be interfaced with a vast amount of software and tools developed in different programming languages. The JSON format is capable of represent-

ing complex structures, such as nested objects and arrays, through the use of multiple data types, including strings, numbers, Booleans, and so forth. The PLUTO procedural language is similarly structured around an object-oriented architecture, which enables the transcription of an entire procedure in a JSON file.

The creation of such a file is possible through the application of the VIPER function `FOP_PLUTO_CHECK`, which structures the JSON FOP file as a list of dictionaries. Each of these dictionaries represents one step of the PLUTO procedure. The hierarchical order of the steps is maintained by processing them one at a time with a *for* loop, whereby the new item is appended to the FOP list whenever the step transcription is concluded. Each dictionary contains the same keys, which are associated with the specific properties of the corresponding step:

Step Dictionary Key	Key Contents
step_name	String
step_ID	Integer
step_if	Dictionary
delay	Integer
TC	Dictionary
TM	Dictionary
Req_Inputs	Dictionary

Table 5.1: JSON FOP Step Dictionary

The only keys that can be guaranteed to have a secure association every time are the ones referring to the step name and the step ID number, while the others can be left as empty keys if such components are not found in the analysed step. The *delay* indicates a possible period of time to wait before the execution of the step itself and it must be expressed in seconds. The *step_if* dictionary represents an imposed condition that, if verified, will prevent the step from being executed. The keys of the associated dictionary refer to the string construction of this condition, so reporting the subject, the object and the related consequences. A more detailed analysis of this dictionary keys is reported in Table 5.6.

The *Req_Inputs* is a key filled with a dictionary only in the first step of each FOP, to report the required inputs of the procedure with the information available in Table 5.2. The keys are associated with strings, which include the satellite name, the name of the

required input and the string containing all the required inputs to display to the operator, together with the *ask_user* expression that regulates this function. Only the *array_inputs* is designed to accept a list element containing the array components.

Req_Inputs Dictionary Key	Key Contents
satellite_name	String
array_input	List
single_input	String
input_log	String
ask_user	String, Yes or " "

Table 5.2: JSON FOP Required Inputs Dictionary

The telecommand and telemetry keys are instead more complex than the other keys, due to the amount of information they have to deal with. The TC dictionary is composed of the keys reported in Table 5.3, and in the event that no telecommand is present in the step under evaluation, an empty dictionary is returned.

TC Dictionary Key	Key Contents
tc_name	String
type	3 Digit Integer
subtype	3 Digit Integer
apid	3 Digit Integer
cont_TC	Integer
delay_TC	Integer
exe_time	String
ackFlag	Integer
ask_user_var	String, Yes or " "
var_name	String
inform	String, Yes or " "
inform_sent	String
param_number	Integer
parameters	Dictionary

Table 5.3: JSON FOP TC Dictionary

The first indication serve to uniquely define the telecommand, namely the TC name and the type, subtype and apid numbers. The time-related keys are inserted to track the information on a possible amount of time to wait before sending the TC to the MCS (`delay_TC`) or if it is needed to specify the telecommand execution time. In this case the ISO format shall be reported as a string. In case an `ackFlag` different from the default value of 11 is requested, the corresponding key must be populated with the desired number. The `ask_user_var` indicates whether the operator must be consulted to define a new variable through an input, whose name is reported in `var_name`. This event typically occurs when a TC parameter that cannot be predicted in advance must be inserted to complete the FOP.

It should be noted that not all TCs require the explicit addition of specific parameters in the TC send. However, when such parameters are requested, the JSON file provides a separate list of dictionaries which enables the reporting of all their details, in addition to a specific entry indicating the required parameters number.

TC Parameters Dictionary Key	Key Contents
<code>parameter_name</code>	String
<code>parameter_value</code>	String
<code>format</code>	Boolean
<code>raw_eng</code>	String

Table 5.4: JSON FOP TC Parameters Dictionary

Each parameter is associated with a dedicated dictionary, which specifies the parameter name, the associated value of reference and an indication on the format of the parameter itself. This format is either raw or engineering, and is indicated by the `format` key, which is activated to "True" only in case of RAW parameter.

The following proposal concerns the telemetry associated dictionary. In a manner analogous to the TC case, if no TM parameter or TM packet request is present within the step under evaluation, the telecommand key will be associated with an empty list. Indeed, a single step may contain more than one provision request, and thus the JSON file has been designed to structure the telemetry key as a list of dictionaries. The telemetry parameter is defined by four pieces of information: its name, the type, the subtype and the APID numbers. The TM packet is similarly defined by two pieces of information: the PI1 and PI2 indicators. These must be inserted in the corresponding keys, such as the mandatory "YES" indicator for the key `TM_pkt`. The logic of the delay amount of time specified

for the TC case is repeated here for the telemetry request. FOPs typically associate the nominal value expected with telemetry parameters requests, either indicated as a single value or contained inside an interval. One of the primary objectives of the VIPER system was to automate the process of these telemetry evaluations, while maintaining the ability to query the operator regarding the values obtained.

TM Dictionary Key	Key Contents
tm_name	String
tm_pkt	String, Yes or " "
type	3 Digit Integer
subtype	3 Digit Integer
apid	3 Digit Integer
PI1	3 Digit Integer
PI2	3 Digit Integer
delay_TM	Integer
log	String
ask_user	String, Yes or " "
event	Dictionary

Table 5.5: JSON FOP TM Dictionary

The latter has been incorporated into the TM dictionary through the *log* and *ask_user* keys, while the autonomous check is registered under the key *event*.

It is important to note that in the PLUTO FOP, whenever a TM parameter does not respect the nominal values indications, a raise event action is planned. This terminology is also used in the JSON file, where the name of the event at issue must be specified and the condition regulated by the *If* loop in PLUTO shall form a new dictionary associated in turn to the *IF* key. This dictionary is structured in an identical manner to the one presented as a key in the general step structure in JSON format, which was introduced at the beginning of this paragraph. The *equal*, *less* and *greater* keys are used to handle the presence of the *if_loop* condition logic operators. The reference parameter to be compared with is indicated, respectively, as the condition object (*if_obj*), the upper bound value (*up_b*) or the lower bound value (*low_b*). The remaining keys pertain to the string that is to be displayed to the operator via the use of the *inform*, *inform_sent* and *ask_user* functions.

TM Event Dictionary Key	Key Contents
event_name	String
IF	Dictionary
TM Event IF Dictionary Key	Key Contents
If_sub	String
equal	String, No or " "
If_obj	String
Low_b	Number
Up_b	Number
less	String, Yes or " "
greater	String, Yes or " "
inform	String, Yes or " "
inform_sent	String
ask_user	String, Yes or " "
next_step	Integer

Table 5.6: JSON FOP TM Event and If Dictionary

The indication regarding the next step concerns the ID of the next FOP step to be executed in the event that the retrieved TM parameter is in an off-nominal condition. Adopting this file structure allows for the transcription of the entire PLUTO FOP in a JSON format. This is because the design of the file evolved after the definition of the new Argotec PLUTO Standard.

Furthermore, while allocating the respective values to the different step dictionary keys, it is possible to perform a further verification on the initial FOP contents correctness. The same logic employed in the PLUTO Consistency Check to identify specific cases and words to examine is replicated during this analysis. However, in this instance, the reference points are not the general PLUTO rules, but rather the specific Argotec Standard. This approach ensures the insertion of accurate data that can be effectively processed by VIPER.

Similarly to the procedure employed during the PLUTO Preliminary Check, in the event that an error is identified during the generation of the FOP JSON file, a specific dictionary is populated with the flag indication and the error message, which is then provided to the operator in the validation report.

5.3. MIB Consistency Check

Although the controls performed on the initial procedure by means of the PLUTO Preliminary Check and the formation of the FOP in JSON format are highly accurate, it is not possible for them to independently verify the veracity of certain data due to the inherent nature of VIPER design. In particular, the data that require more precise verification are the TCs names and parameters numbers, as well as the names of the telemetry single parameters and TM packets. This action could be considered non-essential in the validation process given that, if properly designed, VIPER should not be able to execute procedures containing TCs and TMs that cannot be processed by the MCS.

However, in order to provide a design as robust as possible and to assist the human operator to the fullest extent possible, these cross-check functions were implemented.

As anticipated, the tool requires interfacing with another unit of the Argotec FOS in order to perform this verification, namely the Mission Database. The MIB is, in fact, the set of files that characterise the properties and rules of the satellite telemetry and telecommands. It is imperative that every TM and TC intended for use during a specific space mission be entered in this DB. Retrieving the requested information directly from the MIB is a necessity, and implementing SQL Query Requests appears to be the optimal strategy.

A query is a request sent to a database in order to retrieve, insert, update, and delete data within the database itself. The Structured Query Language is the most common language for managing and manipulating relational databases. Adopting this language allows the use of already developed Python libraries, which have been designed to optimise the DB interactions, such as “SQLAlchemy”.

In order to create a query correctly, it is necessary for VIPER to generate a connection string. This is a string that provides the necessary details to connect to the DB, including the database type, username, password, host, and database name. This generation is carried out by the tool function `GENERATE_CONN_STRING`. Once the connection string has been obtained, it can be exploited by a specific Python library in order to initialise a SQLAlchemy engine. This serves as the core interface to the database specified by the connection string.

At this point it is possible to produce the effective request against the MIB, in order to retrieve the data for the cross-check. In the VIPER case is necessary to develop three different query requests with three distinct functions: still, all of them begins with the creation of the connection string and of the SQLAlchemy engine.

- **TC_QUERY**: to obtain the TC name and the number of required TC parameters.
- **TM_QUERY**: to obtain the TM parameter name only.
- **TMPkt_QUERY**: to obtain the TM packet name only.

The telecommands are defined in the PLUTO FOP solely by their type-subtype-apid combination. However, to be processed by the MCS software, it is necessary to report the TC name. Furthermore, with this function VIPER is capable of verifying that the number of TC parameters inserted in the initial procedure is consistent with the MIB indication. Such a check allows to verify the correctness of the type, subtype and APID numbers, because if no match is found in the database, an error message is sent to the operator, interrupting the validation process.

Regarding the TM parameter, as for the TCs, it is necessary to have the correct name to process its provision request with the MCS software. In the telecommand case there is no need to verify the TC name because it is directly retrieved from the Mission Database, while a comparison must be performed between the TM parameter name inserted in the FOP and the one actually present in the MIB.

The TM packet instance is more similar to the TC query, because the initial procedure does not give the packet name directly, but its type-subtype-apid-PI1-PI2 combination. For both TM functions, if no telemetry name match is found, the validation process is stopped. The entire sequence of functions that is acting in this FOP-MIB Consistency Check is illustrated in Figure 5.2.

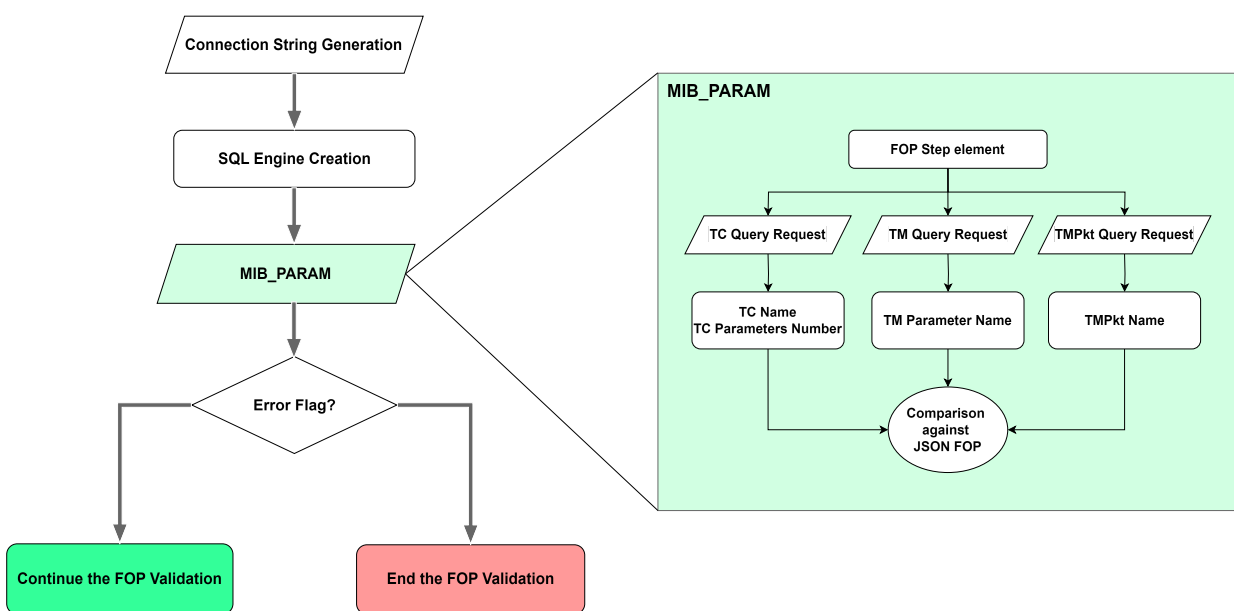


Figure 5.2: FOP MIB Consistency Check Functioning Logic

It is clear that the function responsible for extracting the data from the MIB and comparing them with the FOP is `MIB_PARAM`. The external function `FOP_JSON_MIB` creates a general query request to load the Mission Database, initiate the `MIB_PARAM` function and handle its output. It is here that the correct and consistent data are inserted into the final FOP JSON file or, alternatively, the FOP evaluation is interrupted, triggering the generation of the Validation Report.

5.4. VIPER - Mission Control System Interface

Once the FOP has been fully transcribed into the JSON file and it has been verified that all TCs and TMs are consistent with the MIB, the VIPER design provides for procedure elaboration by the MCS and the Satellite Simulator. As explained in section 4.2, the tool uses an interface already developed by Argotec. The messaging protocol used is Message Queuing Telemetry Transport (MQTT), one of the most commonly used when two different devices need to communicate with each other.

As the interface design was finalised at the time of VIPER development, the FOP has to be adapted to the specified input format of the interface, which only concerns the telecommand and telemetry provisions. To perform this adaptation of the JSON FOP, the Python function `JSON_INTERFACE` has been created and its application leads to a reorganisation of the procedure into a new list, characterised by an empty sublist associated with each FOP step. The latter is populated with a dictionary each time the function recognises a TC send, a TM parameter request or a TM packet provision within the first JSON FOP. A single list can contain several dictionaries, as the PLUTO standard permits the presence of multiple TM requests and the combination of a TC send and a TM provision within the same step.

The two dictionaries dedicated to the telemetries are relatively simple and have only two keys, as shown in Table 5.7: the name of the parameter or of the packet shall be associated to the dedicated key, while the possible options for the *itemType* are limited to the TM or TM packet indications.

A more complex dictionary is instead dedicated to the TCs: its keys are appreciable in Table 5.7 too.

TM Interface Dictionary Key	Key Contents
itemType	String, "TM" or "TMPkt"
itemName	String
TC Interface Dictionary Key	Key Contents
requestID	UUID
tcName	String
tcParameter	List
tcOption	Dictionary

Table 5.7: JSON Interface TM and TC Dictionaries

The telecommand ID is an identification code expressed as a randomly generated UUID hexadecimal string, which serves to uniquely identify the TC send request. In addition, the name obtained from the MIB is associated to *tcName*. In the event that one or more TC parameters are present, the corresponding key must be filled with a list of dictionaries, containing as many elements as the number of parameters. The specific keys are shown in Table 5.8, and they concern the parameter name, value and specification on the ENG or RAW format of it.

tcParameter Dictionary Key	Key Contents
name	String
value	String
forceRaw	Boolean
rawType	String
tcOption Dictionary Key	Key Contents
ackFlag	Integer
MAPID	3 Digits Integer
executiontime	ISO time
Delay	Integer

Table 5.8: JSON Interface TC Parameter and TC Option Dictionaries

An interface rule common to both TC and TM dictionaries stipulates that no empty key must be inserted in the dictionary. Therefore, after their value association, VIPER autonomously searches for empty keys and, in the event of their identification, deletes

them. Following the completion of the second JSON FOP tailored for the MCS interface, it was necessary to gain a deeper understanding of the connection at a more fundamental level in order to exploit it and define the new Python functions that regulate the FOP execution with the Satellite Simulator.

The MQTT communication is broker-based, which necessitates that a server, the broker, receives all messages from users, referred to as clients. Each client is permitted to submit requests to the broker, who will then route them to the appropriate destination. This dialogue is possible through the act of publishing and subscribing, as illustrated in Figure 5.3. [4] By publishing a message to a specific topic, a client can effectively send its message to the broker and, by subscribing to the same topic, all the clients receive the messages directly from the broker. The latter process is defined as routing. In the VIPER specific case, defined in Figure 5.4, the tool publishes the TC send and the TM provision requests that will be received by the broker, which is a component inside the MCS software.

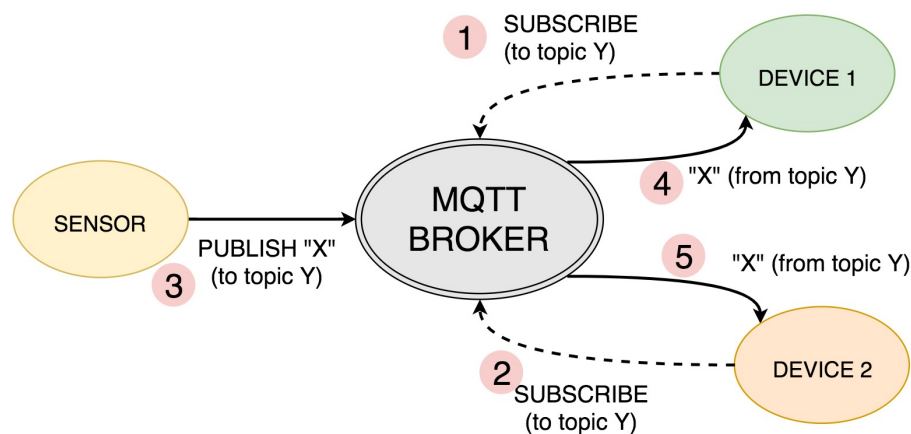


Figure 5.3: MQTT Broker - Client Connection [4]

At this point, the software forwards this request to the Satellite Simulator, in turn directly linked to the MCS Software via its TCP socket connection. [27] This will not be analysed in the context of this thesis. Once the FlatSat has elaborated the requests received, it returns the associated answers to the MCS Software, which will then publish the response into the Broker. As the VIPER tool not only publishes requests with a specific topic but also subscribes to the same topic, it will receive the Satellite Simulator responses directly from the MCS Software through the Broker.

It is evident that the process is tailored to each specific request, such as the associated response times. To facilitate the interaction of the tool with the MCS and the Satellite Simulator, VIPER presents the implementation of a threading condition to oversee the

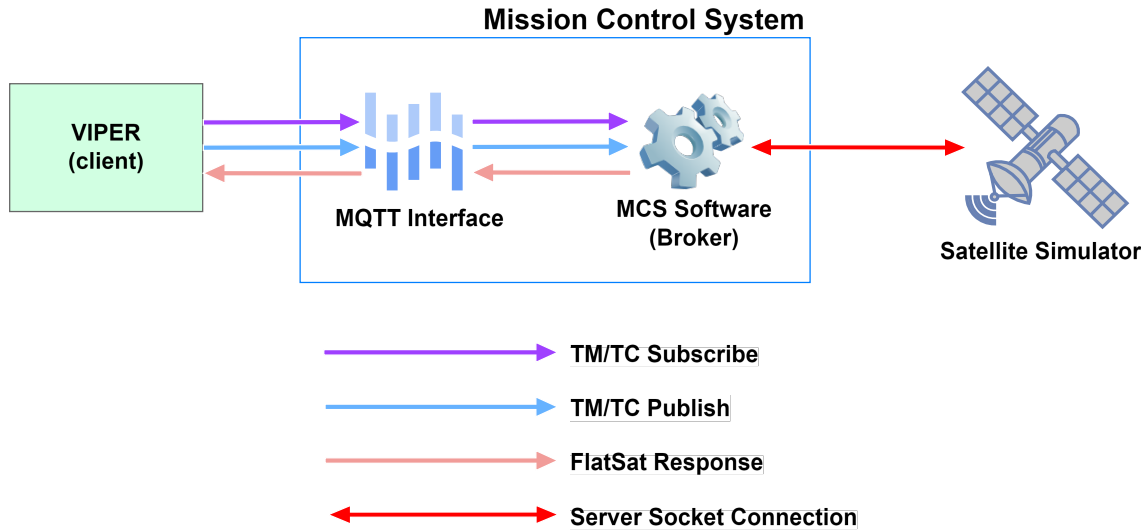


Figure 5.4: VIPER-MCS-Satellite Interfaces

asynchronous communication and data processing. A condition variable, in fact, allows one or more threads to await notification from another thread. One possibility is to assign the thread alert to callbacks, namely specific functions called up by an operating system for the purpose of handling particular events. This process enables a lower-level program to call up a function, or service, defined at a higher level. [8]

The VIPER design aims to create callback functions to handle responses to TC and TM requests such that their execution would not terminate the primary thread, i.e. the `MCS_INTERFACE` main function. More specifically, these functions are triggered each time the satellite response is processed by the MCS software. Once received the answer, the callback processes the incoming message by storing all the information obtained by the MCS in order to make them available and consultable even after the loop with the simulator has ended. Once completed this task, the function has to notify the waiting thread. The above mentioned structured approach ensures that the script can handle asynchronous data reception and processing reliably, while effectively managing potential timeouts and errors through synchronised condition checks and notifications.

Once the callback function has generated the notification, it is the responsibility of VIPER to close the listening loop and analyse the responses obtained by comparing them with the data entered in the initial FOP before unsubscribing to the specific topic. The final stage of the procedure is the storage of the outputs obtained and the closure of the MQTT loop. The Python function, specifically designed to handle this interface and execute the FOP, is `MCS_INTERFACE`. The following steps are taken within this function for each step in order to grant the correct elaboration and execution of the procedure:

1. Connection to the MQTT interface and initialisation of the step loop.
2. Calling of the threading condition.
3. Callbacks functions definition.
4. MQTT topic creation.
5. Callback addition, topic subscribe, publish of the message, condition variable wait loop imposition.
6. Callback function notify, analysis of the Satellite Simulator responses, MQTT topic unsubscribe.
7. Reorganisation of the outputs and MQTT connection closure.

5.5. Analysis and Handling of Satellite Simulator Responses

Focusing on the Satellite Simulator responses elaboration, apposite analysis and handling logics must be implemented in the VIPER design. This process should be executed in a complete automated manner through the calling of the `MCS_INTERFACE` functions. As shown in Figure 5.1, the Python functions dedicated to the VIPER FOP execution with the MCS are:

- `MCS_INTERFACE`
- `MCS_INTERFACE_INPUT`
- `MCS_INTERFACE_DEBUG`

All of these guarantee the processing of the procedure as described in section 5.4 at a general level, differing from each other precisely in how the simulator responses are handled. The default function `MCS_INTERFACE` initiates the processing of these data once the JSON files containing the system responses have been collected.

The expected dictionaries are reported in Table 5.9.

The information related to the TC concerns the execution stages of the telecommand itself, which consists of Pre-Transmission-Verification (PTV), Accept and Complete. The "isFinal" key is associated with a Boolean value correlated to the last verification stage update indication, while the "success" one reports a true value if the telecommand has been successful.

TC Response Dictionary Key	Key Contents
stage	String
isFinal	Boolean
success	Boolean
TM Parameter Response Dictionary Key	Key Contents
rawValue	Number
engValue	String
isValid	Boolean
timestamp	ISO time
TM Packetr Response Dictionary Key	Key Contents
timestamp	ISO time

Table 5.9: JSON Interface Responses from MCS

Conversely, the TM dictionary is characterised by information related to the engineering, or raw, value of the parameter, together with the *isValid* indication. This is imposed as "True" if the TM value passes its validity. The telemetry arrival time is reported as a timestamp and is the only common key of both the TM parameter and packet. In fact, the reception of this data triggers the condition variable notify from the callback function. The telecommand-related callback closes the listening loop whenever the *success* key is set to "True". Given that a single or multiple execution stage responses are generated for each TC stage, a list containing all the generated dictionaries is automatically created and provided to the operator.

Upon receipt of the satellite response, VIPER retrieves the corresponding FOP data to compare them. The initial verification, as depicted in Figure 5.5a, is employed to ascertain the congruence of the two data sets. In the event of an inconsistency or discrepancy, the FOP validation is promptly terminated.

Alternatively, a control on the response format is executed to analyse the data as a RAW value or in its ENG nature. [2] If the two responses are exactly the same, the FOP validation must continue going through the next procedure step, while if they differ a warning alert string is inserted in the validation report. The generation of this warning implies the continuation of the FOP execution within the MCS_INTERFACE.

A comparable analysis architecture has been devised for the MCS_INTERFACE_INPUT function, and its schematic representation is presented in Figure 5.5b. The process is identical to that described in Figure 5.5a, with the exception of instances where the response does not align with the data indicated in the procedure.

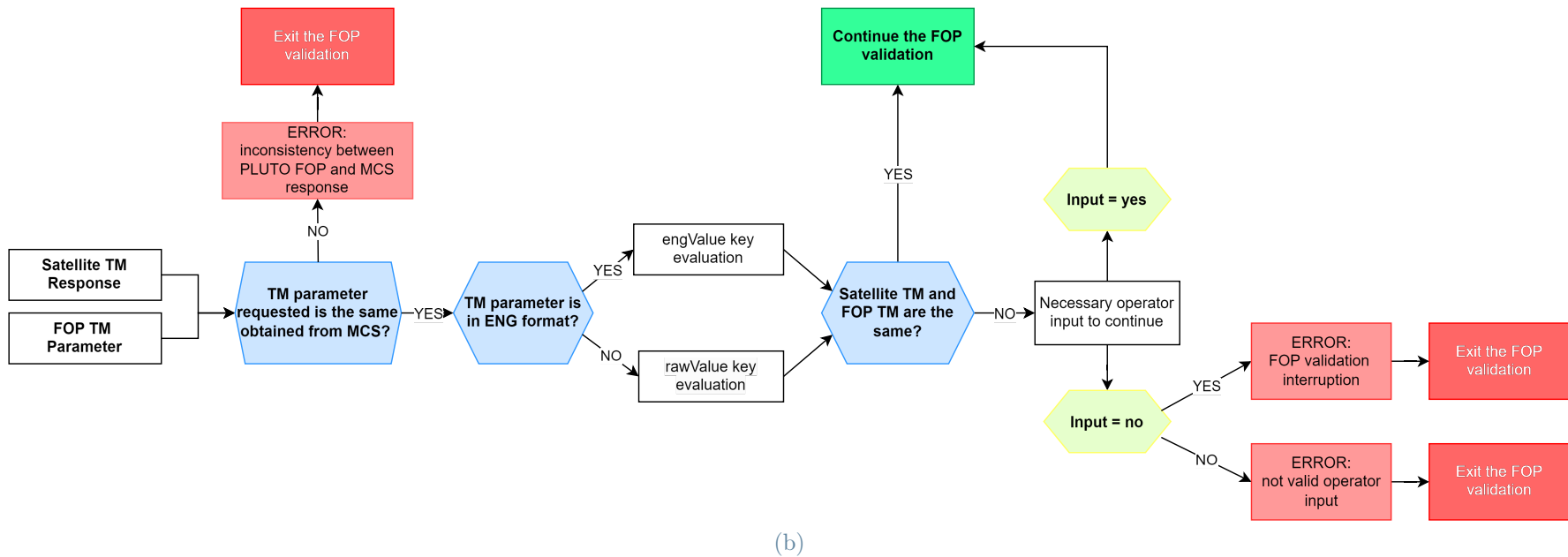
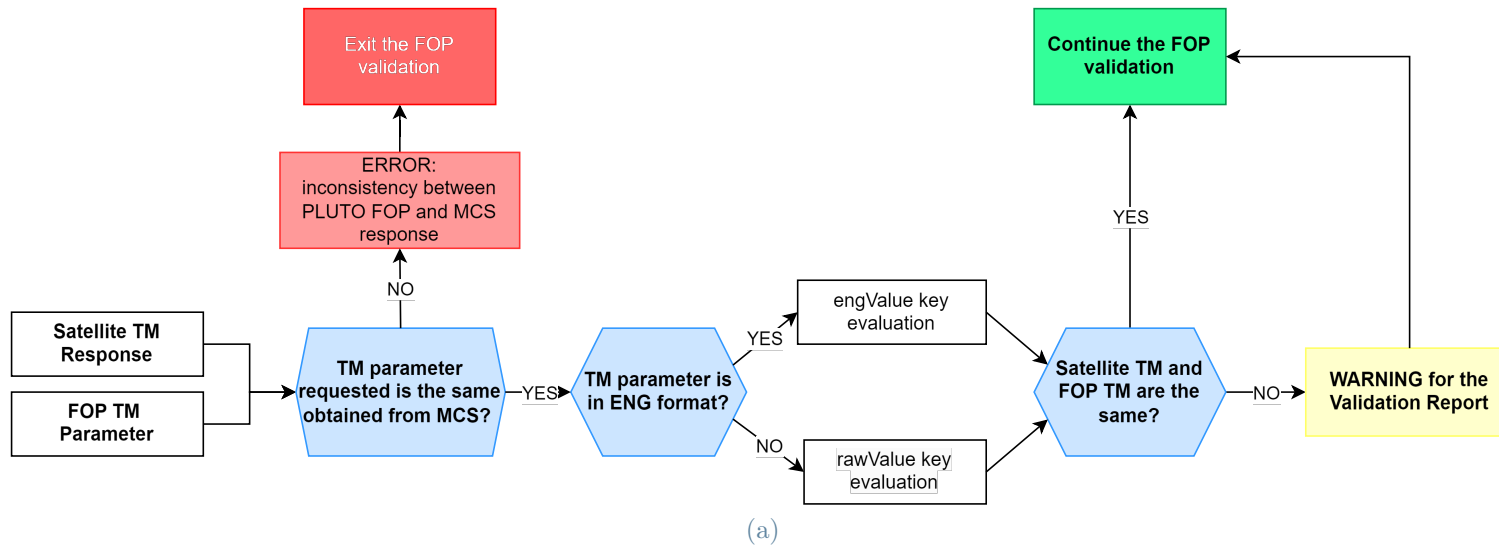


Figure 5.5: Analysis of MCS Interface Responses Handling logic: (a) mcs_interface function, (b) mcs_interface_input function

This function in fact foresees an operator input to continue, or interrupt, the validation process, as foreseen by chapter 4. To facilitate user input, a message indicating the warning encountered is displayed, accompanied by a direct question: "Do you want to continue the FOP validation?". In case an affirmative command is inserted, VIPER resumes its process of automatic execution of the procedure; otherwise, it interrupts the FOP execution by generating the corresponding Validation Report.

Eventually, the `MCS_INTERFACE_DEBUG` function is designed to operate with the same response analysis logic of `MCS_INTERFACE_INPUT` but will also request the operator input at the end of each procedural step, as specified in subsection 4.2.1.

5.6. Validation Report Generation

The generation of the Validation Report is the final step of the VIPER procedure validation process and it is executed both in case of previous smooth elaboration or if any error has risen. This document, provided in *.txt* format, is fundamental to determine the effective status of the procedure testing outcome.

The designed layout of the Validation Report can be subdivided into three sections: the FOP validation test details, the validation session report and the status check report.

An example of the VIPER Validation Report document layout is proposed in Listing 5.2.

The initial block of the report is the one characterised by the test details, where all the provided information must be sufficient to identify both the evaluated procedure and the condition in which the test has been performed.

- **FOP Name:** the PLUTO procedure name.
- **Satellite Name:** the ID related to the satellite adopted for the test.
- **FOP Validation Test Date:** the start time of the test, reported in ISO 8601 standard.
- **FOP Required Input:** the conditions required by the procedure to be efficiently executed.

In the event that the procedure successfully concludes the Preliminary Checks and its successive analysis with the MCS and the Satellite Simulator begins, a validation session report is produced. The aim is to provide a detailed account of the requests made through VIPER interpretation of the FOP and the answers obtained from FlatSat.

```

FOP VALIDATION REPORT
-----
FOP Name: "V_TMTC"
FOP Tested on Satellite: SAT_01
FOP Validation Test Date: 2024-05-21T16:50:53.800736
FOP Required Input: RX_configuration
-----

Step 1: Started @ 16:50:27
OPERATOR: []
SATELLITE: No TM or TC request to send @ step 1
Finished @ 16:50:27
-----

Step 2: Started @ 16:50:27
OPERATOR: [{'requestID': 'f587e247-3b7e-4466-a619-412af6f9c8f4', 'tcName': 'RADIO_CONFIGURATION', '
tcParameter': [{'name': 'PARAM_1_ID', 'value': '2', 'forceRaw': 'true', 'rawType': 'US'}], '
tcOption': {'MAPID': 'YYY'}}]
SATELLITE: [{'stage': 'PTV', 'isFinal': False, 'success': True}, {'stage': 'ACCEPT', 'isFinal':
False, 'success': True}]
Step 2: Finished @ 16:50:47
-----

Step 3: Started @ 16:50:47
OPERATOR: [{'Delay': '5'}]
VIPER and MCS are waiting the time indicated in the FOP step (5 seconds)
Step 3: Finished @ 16:50:52
-----

Step 4: Started @ 16:50:52
OPERATOR: [{'itemType': 'TM', 'itemName': 'TMTC_01'}]
SATELLITE: {'rawValue': 1, 'isValid': True, 'timestamp': '2023-01-01T02:33:54.503000', 'engValue':
'OPERATIONAL_STATUS'}
Step 4: Finished @ 16:50:52
-----

Step 5: Started @ 16:50:52
OPERATOR: [{'itemType': 'TM', 'itemName': 'TMTC_MODE'}]
SATELLITE: {'rawValue': 3, 'isValid': True, 'timestamp': '2023-01-01T02:24:26.222000', 'engValue':
'TX_RX_CONFIG'}
Step 5: Finished @ 16:50:52
-----

Step 6: Started @ 16:50:52
OPERATOR: [{'itemType': 'TM', 'itemName': 'OPMODE'}]
SATELLITE: {'rawValue': 0, 'isValid': True, 'timestamp': '2023-01-01T02:33:50.250000', 'engValue':
'SAFE_MODE'}
Step 6: Finished @ 16:50:53
-----

FOP PLUTO Preliminary Check: PASSED
-----

FOP - MIB Consistency Check: PASSED
-----

FOP Analysis with MCS: COMPLETED
-----

FOP Contents Verification: PASSED
-----

FOP Validation Status: VALIDATED
-----

```

Listing 5.2: VIPER Validation Report Layout

This part is divided into sections to reflect the steps involved in the process. Each section is comprised of the step number coupled with the execution start time, the operator request data sent to the MCS, the satellite response data received and again the step number, but coupled with the execution end time. The generation of this block can be disregarded through the use of a specific operator indication, a feature designed for VIPER users who are not interested in the step-by-step evaluation of the test but rather only wish to analyse the final FOP validation status check.

The final status check report is composed of five different sections, which serve to identify the condition of the main VIPER verifications. According to the tool design, these parameters are:

1. **FOP PLUTO Preliminary Check:** considered as “Passed” if the procedure is written accordingly to the PLUTO Standard.
2. **FOP-MIB Consistency Check:** considered as “Passed” if the previous check has been successful and if the FOP contents are consistent with the Mission Database.
3. **FOP Analysis with MCS:** considered as “Completed” if the previous checks have been successful and if the entire procedure has been processed with the MCS and the Satellite Simulator.
4. **FOP Contents Verification:** considered as “Passed” if the previous checks have been successful and if the procedure content appears to be correct and consistent.
5. **FOP Validation Status:** considered as “Validated” if the previous checks have been successful and if the FOP execution did not highlight any error or warning.

Since the detection of an error implies the forced interruption of the FOP execution, the corresponding check status on the Validation Report will be associated with a “Not Passed” status. Subsequently, all the next verifications will report the “Not Tested” mark. The success of the four initial verifications is a necessary condition for the procedure to be considered as validated, but it is not sufficient since the FOP can be correctly written, entirely executed and still presents one or more warning conditions.

It is worth noting that, in case of error detection, VIPER is able to trace the cause of it and this information must be inserted in the Validation Report.

As it is possible to appreciate in Listing 5.3, the error diagnostic is present inside the status check report and the step reporting the error is explicitly shown even when the report contracted form is generated.

```
FOP VALIDATION REPORT
-----
FOP Name: "V_SCHEDULE_07"
FOP Tested on Satellite: SAT_01
FOP Validation Test Date: 2024-05-21T16:28:38.967980
FOP Required Input: NOT_SAFE_MODE, OPERATIONAL_STATUS, TX_RX_CONFIGURATION
-----
Step 4: Started @ 16:28:33
OPERATOR: [{'itemType': 'TM', 'itemName': 'OPMODE'}]
SATELLITE: {'rawValue': 3, 'isValid': True, 'timestamp': '2023-01-01T01:51:23.433000', 'engValue':
  'COMMUNICATION_MODE'}
-----
FOP PLUTO Preliminary Check: PASSED
-----
FOP - MIB Consistency Check: PASSED
-----
FOP Analysis with MCS: NOT COMPLETED
-----
FOP Contents Verification: NOT PASSED

USER INPUT REPORT
WARNING: OFF-nominal TM in OPMODE. Expected: SAFE_MODE, Obtained: COMMUNICATION_MODE. USER INPUT:
  Not valid user input
ERROR: Not valid user input. FOP Validation Interruption
-----
FOP Validation Status: NOT VALIDATED
```

Listing 5.3: VIPER Validation Report with ERROR Detection

6 | VIPER Test and Validation

6.1. Test Description and Passing Criteria

Once completed the tool design, VIPER must undergo a validation process in order to verify its functionalities and ascertain that its objectives are effectively met. The test campaign planned to prove the functionality of the tool has been designed with the following main objectives:

- Verify that all VIPER requirements are met,
- Test the VIPER capabilities on a relevant number of nominal case studies, proving that it can effectively validate the FOPs already validated by the FCT and, on the contrary, handle possible FOP errors and warnings according to the tool design and requirements.

Concerning the first objective, it should be remembered that the verification methods associated with each requirement were only "Review of Design" and "Test", as reported in Appendix A. The tool design can be considered as frozen at the moment of the validation test, so that the requirements reported in Table 6.1 can be verified and evaluated with an appropriate study of the VIPER architecture and structure.

	Requirements ID
Functional	FOP-FUN-030, FOP-FUN-131, FOP-FUN-132
Design	FOP-OPL-070, FOP-OPL-080, FOP-INT-010, FOP-INT-020, FOP-INT-030, FOP-INT-040, FOP-INT-050, FOP-INT-060, FOP-INT-070

Table 6.1: VIPER RoD Requirements

All the remaining requirements have to be verified by performing appropriate tests. Considering that this necessity can be strictly related to the fulfilment of the dedicated test

cases objectives, it is possible to combine the two needs. Since test cases must be specifically designed, they can be constructed to ensure that their successful execution verifies, at least, one associated requirement. Naturally, this association does not necessarily have to follow a one test case - one requirement relationship: there are instances where fulfilling a requirement necessitates multiple tests, just as a single test can cover multiple topics and, therefore, verify multiple requirements. [9]

The tool validation test is structured to reproduce the nominal working conditions of VIPER by linking it to the FOS systems, as explained in chapter 4. In this context, it was decided to rely on four different procedures to be studied and validated in order to obtain a significant set of results. Indeed, from each FOP, eight test cases will be created, which implies the generation of 32 cases. This number of cases allows to carry out a relevant test, which can treat nominal FOPs validations and examples of FOPs reporting one or more errors. In order to objectively assess whether the tool validation test is effectively passed and, consequently, whether VIPER can be considered validated, five pass criteria have been defined:

- Any VIPER RoD requirement has to be verified with a careful analysis of the tool design.
- The totality of test cases created must have a direct correspondence with all those requirements that involve a test method verification.
- The validation campaign can be considered concluded only if the entire set of the test cases is successfully passed.
- A test case is considered passed whenever the FOP validation results obtained with VIPER are consistent with the expected results specified in the test case definition.
- The evaluation of test cases results shall be performed through the analysis of the Validation Report generated by VIPER.

Once the aforementioned criteria are met, the VIPER tool validation can be considered completed. It is intrinsic in the test logic that the requirements that can be verified through the RoD method, must be checked before setting up the actual test to not reserve and use the Satellite Simulator whenever not necessary.

6.2. VIPER Requirements Review of Design

This paragraph presents the results of the analysis conducted on those VIPER requirements that were deemed necessary to be verified by a Review of the Design, previously

reported in Table 6.1. Indeed, with the final validation test, the tool design was formally investigated and evaluated. With regards to the requirements that referred to the PLUTO language, it has been demonstrated that the tool is effectively projected to deal with and elaborate FOP expressed in PLUTO format. (FOP-FUN-030) This aspect is corroborated by a cursory examination of the VIPER architecture, which reveals that the function of the Preliminary PLUTO-MIB Consistency Check block is to receive and deconstruct the PLUTO FOP, subsequently re-allocating its contents in a JSON file. This aspect specifically satisfies the FOP-INT-040 requirement. The standard adopted for editing was strictly in accordance with the ECSS-E-ST-70-32C standard, thereby positively answering to FOP-INT-030.

The integrability of the tool with other Argotec FOS systems has been granted through the design of a tool developed with the VSC code editor and programmed in Python language. This allows for a direct interface with the MCS software, as detailed in FOP-INT-010, FOP-INT-020 and FOP-INT-060. The VIPER tool has been effectively designed in Python, adopting the VSC editor from the initial function draft to the final main script definition. As stated in section 4.2, the tool exploits an already developed MQTT interface to communicate with the MCS, which demonstrates the VIPER ability to connect to already established elements within the FOS and, at the same time, guarantees a tested and secure connection with the Mission Control System. Similarly, the connection between the Satellite Simulator and the MCS was available to the FCT at the time of the tool design. As VIPER, the simulator connects as a client to the MQTT Broker, namely the MCS software. (FOP-INT-070)

The requirement pertaining to the possibility of cross-referencing both the FOP TCs and TM parameters against the Mission Database was classified as a non-mandatory requirement. Nevertheless, this connection has been implemented. The query requests allow for the investigation of the name and the number of every TC stored in the MIB. This includes the name of each TM packet and the parameters associated with it. These requests satisfy the three functional requirements FOP-FUN-050, FOP-FUN-131 and FOP-FUN-132.

The MCS CPU nominal computational effort shall be maintained at a constant level, with no increase in the computational time requested by other tools and scripts running simultaneously in the software. (FOP-OPL-080, FOP-OPL-070). These requests can be considered satisfied because, according to the VIPER design, the tool relies on interfaces and scripts running in the MCS software. VIPER runs on an external machine, therefore not influencing the CPU computational effort. [32] Consequently, no other tool that exploits the MCS software can be affected by the VIPER functioning.

6.3. Test FOPs Preparation

The cases created for the tool validation, as stated, are based on four different FOPs. These ones belong to the Flight Operations Plan of Argotec Hawk for Earth Observation (HEO) mission. As HEO is part of the IRIDE program, testing the tool with procedures created for this mission ensures relevance to the VIPER validation. At the time of the tool test design, the FOPs selected have already been reviewed and validated by the FCT members, thus ensuring the veracity of the validation campaign.

The selected FOP contents, their contexts of application and the description of the test cases derived from each procedure are presented below. In Appendix B section B.1 is possible to appreciate each FOP in its PLUTO source code.

6.3.1. TMTC Configuration Change

This procedure is thought to be executed whenever the S/C is in Safe Mode and there's the intention to establish for the first time, or alternatively restore, an active communication with it. In more detail, the FOP has the purpose of changing the satellite radio configuration, from an only receive status, to a transmission-reception mode, respectively indicated as Rx and TxRx. The full procedure in PLUTO format is reported in Appendix B section B.1.

The Preconditions Body indicates that the following FOP can be executed if the satellite is in its RX configuration. The Main Body is formed by five steps:

1. A TC step to send the command that set the radio in TxRx configuration.
2. A time-delay step of 5 seconds in order to wait for the radio configuration change.
3. A TM parameter request, to verify the status of the communication with the satellite.
4. A TM parameter request, to check that the radio is effectively in TxRx configuration.
5. A TM parameter request, to verify that the operational mode of the satellite is still the Safe Mode.

The FOP ID associated with the test is "V_TMTC". From the nominal procedure, eight test cases have been identified, primarily to verify the correct behaviour of the PLUTO Preliminary Check.

- **V-TMTC-00:** Test of the nominal V_TMTC procedure to acknowledge that VIPER can associate the validation status to an already validated FOP, reporting TC send, TM parameter provision and Delay steps.
- **V-TMTC-01:** Test of the nominal V_TMTC procedure indicating a wrong satellite name in the VIPER calling command, to verify the immediate interruption of the validation process.
- **V-TMTC-02:** Test of the nominal V_TMTC procedure indicating a wrong operational mode name in the VIPER calling command, to verify the immediate interruption of the validation process, but with a different error source with respect to the V_TMTC_01 case.
- **V-TMTC-03:** Test of a false negative V_TMTC procedure with a wrong *Main Body* construct, to check the VIPER inspection on the characteristic PLUTO structure.
- **V-TMTC-04:** Test of a false negative V_TMTC procedure with a missing PLUTO *End Step* structure, to verify the VIPER capability to detect whenever a step is not closed with the appropriate command.
- **V-TMTC-05:** Test of a false negative V_TMTC procedure with a wrong PLUTO *If and Then* loop construct, to verify the VIPER capacity to respect the ECSS Standard.
- **V-TMTC-06:** Test of a false negative V_TMTC procedure with a wrong TC send string, to check the VIPER ability to crosscheck PLUTO default commands and instructions.
- **V-TMTC-07:** Test of the nominal V_TMTC procedure, but disconnecting the MCS interface in order to force VIPER to interrupt the validation producing an error while trying to elaborate the first TC step.

The presented test cases have been associated with a group of VIPER functional and operational requirements, in order to explicitly indicate which of these are fulfilled on passing each individual test. Such an association is appreciable in Table 6.2.

Test ID	Associated VIPER Requirements
V-TMTC-00	FOP-FUN-010, FOP-FUN-020, FOP-FUN-050, FOP-FUN-110, FOP-FUN-274, FOP-OPL-060
V-TMTC-01	FOP-FUN-070
V-TMTC-02	FOP-FUN-072
V-TMTC-03	FOP-FUN-120
V-TMTC-04	FOP-FUN-275
V-TMTC-05	FOP-FUN-110
V-TMTC-06	FOP-FUN-271
V-TMTC-07	FOP-FUN-180

Table 6.2: TMTC Configuration Change Test Cases

6.3.2. Operational Mode Change

Operational Mode change procedures can be executed on numerous occasions throughout the mission, both in routine cases and during contingency events. The typical satellite operational modes that can be selected are: Safe, Communication, Sun Pointing, Service and Manoeuvre. [24] This specific FOP has the purpose of changing the S/C OPMODE, namely exiting from its Safe mode to enter the Communication mode. The full procedure in PLUTO format is reported in Appendix B section B.1. The Preconditions Body indicates that the following FOP can be executed if the satellite is effectively in Safe Mode, in its TX_RX configuration and with a Locked Status. The Main Body is instead formed by five steps:

1. A TC step to send the command that grant the Safe Mode exit.
2. A TC step to set the Communication Mode entering.
3. A TM Packet provision step, requesting the HK telemetry packet containing the updated OPMODE parameters.
4. A TM parameter request, to verify that the S/C is no longer in Safe Mode.
5. A TM parameter request, to check that the S/C is effectively in its Communication Mode.

The FOP ID associated with the test is “V_OPMODE”. From the nominal procedure, eight test cases have been identified with the intent of verify the correct behaviour of the PLUTO Preliminary Check on TC calling commands, to test the FOP-MIB Consistency

Check and to ensure that even with formally correct contents, but inserted with the wrong logic, the FOP won't be validated by VIPER.

- **V-OPMODE-00:** Test of the nominal V_OPMODE procedure to acknowledge that VIPER can associate the validation status to an already validated FOP, different from the first procedure analysed. The tool has to prove the correct elaboration of a TM packet request.
- **V-OPMODE-01:** Test of the nominal V_OPMODE procedure with a wrong Satellite Simulator initial configuration, to verify that VIPER indicates a formally correct FOP, but does not achieve the desired effects and is therefore not considered validated.
- **V-OPMODE-02:** Test of a false negative V_OPMODE procedure indicating a wrong TC PLUTO calling command, to check the VIPER ability to adhere to Argotec PLUTO Standard.
- **V-OPMODE-03:** Test of a false negative V_OPMODE procedure with a wrong TC subtype number leading to the verification of VIPER error of MIB inconsistency generation.
- **V-OPMODE-04:** Test of a false negative V_OPMODE procedure with a wrong TM parameter name in order to obtain the same result of V_OPMODE_03, but caused by a different source error.
- **V-OPMODE-05:** Test of a false negative V_OPMODE procedure with a TM Packet actually present in the Mission Database, but which cannot be requested through the application of this FOP.
- **V-OPMODE-06:** Test of a false negative V_OPMODE procedure with a wrong TM nominal parameter indication, to check the VIPER ability to compare the telemetries obtained from the simulator against the ones indicated in the PLUTO FOP.
- **V-OPMODE-07:** Test of the nominal V_OPMODE procedure with more than one wrong TM nominal parameter indications, to obtain the same result of the test V_OPMODE_06, but verifying the VIPER ability of handling different warnings and to register all of them with the *full* mode.

The presented test cases have been associated with a group of VIPER functional and operational requirements, in order to explicitly indicate which of these are fulfilled on passing each individual test. Such an association is appreciable in Table 6.3.

Test ID	Associated VIPER Requirements
V-OPMODE-00	FOP-FUN-010, FOP-FUN-020, FOP-FUN-040, FOP-FUN-270
V-OPMODE-01	FOP-FUN-276
V-OPMODE-02	FOP-FUN-273
V-OPMODE-03	FOP-FUN-133, FUN-OPL-010
V-OPMODE-04	FOP-FUN-134, FUN-OPL-020
V-OPMODE-05	FOP-FUN-250
V-OPMODE-06	FOP-FUN-060, FOP-FUN-210, FOP-FUN-230
V-OPMODE-07	FOP-FUN-220, FOP-FUN-272

Table 6.3: Operational Mode Change Test Cases

6.3.3. House Keeping Report Request

The satellite House Keeping telemetries are sent to the Ground Segment autonomously, to ensure a continue update on the S/C status. In order to avoid overwhelming the reception of this data packet, only those parameters considered of primary importance are sent automatically, while the remaining data requires a specific request. [10]

This FOP contains all the necessary commands to enable a House Keeping Report provision to retrieve other specific telemetry parameters and the subsequent disabling of the same after its reception. It is believed in fact that the latter will not clog up the telemetry data reception. The full procedure in PLUTO format is reported in Appendix B section B.1.

The Preconditions Body indicates that the following FOP can be executed if the satellite is effectively in Communication Mode, in its TX_RX configuration and with a Locked Status. The Main Body is formed by five steps:

1. A TC step to send the command that enable the HK report packet provision.
2. A TM Packet provision step, requesting the HK packet containing the enabled report.
3. A TC step to send the command that disable the HK report packet provision.
4. A TM parameter request, to check that the radio is effectively in TxRx configuration.
5. A TM parameter request, to verify that the operational mode of the satellite is still the Safe Mode.

The FOP ID associated with the test is “V_HK”. From the nominal procedure, eight test cases have been identified, mainly to verify the correct behaviour of the VIPER different operational modes and check the nominal handling of RAW TC parameters.

- **V-HK-00:** Test of the nominal V_HK procedure elaborated with the *full* VIPER mode to acknowledge that the tool can associate the validation status to a different and already validated FOP, reporting TCs with more than one parameters.
- **V-HK-01:** Test of the nominal V_HK procedure elaborated with the *full_input* VIPER mode, to verify that the same FOP receives the validated status with the use of a different operational mode.
- **V-HK-02:** Test of the nominal V_HK procedure elaborated with the *pluto_mib* and *mcs_interface* VIPER modes, to verify that the same FOP receives the validated status with the combination of different operational modes.
- **V-HK-03:** Test of the nominal V_HK procedure elaborated with the *pluto_mib* and *mcs_interface_input* VIPER modes, to verify that the same FOP receives the validated status with the combination of different operational modes with respect to the V_HK_02 case.
- **V-HK-04:** Test of the nominal V_HK procedure elaborated with the *pluto_mib* and *mcs_interface_debug* VIPER modes, to verify that the same FOP receives the validated status with the action of the debug mode.
- **V-HK-05:** Test of the nominal V_HK procedure that indicates a TC RAW parameter expressed in its ENG format, to demonstrate that the same FOP purpose is accomplished, obtaining the validated status.
- **V-HK-06:** Test of a false negative V_HK procedure with a wrong TC parameter, to verify the VIPER ability to interpretate the FOP correctness of contents. It shall produce an error message not related to the specific TC, but to the following steps that aims check the telecommand action.
- **V-HK-07:** Test of a false negative V_HK procedure with a wrong number of TC parameters leading to the VIPER error of MIB inconsistency generation.

The presented test cases have been associated with a group of VIPER functional and operational requirements, in order to explicitly indicate which of these are fulfilled on passing each individual test. Such an association is appreciable in Table 6.4.

Test ID	Associated VIPER Requirements
V-HK-00	FOP-FUN-010, FOP-FUN-020, FOP-INT-080, FOP-OPL-090
V-HK-01	FOP-FUN-150, FOP-INT-090, FOP-OPL-090
V-HK-02	FOP-FUN-160, FOP-INT-100, FOP-OPL-090
V-HK-03	FOP-FUN-170, FOP-INT-110, FOP-OPL-090
V-HK-04	FOP-FUN-240, FOP-INT-120, FOP-OPL-050, FOP-OPL-090
V-HK-05	FOP-OPL-040
V-HK-06	FOP-FUN-130
V-HK-07	FOP-FUN-071

Table 6.4: House Keeping Report Request Test Cases

6.3.4. On-Board Schedule Update

Both real-time and time-tagged TCs can be sent to the S/C during the visibility windows. In order to save this kind of telecommands in the on-board schedule, they must contain a specific indication regarding their execution time. [18] The FOP selected for the test has the purpose of inserting three different TCs in an empty on-board schedule and then verifying their actual presence inside it. Modifying the number of TCs and their execution times, this FOP can be executed allowing for the updating of the satellite schedule throughout the entirety of the mission. The full procedure in PLUTO format is reported in Appendix B section B.1.

The Preconditions Body indicates that the following FOP can be executed if the satellite is effectively in Communication Mode, in its TX_RX configuration and with a Locked Status. The Main Body is formed by seven steps:

1. A TC send - TM Packet provision step, requesting the on-board schedule report with the telecommand and checking the associated telemetry packet reception.
2. A TM parameter request, to verify the number of activities inside the on-board schedule.
3. A TC send step to insert the first time-tagged TC in the schedule.
4. A TC send step to insert the second time-tagged TC in the schedule.
5. A TC send step to insert the third time-tagged TC in the schedule.

6. A TC send - TM Packet provision step, requesting the updated on-board schedule report and checking the associated telemetry packet reception.
7. A TM parameter request, to verify that the three TC sent are effectively inserted in the on-board schedule, by checking the updated number of activities.

The FOP ID associated with the test is “V_SCHEDULE”. From the nominal procedure, eight test cases have been identified, to verify the correct insertion and handling of activities within the on-board schedule. Additionally, the test cases serve to assess the ability of VIPER to interface with the operator in the event of off-nominal parameters being received, and to test the execution of time-related activities.

Furthermore, an alternative version of the presented FOP has been integrated into the test cases. The goal of this test is to verify the possibility of obtaining the same outcome without explicitly requesting the on-board schedule telemetry packet. This has been done because the TC selected provides, automatically, the TM packet: for this reason a time delay of five seconds after the completion of the telecommand is thought to be sufficient for the FOP continuation. The full procedure in PLUTO format is reported in Appendix B section B.1, with an associated “V_S_V2” FOP ID.

- **V-SCHEDULE-00:** Test of the nominal V_SCHEDULE and V_S_V2 procedures to acknowledge that VIPER elaborates a FOP characterised by TCs with execution time specifics and with step comprehending both a TC send and a TM request. Testing V_S_V2 provides an adding verification on this second procedure, proving that the same outcome can be effectively obtained with two different methods. Being these procedures already validated, the tool shall report two validated status.
- **V-SCHEDULE-01:** Test of the nominal V_SCHEDULE procedure, but disconnecting the MCS interface in order to force VIPER to interrupt the validation producing an error while trying to elaborate the first TM parameter request.
- **V-SCHEDULE-02:** Test of a false negative V_SCHEDULE procedure indicating wrong required inputs, while using the *full_input* operational mode. The VIPER interface with the operator will be tested and, after a command of validation interruption, the error must be checked to ensure that the tool effectively reports the interruption of the validation process.
- **V-SCHEDULE-03:** Test of a false negative V_S_V2 procedure with a wrong time delay step construct, to check the VIPER inspection on the time characteristics of the FOP structure.

- **V-SCHEDULE-04:** Test of a false negative V_SCHEDULE procedure with a wrong TM packet PI2 number leading to the verification of VIPER error of MIB inconsistency generation.
- **V-SCHEDULE-05:** Test of a false negative V_SCHEDULE procedure with a TC execution time not consistent with the onboard time. It must be verified that VIPER effectively interrupt the validation of the procedure caused by the activation of a timeout indication while processing the TC.
- **V-SCHEDULE-06:** Test of a false negative V_SCHEDULE procedure with a wrong TC execution time calling command, to check the VIPER ability to adhere to Argotec PLUTO Standard.
- **V-SCHEDULE-07:** Test of a false negative V_SCHEDULE procedure indicating wrong TM nominal parameters in *full_input* operational mode. The VIPER interface with the operator will be tested and, after the insertion of an invalid input, the error must be checked to ensure that the tool effectively reports the interruption of the validation process and the error source.

The presented test cases have been associated with a group of VIPER functional and operational requirements, in order to explicitly indicate which of these are fulfilled on passing each individual test. Such an association is appreciable in Table 6.5.

Test ID	Associated VIPER Requirements
V-SCHEDULE-00	FOP-FUN-010, FOP-FUN-020, FOP-FUN-080, FOP-FUN-200
V-SCHEDULE-01	FOP-FUN-260
V-SCHEDULE-02	FOP-FUN-090, FOP-FUN-100, FOP-OPL-050
V-SCHEDULE-03	FOP-OPL-061
V-SCHEDULE-04	FOP-OPL-030
V-SCHEDULE-05	FOP-FUN-140, FOP-FUN-190
V-SCHEDULE-06	FOP-FUN-120
V-SCHEDULE-07	FOP-FUN-050, FOP-FUN-061, FOP-FUN-062

Table 6.5: On-Board Schedule Update Test Cases

6.4. Test Set-Up

In principle, the test is designed to ascertain the correct functioning of the VIPER system. Consequently, the FOP validation environment must be recreated. The tool equipment configuration can be presented by means of a checklist, where each action is listed in order of verification. Such a directory must include all the necessary items and a list with verification actions on their operation, together with specific checks on interfaces. The items required to perform the test are:

- Laptop, keyboard, mouse and monitor
- Visual Studio Code Editor (VSC)
- VIPER Main Script and functions
- FOPs in PLUTO format
- MIB server
- MCS Server
- Satellite Simulator

It should be noted that the VIPER software is in fact designed in Python, which means that the code editor VSC is essential for performing the necessary validation. In order for the Main Script to be correctly validated, all of the necessary data must be available at the time of the test setup. This includes the Main Script name, the validation mode name, the non-positional mandatory and optional arguments.

Subsequently, it is necessary to verify that both the server where the MIB is stored and the MCS server are operational. This allows for the verification of the Mission Database presence on the aforementioned server and of the loaded database version, which must correspond to the latest MIB update for the FOP validation. On the Satellite Simulator side, the FlatSat shall be activated and configured according to the test specifics, verifying its readiness to connect with the MCS Software. It is ultimately necessary to test all interfaces to ensure that they are compatible with the VIPER connectivity. In accordance with the VIPER functioning architecture, the interfaces that must be verified are:

- Query capacity to connect VIPER to the MIB
- VIPER capacity to connect as client to the MQTT broker
- The readiness of communication between VIPER and MCS
- Communication between MCS and the Simulator

Once confirmed that all the stated passages and verifications have been completed, together with the check on the FOP test-cases availability to the operator, the validation test of VIPER can start.

6.5. Test Results Analysis

The VIPER Validation Test was conducted in the Argotec Mission Control Centre, in accordance with all the mandatory indications and suggestions previously outlined. The totality of the 32 test cases identified has been tested and successfully completed.

For the sake of completeness, in Appendix B section B.2 are reported all the cases implemented in VIPER, coupled with the respective expected results, the ones actually obtained during the tool validation and the test status. Hereafter, a selection of the tested cases is presented in order to demonstrate the most pertinent examples that clearly illustrate the results obtained. To substantiate the veracity of these outcomes, the validation reports generated for each procedure directly by VIPER will be subjected to analysis.

```

FOP VALIDATION REPORT
-----
FOP Name: "V_SCHEDULE"
FOP Tested on Satellite: SAT_01
FOP Validation Test Date: 2024-05-21T16:16:48.370304
FOP Required Input: OPERATIONAL_STATUS, TX_RX_CONFIGURATION, NOT_SAFE_MODE
VIPER Operative Mode: full
-----
FOP PLUTO Preliminary Check: PASSED
-----
FOP - MIB Consistency Check: PASSED
-----
FOP Analysis with MCS: COMPLETED
-----
FOP Contents Verification: PASSED
-----
FOP Validation Status: VALIDATED

```

Listing 6.1: V_SCHEDULE FOP Validation Report

V_SCHEDULE_00 was one of those dedicated to obtaining a positive validation result for an already validated nominal procedure. As evidenced by the validation reports in Listing 6.1 and Listing 6.2, all characteristics of the test were duly reported, thereby certifying fidelity to the specifications of the test case. All five validation steps were successfully completed, with no errors reported. Furthermore, the second procedure in the test was processed and obtained the same positive result, thereby reinforcing the capacity

of the tool to facilitate a meticulous and comprehensive examination of the contents of a correctly written FOP that adheres to the designed standard.

FOP VALIDATION REPORT

```

-----
FOP Name: "V_S_V2"
FOP Tested on Satellite: SAT_01
FOP Validation Test Date: 2024-05-21T16:13:43.952357
FOP Required Input: OPERATIONAL_STATUS, TX_RX_CONFIGURATION, NOT_SAFE_MODE
VIPER Operative Mode: full
-----
FOP PLUTO Preliminary Check: PASSED
-----
FOP - MIB Consistency Check: PASSED
-----
FOP Analysis with MCS: COMPLETED
-----
FOP Contents Verification: PASSED
-----
FOP Validation Status: VALIDATED

```

Listing 6.2: V_S_V2 FOP Validation Report

Once it has been established that the VIPER system is capable of recognising and validating correctly written FOPs, the focus can be shifted to the testing of the tool in different operational modes. The combination of V_HK_00, V_HK_01, V_HK_02, V_HK_03, and V_HK_04 was created to demonstrate that the same validation status can be effectively assigned to a procedure through the application of the *full*, *full_input* and the *pluto_mib* coupled with *mcs_interface*, *mcs_interface_input* and *mcs_interface_debug*. Table B.3 in Appendix B section B.2 demonstrates that the entirety of these tests has been successfully completed.

Nevertheless, it was not possible to test the operator input functions due to the intrinsic correctness of the procedure contents. The V_SCHEDULE_02 test was designed to create an incongruence between the required inputs inserted in the FOP and the ones that should have been indicated, forcing the operator to insert a command for the validation process to be interrupted. The validation report, as illustrated in Listing 6.3, provides an overview of the VR layout, which allows for the identification of the error source and the user input history.

As V_SCHEDULE_00 demonstrated that VIPER is capable of accurately validating procedures written in accordance with the PLUTO standard, test V_OPMODE_02 proved that the tool is reliable in the instance of a procedural structure that is not formally correct too.

```

FOP VALIDATION REPORT
-----
FOP Name: "V_SCHEDULE_02"
FOP Tested on Satellite: SAT_01
FOP Validation Test Date: 2024-05-21T15:06:03.967496
FOP Required Input: OPERATIONAL_STATUS, TX_RX_CONFIGURATION, NOT_SAFE_MODE
VIPER Operative Mode: full
-----
FOP PLUTO Preliminary Check: NOT PASSED

REQUIRED INPUT ERROR. USER INPUT: Interrupt the FOP validation
-----
FOP - MIB Consistency Check: NOT TESTED
-----
FOP Analysis with MCS: NOT TESTED
-----
FOP Contents Verification: NOT TESTED
-----
FOP Validation Status: NOT VALIDATED

```

Listing 6.3: V_SCHEDULE_02 FOP Validation Report

```

FOP VALIDATION REPORT
-----
FOP Name: "V_OPMODE_02"
FOP Tested on Satellite: SAT_01
FOP Validation Test Date: 2024-05-21T15:02:16.459290
VIPER Operative Mode: full
-----
FOP PLUTO Preliminary Check: NOT PASSED

The Consistency Check highlights an ERROR of missing key command in initiate and confirm of
TC_128_XXX of YYY
-----
FOP - MIB Consistency Check: NOT TESTED
-----
FOP Analysis with MCS: NOT TESTED
-----
FOP Contents Verification: NOT TESTED
-----
FOP Validation Status: NOT VALIDATED

```

Listing 6.4: V_OPMODE_02 FOP Validation Report

As it is possible to appreciate in Listing 6.4, VIPER revealed the error specifically included in the nominal V_OPMODE procedure, namely the absence of a key element in the TC send command. As anticipated, this resulted in the immediate termination of the validation process of the procedure, which generated the shown Validation Report.

The cause of the error encountered is evident, clearly reported in the alert string together with the identification of the specific command containing the error. This allows for a quick and easy identification in the initial procedure, which facilitates the operator's ability to report a prompt resolving action to correct the identified issue. It is evident that the subsequent steps following the FOP-PLUTO preliminary check have not been addressed, and thus the procedure has not been validated.

At this stage of the validation process, the tool is required to provide clear and unambiguous feedback on the source of potential errors. This requirement is met by the V_HK_07, which generates an error alert when an error is inserted in the number of a specific TC parameter during the FOP contents consistency check against the Mission Database. As evidenced by the Validation Report in Listing 6.5, the VIPER error diagnostic function is working as intended, indicating which checks have failed and which have been unaffected by the error (in this case, the FOP-PLUTO Preliminary Check). The error message is clearly and comprehensively written, providing precise details of the submitted procedure and the expected outcome in relation to the query request. Naturally, the FOP has not been validated.

```
FOP VALIDATION REPORT
-----
FOP Name: "V_HK_07"
FOP Tested on Satellite: SAT_01
FOP Validation Test Date: 2024-05-21T15:05:20.565907
FOP Required Input: OPERATIONAL_STATUS, TX_RX_CONFIGURATION, NOT_SAFE_MODE
VIPER Operative Mode: full
-----
FOP PLUTO Preliminary Check: PASSED
-----
FOP - MIB Consistency Check: NOT PASSED

ERROR: Inconsistency between TC parameters number required (2) and reported (1)
-----
FOP Analysis with MCS: NOT TESTED
-----
FOP Contents Verification: NOT TESTED
-----
FOP Validation Status: NOT VALIDATED
```

Listing 6.5: V_HK_07 FOP Validation Report

So far it has been reported the test that have faced a forced interruption of the validation process prior to the use of the MQTT interface, and so of the MCS and Satellite Simulator connection. This space has been dedicated intentionally, reflecting a significant part of the test cases prepared for the VIPER validation campaign that aimed to prove the tool

capabilities of diagnosing FOP structural and contents consistency errors.

As illustrated by the Figure 6.1, the 44% of the cases executed during the test has been concluded before reaching the MCS elaboration of the FOP. This served to verify and check all the VIPER capabilities that can effectively provide an initial revision and, whenever necessary, a correction before actively using the simulator.

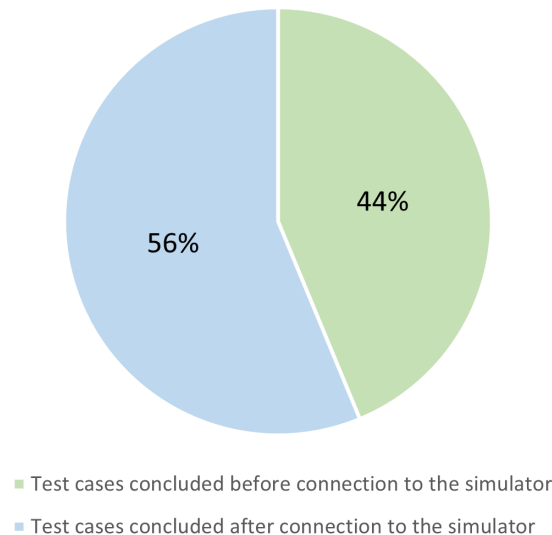


Figure 6.1: Satellite Simulator Use during VIPER Validation Test

As stated in subsection 2.1.1, the FlatSat is a precious resource and its availability is not always granted at any time and at any working unit during the development of a space mission. The tool has positively demonstrated the efficacy of its preliminary checks, which do not necessitate the direct utilisation of the simulator. The exploitation of this VIPER characteristic has the potential to significantly reduce the instances where the Satellite Simulator is connected and ready for the FOP validation, only to be discovered that the procedure is not correctly written, resulting in a considerable loss of time.

Nevertheless, the majority of the cases has been dedicated to the testing of the VIPER connection with the MCS and the Satellite Simulator, as this represented the most critical and delicate phase of the tool design. Still, the validation test has demonstrated an optimal behaviour of VIPER in elaborating the procedure, processing them with the MCS software and in handling the FlatSat responses. In support of this assertion, alongside the examples already given of validated procedures (Listing 6.1 and Listing 6.2), two further test cases are presented below: V_SCHEDULE_05 and V_OPMODE_07.

The initial discrepancy between the nominal V_SCHEDULE and the V_SCHEDULE_05 FOP is the result of erroneous execution times indicated during the definition of time-

tagged TCs parameters. Consequently, during the test phase, VIPER must proceed smoothly during the PLUTO MIB Preliminary Checks, with the error detection occurring solely at the MCS analysis level. It is evident that the procedure is correctly written, and that the tool is unable to predict the error from the initial verifications.

```

FOP VALIDATION REPORT
-----
FOP Name: "V_SCHEDULE_05"
FOP Tested on Satellite: SAT_01
FOP Validation Test Date: 2024-05-21T16:24:31.340204
FOP Required Input: OPERATIONAL_STATUS, TX_RX_CONFIGURATION, NOT_SAFE_MODE
VIPER Operative Mode: full
-----
Step 7: Started @ 16:24:11
OPERATOR: [{'requestID': '2bd27c78-bd56-43cb-9157-644f5385a65e', 'tcName': 'TC_1', 'tcOption': {'
  MAPID': 'YYY', 'executiontime': '2020-05-13T19:20:00'}}]
-----
FOP PLUTO Preliminary Check: PASSED
-----
FOP - MIB Consistency Check: PASSED
-----
FOP Analysis with MCS: NOT TESTED

ERROR: Timeout activated in TC_1. No response from MCS
-----
FOP Contents Verification: NOT TESTED
-----
FOP Validation Status: NOT VALIDATED

```

Listing 6.6: V_SCHEDULE_05 FOP Validation Report

The associated Validation Report corroborates the expectation, indicating the step containing the error and providing the alert string to the operator. Indeed, the TCs reporting the wrong time indications are processed by VIPER, which correctly attempted to create the telecommand topic, subscribed to it and published the TC send. At this point, the MCS received the request, but, as expected, it did not elaborate the telecommand and, consequently, it did not produce a response for the FOP tool. This resulted in the tool's timeout activation and the forced interruption of the validation process.

The execution of this specific test highlighted a result worthy of analysis: the same alert string produced by VIPER has in fact been provided for the V_TMTC_07 case too, despite the different nature of the error source. The corresponding Validation Report is appreciable in Listing 6.7, where the "ERROR: Timeout activated in [...]. No response from MCS" message has been associated during the FOP second step execution.

```

FOP VALIDATION REPORT
-----
FOP Name: "V_TMTC_07"
FOP Tested on Satellite: SAT_01
FOP Validation Test Date: 2024-05-21T15:12:19.265324
FOP Required Input: RX_configuration
-----
Step 2: Started @ 15:11:59
OPERATOR: [{'requestID': '470c148a-ab8f-40aa-8d69-567a42636f5b', 'tcName': 'RADIO_CONFIGURATION',
          'tcParameter': [{'name': 'PARAM_1_ID', 'value': '2', 'forceRaw': 'true', 'rawType': 'US'}], '
          tcOption': {'MAPID': 'YYY'}}]
-----
FOP PLUTO Preliminary Check: PASSED
-----
FOP - MIB Consistency Check: PASSED
-----
FOP Analysis with MCS: NOT TESTED

ERROR: Timeout activated in RADIO_CONFIGURATION. No response from MCS
-----
FOP Contents Verification: NOT TESTED
-----
FOP Validation Status: NOT VALIDATED

```

Listing 6.7: V_TMTC_07 FOP Validation Report

The reasons that led to the generation of the V_TMTC_07 Validation Report, and the consequent forced interruption, concern a disconnection between the tool and the MCS software, while the V_SCHEDULE_05 timeout has been triggered because of the upload of a wrong execution time in a time-tagged TC parameter. In other words, an interface mistake against an error within the procedure itself.

This event has been registered during the tool test campaign, but it has been accepted and not considered as a crippling error for the test cases. VIPER was in fact designed to detect any errors within a FOP, returning a description of the error found to the operator, and all of this was carried out as expected. However, a more precise reconstruction of error triggering events could be of help to in order to implement the specific changes to the FOP after the validation test.

Passing instead to the V_OPMODE_07 test, the main objective was to verify the correct handling of more than one off-nominal TM parameter warnings by VIPER. Being the FOP tested in *full* mode, the messages has been stored during the validation process and provided in the Validation Report. Going through this document step by step, the procedure appears to be correctly written, with TCs/TMs properly reported and actually present in the MIB.

Moreover, the entire FOP has been analysed and elaborated with the MCS and its contents have been found to correlate with the Satellite Simulator answers. Despite these positive outcomes, the three warning messages have been correctly identified and reported, as the transcription of the operator-MCS history of the procedure steps affected by these off-nominal parameters. Eventually, the NOT VALIDATED status has been assigned to the FOP, as expected.

```

FOP VALIDATION REPORT
-----
FOP Name: "V_OPMODE_07"
FOP Tested on Satellite: SAT_01
FOP Validation Test Date: 2024-05-21T15:30:37.555747
FOP Required Input: SAFE_MODE, TX_RX_CONFIGURATION, OPERATIONAL_STATUS
VIPER Operative Mode: full
-----
Step 3: Started @ 15:30:10
OPERATOR: [{ 'itemType': 'TM', 'itemName': 'TMTC_MODE' }]
SATELLITE: { 'rawValue': 3, 'isValid': True, 'timestamp': '2023-01-01T01:12:31.355000', 'engValue':
  'TX_RX_CONFIG' }
Step 3: Finished @ 15:30:10
-----
Step 4: Started @ 15:30:10
OPERATOR: [{ 'itemType': 'TM', 'itemName': 'TMTC_MODE' }]
SATELLITE: { 'rawValue': 3, 'isValid': True, 'timestamp': '2023-01-01T01:08:48.300000', 'engValue':
  'COMMUNICATION_MODE' }
Step 4: Finished @ 15:30:10
-----
Step 5: Started @ 15:30:10
OPERATOR: [{ 'requestID': 'bd52c88d-dc82-4933-9d8d-b18df5aa0661', 'tcName': 'EXIT_SAFE_MODE', '
  tcOption': { 'MAPID': 'YYY' } }]
SATELLITE: [{ 'stage': 'PTV', 'isFinal': False, 'success': True }, { 'stage': 'ACCEPT', 'isFinal':
  False, 'success': True }, { 'stage': 'COMPLETE', 'isFinal': True, 'success': False } ]
Step 5: Finished @ 15:30:15
-----
FOP PLUTO Preliminary Check: PASSED
-----
FOP - MIB Consistency Check: PASSED
-----
FOP Analysis with MCS: COMPLETED
-----
FOP Contents Verification: PASSED

NUMBER OF WARNING MESSAGES RECEIVED: 3
WARNING: OFF-nominal TM in TMTC_MODE. Expected: RX_CONFIG, Obtained: TX_RX_CONFIG
WARNING: OFF-nominal TM in OPMODE. Expected: SAFE_MODE, Obtained: COMMUNICATION_MODE
WARNING: The TC EXIT_SAFE_MODE COMPLETE status is not successfully completed
-----
FOP Validation Status: NOT VALIDATED

```

Listing 6.8: V_OPMODE_07 FOP Validation Report

The aforementioned tests clearly demonstrate the ability of VIPER to elaborate the FOP steps and its correct behaviour in handling the simulator responses. Furthermore, it is able to process these responses in order to generate the validation status of the procedure.

However, these examples are merely a small part of the VIPER-MCS-Simulator tests, which have been satisfactorily concluded. For the sake of completeness, the remaining cases are reported in Appendix B section B.2 together with the corresponding transcript of the validation outcomes.

It is important to note that VIPER has undergone a rigorous examination of its design process, and during the test, it has been subjected to a significant number of test cases. Throughout this process, all the requirements for VIPER have been met and the objectives stated at the beginning of the tool design can be considered judiciously achieved. Consequently, VIPER can be integrated within Argotec FOS to assist the Flight Control Team in the automated validation of Flight Operations Procedures.

7 | Conclusions

The primary objective of this Thesis was to develop an automated tool to validate the Flight Operations Procedures in the context of an Earth Observation space mission.

The first analysis concerned the typical structure of a FOP and the study of its characteristics in order to better understand the needs to be met and the factors that can be considered as recommendations only. To adequately satisfy this requirement, an in-depth review of the State of the Art was carried out, focusing on the actual experience of automation in the field of procedure execution and on the heritage of automated tools for FOP validation. After obtaining a relevant amount of implementation guidelines and after a careful study of the OPSVAL and ARVALIS tools working principles, the design of VIPER has been addressed.

The initial aspect to discuss regarded the necessity to translate a purely textual FOP into a machine-readable procedure, which implied a meticulous trade-off between the virtual machine languages developed so far, like STOL, SPELL, CCL, etc. . . . Eventually, the ECSS PLUTO Standard has been selected: the language application heritage, together with its intrinsic high level of compatibility with Mission Control Systems adopted by major space organisations, have been decisive in this choice.

On the basis of the functional and design requirements definition, the architecture and working principle of the new Argotec validation instrument could be established. The tool input was represented by the PLUTO FOP that must undergo a set of preliminary checks on the procedure correctness, both in terms of PLUTO Standard structure and of its contents consistency. The latter has been designed to be verified through query requests toward the Mission Database. Ensured the procedure correctness and once completed the transcription of the procedure into a JSON file, the VIPER design planned the execution of every FOP step through the Mission Control Software and the Satellite Simulator, or even the actual Flight Model during the spacecraft testing campaigns. It is in this way that the effective operability of a procedure can be ensured, verifying not only the correctness of its contents but analysing the satellite responses to the TC/TM sequences too. Appropriate operational modes have been designed to accomplish this objective and to enable VIPER

to generate a Validation Report showing the actual FOP status, together with the possible error and warning messages that could arise during the procedure evaluation process.

In order to validate the designed tool and verify its compatibility with the Argotec FOS developed within the IRIDE program, a testing campaign has been prepared and successfully performed. The test included the use of four FOPs, adapted without major modifications from four procedures belonging to the HEO Mission Flight Operations Plan. This aspect made the test campaign gain in relevance, planning for the execution of procedures already validated by the Flight Control Team in order to verify that the tool would produce the same result, effectively replacing human operators in the FOP validation process.

In addition, eight different cases were created from each procedure, thus reaching a total of 32 tests, with the aim of verifying not only VIPER validation capabilities of correct procedures, but also testing its ability to detect possible errors and warnings. The latter were specifically included in most of the tested cases, creating a set of false negatives.

As demonstrated by the results obtained, specifically reported in this thesis, VIPER has faced, completed and passed the entire test campaign, certifying its proper functioning and its integrability within the Argotec Flight Operations Segment.

7.1. VIPER Future Developments

Although every test case has been successfully completed, few remarks can be advanced in order to address the future developments of VIPER.

During the first testing phases in preparation of the final validation campaign, cases of false positives were encountered. The term “false positives” refers to the instances where VIPER returned a validated procedure status without any error or warning message, when this actually appeared to be an incorrect evaluation. This aspect must be carefully monitored, since extreme trust in an automated tool process could lead to ignore the presence of some errors within the procedure and to the insertion of wrong FOPs inside the Flight Operations Plan. VIPER aims to facilitate the FOPs validation while reducing both the workforce and the workload, but the assistance of properly trained human operators assistance cannot be eluded during its future applications.

The actual version of VIPER has passed the validation test and proved to be fully operative, but is to be considered open to the addition of improvements, both purely technical and concerning possible extensions and developments. For example, VIPER was designed to detect any errors within a FOP, returning a description of the error found to the operator and this functionality has been successfully verified.

However, it is strongly recommended to bring back an enhancement to the next VIPER release that integrates a finest error detection function, allowing a more precise reconstruction of error triggering events. Indeed, it has been observed that the same error message could be generated by different causes. Improving this diagnostic function would represent a substantial advance in the identification of the FOP elements that impede the procedure validation process, thereby facilitating the prompt subsequent modification of these elements.

In addition, while the PLUTO language is an excellent choice for writing procedures in the required machine-readable format, individuals outside the Flight Control Team might not have sufficient knowledge of it, as it is quite specific. To address this gap, it would be advisable to design a parser that interfaces between the operators and the tool. The purpose of this interface would be to process a standardised yet purely textual procedure, easily prepared by other Units, and convert it into the corresponding PLUTO format required by the tool. This update would ensure that the validation of the FOP is not solely assigned to the testing campaign conducted by the FCT, but that intermediate tests on the procedure itself can be performed by other parties involved in the writing of the FOP, such as the Software Engineering and System Engineering Units.

Furthermore, the tool architecture has been designed to interface with the system developed for the specific HEO mission. Nevertheless, with the addition of simple minor interface modifications, VIPER can also be used for different mission scenarios. The resulting tool is indeed capable of leveraging the properties of automated procedure processing, which not only opens the door to the use of VIPER for missions other than Earth Observation, but also constitutes the first step towards the design of a more complex system capable of elaborating and executing FOP with in-orbit satellites during real-time operations.

Bibliography

- [1] S. J. Carter and D. Quigley. Satellite test and operation procedures cost reduction through standardization. *IEEE Aerospace Conference*, 2006.
- [2] CCSDS. Tm synchronization and channel coding. *Recommended Standard: CCSDS 131.0-B-4*, 2022.
- [3] G. Chaudhri, J. Cater, and B. Kizzort. A model for a spacecraft operations language. *SpaceOps 2006 Conference*, 6 2006.
- [4] N. Creations. Mqtt what is it and how can you use it, 2017. URL <https://www.norwegiancreations.com/2017/07/mqtt-what-is-it-and-how-can-you-use-it/>.
- [5] R. DeHart and B. Sahin. Strategies and techniques for automation as implemented in eo-1 flight operations. *SpaceOps 2008 Conference*, 2008.
- [6] B. Demeuse and S. Valera. Pluto, a procedural language for users in tests and operations. *DASIA 1998-Data Systems In Aerospace*, 1998.
- [7] E. S. R. . S. Division. *Space engineering - Telemetry and telecommand packet utilization*. ESA-ESTEC, Noordwijk, The Netherlands, 2016. ECSS-E-ST-70-41C.
- [8] B. D’Urso. Multiprocess system for virtual instruments in python. *SciPy Conference - Pasadena, CA*, 2 2010.
- [9] ECSS. *Space engineering – Testing*. ESA-ESTEC, Noordwijk, The Netherlands, 2002. ECSS-E-10-03A.
- [10] ECSS. *Space engineering – Ground systems and operations - Monitoring and control data definition*. ESA-ESTEC, Noordwijk, The Netherlands, 2008. ECSS-E-ST-70-31C.
- [11] ECSS. *Space engineering – Test and operations procedure language*. ESA-ESTEC, Noordwijk, The Netherlands, 2008. ECSS-E-ST-70-32C.

- [12] ECSS. *Space engineering – Technical requirements specification*. ESA-ESTEC, Noordwijk, The Netherlands, 2009. ECSS-E-ST-10-06C.
- [13] ESA. Iride: La squadra è al completo, 2023. URL https://www.esa.int/Space_in_Member_States/Italy/IRIDE_La_squadra_e_al_completo.
- [14] ESA. Earth observation, 2024. URL https://www.esa.int/About_Us/Earth_observation.
- [15] J. C. Gil, N. Narula, and T. Lopez. There can only be one: heterogeneous satellite fleet automated operations with a single tool and language, the measat case. *SpaceOps 2014 Conference*, 2014.
- [16] C. A. Grasso. The fully programmable spacecraft: procedural sequencing for jpl deep space missions using vml (virtual machine language). *Proceedings, IEEE Aerospace Conference*, 1, 3 2002.
- [17] W. Jackson and W. Jackson. An introduction to json: Concepts and terminology. *JSON Quick Syntax Reference*, 2016.
- [18] S. Y. Kang, S. K. Lee, and K. H. Yang. The coms telecommand processing in the flight software. *INTELEC 2009-31st International Telecommunications Energy Conference, IEEE*, 2009.
- [19] D. Kortenkamp, R. P. Bonasso, K. M. Schreckenghost, D. and Dalal, V. Verma, and L. Wang. The fully programmable spacecraft: procedural sequencing for jpl deep space missions using vml (virtual machine language). *Proceedings of the 9th International Symposium on Artificial Intelligence, Robotics and Automation in Space*, 2 2008. i-SAIRAS-08.
- [20] C. Laroque. Model based operations validation system. *SpaceOps 2012*, 2012.
- [21] J. C. Lloret. Process for the definition and validation of flight control procedures. *DASIA 2003-Data Systems In Aerospace*, 532, 2003.
- [22] D. Lucia, R. Van Gijlswijk, S. Carosi, S. de la Rosa Steinz, S. Haag, and P. Athmann. Towards automated end-to-end testing and validation of operational data. *14th International Conference on Space Operations*, 2016.
- [23] G. Morelli, F. Bouleau, R. Chinchilla, and J. Noguero. A model for a spacecraft operations language. *Delivering on the Dream, Huntsville, Alabama*, 2010.
- [24] V. Nardone, A. Santone, M. Tipaldi, D. Liuzza, and L. Glielmo. Model checking tech-

- niques applied to satellite operational mode management. *IEEE Systems Journal*, 13(1), 1018-1029., 2018.
- [25] NASA. *Expanded guidance for NASA systems engineering*. National Aeronautics and Space Administration, Washington, D.C. - USA, 3 2016. HQ-E-DAA-TN42999. Volume 1: Systems Engineering Practices.
- [26] NASA. “ground data systems and mission operations, 2024. URL https://www.nasa.gov/smallsat-institute/sst-soa/ground-data-systems-and-mission-operations#_Toc121310346.
- [27] L. Parziale, D. T. Britt, C. Davis, J. Forrester, W. Liu, C. Matthews, and N. Rosselot. Tcp/ip tutorial and technical overview. *IBM - Redbooks*, 2006.
- [28] S. Pearson, F. Trifin, S. Reid, and W. Heinen. An integrated development and validation environment for operations automation. *SpaceOps 2012 Conference*, 2012.
- [29] M. Y. S. Prasad, S. K. Shivakumar, and V. Adimurthy. Space mission planning and operations. *Current Science*, 1791-1811, 2007.
- [30] S. Reid, W. Heinen, and S. Pearson. A full end-to-end automation chain with mois, pluto, matis, smf and scos-2000. *SpaceOps 2014 Conference*, 2014.
- [31] F. Sellmaier, T. Uhlig, and M. Schmidhuber. *Spacecraft Operations*. Springer International Publishing., 2022.
- [32] D. Versick, I. Wassmann, and D. Tavangarian. Power consumption estimation of cpu and peripheral components in virtual machines. *ACM SIGAPP Applied Computing Review*, 13(3), 2013.
- [33] J. R. Wertz, D. F. Everett, and J. J. Puschell. *Space Mission Engineering: The New SMAD*. Springer Dordrecht, 2011.

A | Appendix A

Hereafter are proposed the complete lists of VIPER functional and design requirements, previously discussed in chapter 4: as stated, the design requirements are additionally specified in operational and interface requirements.

ID	VIPER Functional Requirements	Verification Method
FOP_FUN_010	The tool shall validate Flight Operations Procedures (FOP)	T
FOP_FUN_020	The tool main script shall be designed to handle and validate different FOPs	T
FOP_FUN_030	The tool shall be capable to elaborate FOP written in PLUTO procedural language	RoD
FOP_FUN_040	The tool shall be able to process all the FOP steps in hierarchical order	T
FOP_FUN_050	The tool shall be able to perform the tasks required by each FOP step	T
FOP_FUN_060	The tool shall be able to produce outcomes classified as WARNING messages	RoD
FOP_FUN_061	Whenever a WARNING message is produced, the tool shall report the warning triggering condition	T
FOP_FUN_062	Whenever an WARNING message is produced, the tool shall provide the user with the possibility to proceed or exit the procedure validation process	T
FOP_FUN_070	The tool shall be able to produce outcomes classified as ERROR messages	RoD
FOP_FUN_071	Whenever an ERROR message is produced, the tool shall report the error triggering condition	T
FOP_FUN_072	Whenever an ERROR message is produced, the tool shall interrupt the procedure validation process	T
FOP_FUN_080	The tool shall report to the user the required inputs registered within the FOP	T

Table A.1: VIPER Functional Requirements (First Set)

ID	VIPER Functional Requirements	Verification Method
FOP_FUN_090	The tool shall request the user for an input to proceed the execution process after the display of the FOP required input parameters	RoD
FOP_FUN_100	In case the operator indicates that one or more of the requested inputs are wrong, the tool shall terminate the execution process	T
FOP_FUN_110	During the FOP validation, a consistency check shall be executed to verify the FOP PLUTO language correctness	T
FOP_FUN_120	If the PLUTO structure of the FOP is evaluated wrong, the tool shall report an ERROR message	RoD
FOP_FUN_130	The tool shall be designed to provide a verification on the FOP contents correctness before processing the validation	T
FOP_FUN_131	The consistency of a TC command should be checked inside the MIB	T
FOP_FUN_132	The consistency of a TM should be checked inside the MIB	RoD
FOP_FUN_133	In case of inconsistency between the TC specifics and the ones present inside the MIB, the tool should provide an ERROR message	T
FOP_FUN_134	In case of inconsistency between the TM specifics and the ones present inside the MIB, the tool should provide an ERROR message	T
FOP_FUN_140	If the PLUTO FOP contents are considered wrong, the tool shall report an ERROR message	T
FOP_FUN_150	The tool shall load and validate all the instructions inside the FOP including TM, TC, warnings messages and action commands	RoD
FOP_FUN_160	During the FOP execution, the tool shall follow the FOP TM, TC, warnings instructions and action commands in the hierarchical order expressed in the FOP step	T
FOP_FUN_170	The tool shall send the TC triggering action to the satellite simulator	T
FOP_FUN_180	In case a TC is not received by the satellite simulator, the tool shall report an ERROR message	RoD
FOP_FUN_190	In case a TC is not executed by the satellite simulator, the tool shall report an ERROR message	T

Table A.2: VIPER Functional Requirements (Second Set)

ID	VIPER Functional Requirements	Verification Method
FOP_FUN_200	The tool shall be capable to elaborate timetagged TCs in order to insert them in the satellite operations schedule	T
FOP_FUN_210	The tool shall be able to provide single TM parameters and check the parameter obtained against the ones indicated as nominal values in the FOP	RoD
FOP_FUN_220	In case a TM indicator verifies an off-nominal parameter, the tool shall report a WARNING message	T
FOP_FUN_230	In case of a FOP validation process interruption due to a TM parameter off-nominality, the tool shall report the contingency procedure ID to follow if previously specified inside the FOP	T
FOP_FUN_240	The tool shall be capable to retrieve TM packets from the satellite simulator	T
FOP_FUN_250	In case a requested TM packet is not received by the tool, the Validation Report shall indicate an ERROR message	RoD
FOP_FUN_260	In case a requested TM parameter is not received by the tool, the Validation Report shall indicate an ERROR message	T
FOP_FUN_270	The tool shall generate a Validation Report at the end of the FOP execution	T
FOP_FUN_271	The tool shall indicate the name of the satellite used for the test, the FOP name and the FOP validation execution time in the Validation Report	RoD
FOP_FUN_272	The tool shall indicate the number of WARNINGS received throughout the validation process and their relative descriptions in the Validation Report	T
FOP_FUN_273	The tool shall indicate the number of ERRORS received throughout the validation process and their relative descriptions in the Validation Report	T
FOP_FUN_274	In case no WARNING or ERROR are highlighted during the procedure execution, the tool shall assign a positive FOP validation status in the Validation Report	RoD
FOP_FUN_275	In case the procedure validation outcome do highlights an ERROR, the tool shall assign a negative FOP validation status in the Validation Report	T
FOP_FUN_276	In case the procedure validation outcome do highlights at least one WARNING, the tool shall assign a negative FOP validation status in the Validation Report	T

Table A.3: VIPER Functional Requirements (Third Set)

ID	VIPER Interface Requirements	Verification Method
FOP_INT_010	The tool shall be compatible with VSC source code editor	RoD
FOP_INT_020	The tool main script shall be produced in Python language	T
FOP_INT_030	The PLUTO FOP shall be compliant with ECSS-E-ST-70-32C Standard to be interpreted by the tool	T
FOP_INT_040	The tool script shall elaborate the PLUTO FOP and report it contents in a python JSON file	RoD
FOP_INT_050	The tool should interface the MIB	T
FOP_INT_060	The tool shall exchange TMs and TCs with the Mission Control Software through a dedicated interface	T
FOP_INT_070	The Mission Control Software shall be connected to the satellite simulator through a dedicated interface	T
FOP_INT_080	The tool shall be capable to receive the TM packets elaborated by the MCS software exploiting the MQTT interface	T
FOP_INT_090	The tool shall display to the operator the sent TC execution status, the received TM parameters and TM packets	T
FOP_INT_100	The tool should show at display a human readable format of the TC sent and TM parameters	T
FOP_INT_110	The tool shall autonomously save the generated Validation Report in an apposite output folder	T
FOP_INT_120	The generated Validation Report shall be provided to the operator in a .txt file format	T

Table A.4: VIPER Interface Requirements

ID	VIPER Operational Requirements	Verification Method
FOP_OPL_010	The tool shall identify the TCs through the Type-Subtype-Apid classification	RoD
FOP_OPL_020	The tool shall identify the TM parameters through their TM parameter name	T
FOP_OPL_030	The tool shall identify TM packets through the Type-Subtype-Apid-PI1-PI2 classification	T
FOP_OPL_040	The tool shall be able to process the TM values expressed in both engineering and raw value format	RoD
FOP_OPL_050	Operator inputs shall be analysed and processed indicating if they correspond to a continuation of the validation process, a termination or whether the user input is considered invalid	T
FOP_OPL_060	In presence of a wait-time instruction, the indicated time interval shall be set and effectively waited during the FOP execution process	T
FOP_OPL_070	During the validation process, the tool shall not increase the computational time requested by other scripts running simultaneously in the MCS	T
FOP_OPL_080	The tool activity shall not increase the computational effort of the MCS of more than TBD in TBD [time interval]	T
FOP_OPL_090	The identical FOP validation status of a specific procedure shall be obtained through the application of different operative modes	T

Table A.5: VIPER Operational Requirements

B | Appendix B

This appendix is dedicated to all documentation produced in preparation for, during and after the execution of the VIPER tool validation test. In section B.1 is possible to consult the four procedures applied to create the different test cases, while section B.2 reports the expected and obtained results of the test, grouped for each of the FOP.

B.1. VIPER Test FOPs

```

procedure
  preconditions
    initiate and confirm step REQUIRED_INPUTS
      declare RX_configuration;
      log RX_configuration
      ask user
    end step;
  end preconditions
  main
    initiate and confirm step RADIO_CONFIGURATION
      initiate and confirm TC of TC_129_XXX of YYY;
      with arguments
        PARAM_1_ID := US 2;
      end with
    end step;

    initiate and confirm step Delay
      declare delay
      delay = 5
      wait for delay
    end step;

    initiate and confirm step TMTc_Status
      LS := get value "TMTc_01" of TM_003_XXX;
      if LS is != "OPERATIONAL_STATUS" then
        inform user "TMTc_01 IS OFF-NOMINAL";
        raise event OFF_NOMINAL_TMTc_01
      end if
    end step;

    initiate and confirm step Radio_mode_check
      RADIO_MODE := get value "TMTc_MODE" of TM_003_XXX;
      if RADIO_MODE is != "TX_RX_CONFIG" then
        inform user "TMTc_MODE IS OFF-NOMINAL, follow procedure FOP-220049 to change the mode
          ";
        raise event OFF_NOMINAL_TMTc_MODE
      end if
    end step;

    initiate and confirm step Safe_Mode
      MODE := get value "OPMODE" of TM_003_XXX;
      if MODE is != "SAFE_MODE" then
        inform user "SAFE_MODE IS OFF-NOMINAL, S/C NOT IN SAFE MODE ";
        raise event OFF_NOMINAL_OPMODE
      end if
    end step;
  end main;
end procedure

```

Listing B.1: V_TMTc FOP

```

procedure
  preconditions
    initiate and confirm step REQUIRED_INPUTS
      declare NOT_SAFE_MODE; declare TX_RX_CONFIGURATION; declare OPERATIONAL_STATUS;
      log NOT_SAFE_MODE; log TX_RX_CONFIGURATION; log OPERATIONAL_STATUS;
      ask user
    end step;
    initiate and confirm step TMTc_Status
      LS := get value "TMTc_01" of TM_003_XXX;
      if LS is != "OPERATIONAL_STATUS" then
        inform user "TMTc_01 IS OFF-NOMINAL";
        raise event OFF_NOMINAL_TMTc_01
      end if
    end step;
    initiate and confirm step Radio_mode_check
      RADIO_MODE := get value "TMTc_MODE" of TM_003_XXX;
      if RADIO_MODE is != "TX_RX_CONFIG" then
        inform user "TMTc_MODE IS OFF-NOMINAL, follow procedure FOP-220049 to change the mode
          ";
        raise event OFF_NOMINAL_TMTc_MODE
      end if
    end step;
    initiate and confirm step Not_Safe_Mode
      SAFE_MODE := get value "OPMODE" of TM_003_XXX;
      if SAFE_MODE is = "SAFE" then
        inform user "OPMODE IS OFF-NOMINAL, S/C NOT IN SAFE MODE ";
        raise event OFF_NOMINAL_OPMODE
      end if
    end step;
  end preconditions
  main
    initiate and confirm step HK_TC_ENABLE
      initiate and confirm TC of TC_003_XXX of YYY;
      with arguments
        HK_REPORT_ENABLE := LO 1,
        HK_REPORT_ENABLE_PARAMETER_1 := LO 2
      end with;
    end step;
    initiate and confirm step HK_TMPkt_report
      verify packet of TM_003_XXX of YYY_002_000;
    end step;
    initiate and confirm step HK_TC_DISABLE
      initiate and confirm TC of TC_003_XXX of YYY;
      with arguments
        HK_REPORT_DISABLE := LO 1,
        HK_REPORT_DISABLE_PARAMETER_1 := LO 2
      end with;
    end step;
  end main;
end procedure

```

Listing B.2: V_HK FOP

```

procedure
  preconditions
    initiate and confirm step REQUIRED_INPUTS
      declare SAFE_MODE; declare TX_RX_CONFIGURATION; declare OPERATIONAL_STATUS;
      log SAFE_MODE; log TX_RX_CONFIGURATION; log OPERATIONAL_STATUS;
      ask user
    end step;

    initiate and confirm step TMTC_Status
      LS := get value "TMTC_01" of TM_003_XXX;
      if LS is != "OPERATIONAL_STATUS" then
        inform user "TMTC_01 IS OFF-NOMINAL";
        raise event OFF_NOMINAL_TMTC_01
      end if
    end step;

    initiate and confirm step Radio_mode_check
      RADIO_MODE := get value "TMTC_MODE" of TM_003_XXX;
      if RADIO_MODE is != "TX_RX_CONFIG" then
        inform user "TMTC_MODE IS OFF-NOMINAL, follow procedure FOP-220049 to change the mode
          ";
        raise event OFF_NOMINAL_TMTC_MODE
      end if
    end step;

    initiate and confirm step Not_Safe_Mode
      SAFE_MODE := get value "OPMODE" of TM_003_XXX;
      if SAFE_MODE is != "SAFE" then
        inform user "OPMODE IS OFF-NOMINAL, S/C NOT IN SAFE MODE ";
        raise event OFF_NOMINAL_OPMODE
      end if
    end step;
  end preconditions
  main
    initiate and confirm step EXIT_SAFE_MODE
      initiate and confirm TC of TC_128_XXX of YYY
    end step;

    initiate and confirm step ENTER_COMMUNICATION_MODE
      initiate and confirm TC of TC_128_XXX of YYY;
      with arguments
        OPMODE := "COMMUNICATION_MODE"
      end with;
    end step;

```

Listing B.3: V_OPMODE FOP - First Part

```

initiate and confirm step HK_PKT_REQUEST
    verify packet of TM_003_XXX of YYY_001_000;
end step;

initiate and confirm step SVerify_Exit_Safe_Mode
    MODE := get value "OPMODE" of TM_003_XXX;
    if MODE is = "SAFE_MODE" then
        inform user "MODE IS OFF-NOMINAL, S/C IN SAFE MODE ";
        raise event OFF_NOMINAL_OPMODE
    end if
end step;

initiate and confirm step VERIFY_COMM_MODE
    MODE := get value "OPMODE" of TM_003_XXX;
    if MODE is != "COMM" then
        inform user "OPMODE parameter OFF-NOMINAL, S/C not in COMMUNICATION MODE";
        raise event SC_NOT_COMM
    end if
end step;
end main;
end procedure

```

Listing B.4: V_OPMODE FOP - Second Part

```

procedure
    preconditions
        initiate and confirm step REQUIRED_INPUTS
            declare NOT_SAFE_MODE; declare TX_RX_CONFIGURATION; declare OPERATIONAL_STATUS;
            log NOT_SAFE_MODE; log TX_RX_CONFIGURATION; log OPERATIONAL_STATUS;
            ask user
        end step;

        initiate and confirm step TMTc_Status
            LS := get value "TMTc_01" of TM_003_XXX;
            if LS is != "OPERATIONAL_STATUS" then
                inform user "TMTc_01 IS OFF-NOMINAL";
                raise event OFF_NOMINAL_TMTc_01
            end if
        end step;

        initiate and confirm step Radio_mode_check
            RADIO_MODE := get value "TMTc_MODE" of TM_003_XXX;
            if RADIO_MODE is != "TX_RX_CONFIG" then
                inform user "TMTc_MODE IS OFF-NOMINAL, follow procedure FOP-220049 to change the mode
                    ";
                raise event OFF_NOMINAL_TMTc_MODE
            end if
        end step;

```

Listing B.5: V_SCHEDULE FOP - First Part

```

initiate and confirm step Not_Safe_Mode
  SAFE_MODE := get value "OPMODE" of TM_003_XXX;
  if SAFE_MODE is = "SAFE" then
    inform user "OPMODE IS OFF-NOMINAL, S/C NOT IN SAFE MODE ";
    raise event OFF_NOMINAL_OPMODE
  end if
end step;
end preconditions
main
  initiate and confirm step Schedule_Request
    initiate and confirm TC of TC_011_XXX of YYY;
    verify packet of TM_011_XXX of YYY_000_000;
  end step;

  initiate and confirm step scheduling_activities
    TSA := get value "SCHEDULE_ACTIVITIES" of TM_003_XXX;
    if TSA is != 0 then
      inform user "Time schedule for the activities is not empty";
      raise event NOT_EMPTY_SCHEDULE
    end if
  end step;

  initiate and confirm step Telecommand_1
    initiate and confirm TC of TC_017_XXX of YYY;
    with directives
      exe_time := "2024-05-20T14:20:00";
    end with;
  end step;

  initiate and confirm step Telecommand_2
    initiate and confirm TC of TC_017_XXX of YYY;
    with directives
      exe_time := "2024-05-20T14:25:00";
    end with;
  end step;

  initiate and confirm step Telecommand_3
    initiate and confirm TC of TC_017_XXX of YYY;
    with directives
      exe_time := "2024-05-20T14:30:00";
    end with;
  end step;

  initiate and confirm step Check_schedule
    initiate and confirm TC of TC_011_XXX of YYY;
    verify packet of TM_011_XXX of YYY_000_000;
  end step;

  initiate and confirm step SCHEDULE_TM
    TSA := get value "SCHEDULE_ACTIVITIES" of TM_003_XXX;
    if TSA is != 3 then
      inform user "Time schedule for the activities is not containing 3 activities";
      raise event WRONG_SCHEDULE
    end if
  end step;
end main;
end procedure

```

Listing B.6: V_SCHEDULE FOP - Second Part

```

procedure
  preconditions
    initiate and confirm step REQUIRED_INPUTS
      declare NOT_SAFE_MODE; declare TX_RX_CONFIGURATION; declare OPERATIONAL_STATUS;
      log NOT_SAFE_MODE; log TX_RX_CONFIGURATION; log OPERATIONAL_STATUS;
      ask user
    end step;

    initiate and confirm step TMTc_Status
      LS := get value "TMTc_01" of TM_003_XXX;
      if LS is != "OPERATIONAL_STATUS" then
        inform user "TMTc_01 IS OFF-NOMINAL";
        raise event OFF_NOMINAL_TMTc_01
      end if
    end step;

    initiate and confirm step Radio_mode_check
      RADIO_MODE := get value "TMTc_MODE" of TM_003_XXX;
      if RADIO_MODE is != "TX_RX_CONFIG" then
        inform user "TMTc_MODE IS OFF-NOMINAL, follow procedure FOP-220049 to change the mode
          ";
        raise event OFF_NOMINAL_TMTc_MODE
      end if
    end step;

    initiate and confirm step Not_Safe_Mode
      SAFE_MODE := get value "OPMODE" of TM_003_XXX;
      if SAFE_MODE is = "SAFE" then
        inform user "OPMODE IS OFF-NOMINAL, S/C NOT IN SAFE MODE ";
        raise event OFF_NOMINAL_OPMODE
      end if
    end step;
  end preconditions
  main
    initiate and confirm step Schedule_Request
      initiate and confirm TC of TC_011_XXX of YYY;
    end step;

    initiate and confirm step Delay_1
      declare delay
      delay = 15
      wait for delay
    end step;

    initiate and confirm step scheduling_activities
      TSA := get value "SCHEDULE_ACTIVITIES" of TM_003_XXX;
      if TSA is != 0 then
        inform user "Time schedule for the activities is not empty";
        raise event NOT_EMPTY_SCHEDULE
      end if
    end step;

```

Listing B.7: V_S_V2 FOP - First Part

```
    initiate and confirm step Telecommand_1
  initiate and confirm TC of TC_017_XXX of YYY;
    with directives
      exe_time := "2024-05-20T14:20:00";
    end with;
  end step;
  initiate and confirm step Telecommand_2
    initiate and confirm TC of TC_017_XXX of YYY;
      with directives
        exe_time := "2024-05-20T14:25:00";
      end with;
    end step;
  initiate and confirm step Telecommand_3
    initiate and confirm TC of TC_017_XXX of YYY;
      with directives
        exe_time := "2024-05-20T14:30:00";
      end with;
    end step;

  initiate and confirm step Check_schedule
    initiate and confirm TC of TC_011_XXX of YYY;
  end step;

  initiate and confirm step Delay_2
    declare delay
      delay = 15
      wait for delay
    end step;

  initiate and confirm step SCHEDULE_TM
    TSA := get value "SCHEDULE_ACTIVITIES" of TM_003_XXX;
    if TSA is != 3 then
      inform user "Time schedule for the activities is not containing 3 activities";
      raise event WRONG_SCHEDULE
    end if
  end step;
end main;
end procedure
```

Listing B.8: V_S_V2 FOP - Second Part

B.2. VIPER Test Results

Test ID	Expected Results	Obtained Results	Status
V-TMTC-00	FOP Validated	FOP Validation Status: Validated	PASSED
V-TMTC-01	FOP PLUTO Preliminary Check: NOT PASSED. FOP Not Validated	FOP PLUTO Preliminary Check: NOT PASSED. ERROR: Mode name not correct	PASSED
V-TMTC-02	FOP PLUTO Preliminary Check: NOT PASSED. FOP Not Validated	FOP PLUTO Preliminary Check: NOT PASSED. ERROR: Satellite name is not correct in full mode	PASSED
V-TMTC-03	FOP PLUTO Preliminary Check: NOT PASSED. FOP Not Validated	FOP PLUTO Preliminary Check: NOT PASSED. The Consistency Check highlights an ERROR of Main_inconsistency	PASSED
V-TMTC-04	FOP PLUTO Preliminary Check: NOT PASSED. FOP Not Validated	FOP PLUTO Preliminary Check: NOT PASSED. The Consistency Check highlights an ERROR of missing "end step" command in end	PASSED
V-TMTC-05	FOP PLUTO Preliminary Check: NOT PASSED. FOP Not Validated	FOP PLUTO Preliminary Check: NOT PASSED. The Consistency Check highlights an ERROR of If_and_Then_inconsistency inside the step TMTC_Status	PASSED
V-TMTC-06	FOP PLUTO Preliminary Check: NOT PASSED. FOP Not Validated	FOP PLUTO Preliminary Check: NOT PASSED. ERROR in TC inside RADIO_CONFIGURATION: missing key element in TC command	PASSED
V-TMTC-07	FOP Analysis with MCS: NOT COMPLETED. FOP Not Validated	FOP PLUTO Preliminary Check: PASSED, FOP-MIB Consistency Check: PASSED, FOP Analysis with MCS: NOT COMPLETED, FOP Not Validated. ERROR:Timeout activated in RADIO_CONFIGURATION. No response from MCS	PASSED

Table B.1: V-TMTC Test Cases for VIPER Validation: expected and obtained results

Test ID	Expected Results	Obtained Results	Status
V-OPMODE-00	FOP Validated	FOP Validation Status: Validated	PASSED
V-OPMODE-01	FOP Not Validated, TM warnings at Validation level caused by wrong satellite initial configuration	FOP Not Validated. OFF-nominal TM in OPMODE. Ex: COMMUNICATION_MODE, Ob: SAFE_MODE, OFF-nominal TM in OPMODE. Ex: Not SAFE_MODE, Ob: SAFE_MODE	PASSED
V-OPMODE-02	FOP Not Validated, wrong key element in TC sending at PLUTO Preliminary Check level	FOP PLUTO Preliminary Check: NOT PASSED. The Consistency Check highlights an ERROR of missing key command in TC_128_XXX	PASSED
V-OPMODE-03	FOP Not Validated, TC inconsistency at MIB consistency check level	FOP - MIB Consistency Check: NOT PASSED, FOP Not Validated. Error in TC type, subtype or apid in Query	PASSED
V-OPMODE-04	FOP Not Validated, TM parameter inconsistency with MIB at MIB consistency Check level	FOP - MIB Consistency Check: NOT PASSED, FOP Not Validated. ERROR: Inconsistency between TM name required and TM name reported	PASSED
V-OPMODE-05	FOP Not Validated, missing response from the MCS at FOP analysis with MCS level	FOP Analysis with MCS: NOT COMPLETED, FOP Not Validated. ERROR: Timeout activated in XXX_YYY_ZZZ_TMPKT. No response from MCS	PASSED
V-OPMODE-06	FOP Not Validated, TM warning at Validation level	FOP Not Validated. WARNING: OFF-nominal TM in OPMODE. Ex: SUNPOINTING, Ob: COMMUNICATION_MODE	PASSED
V-OPMODE-07	FOP Not Validated, TM warnings at Validation level	FOP Not Validated. WARNING: OFF-nominal TM in TMTTC_MODE. Ex: RX_CONFIG, Ob: TX_RX_CONFIG, OFF-nominal TM in OPMODE. Ex: SAFE_MODE, Ob: COMMUNICATION_MODE, The TC EXIT_SAFE_MODE COMPLETE status is not completed	PASSED

Table B.2: V-OPMODE Test Cases for VIPER Validation: expected and obtained results

Test ID	Expected Results	Obtained Results	Status
V-HK-00	FOP Validated with full mode	FOP Validation Status: Validated	PASSED
V-HK-01	FOP Validated with full_input mode	FOP Validation Status: Validated	PASSED
V-HK-02	FOP Validated with pluto_mib + mcs_interface modes	FOP Validation Status: Validated	PASSED
V-HK-03	FOP Validated with pluto_mib + mcs_interface_input modes	FOP Validation Status: Validated	PASSED
V-HK-04	FOP Validated with pluto_mib + mcs_interface_debug modes	FOP Validation Status: Validated	PASSED
V-HK-05	FOP Validated with TC parameter not expressed in its RAW format	FOP Validation Status: Validated	PASSED
V-HK-06	FOP Not Validated, missing response from the MCS at FOP analysis with MCS level because TC action was not successful	FOP Analysis with MCS: NOT COMPLETED, FOP Not Validated. ERROR: Timeout activated in XXX_YYY_ZZZ_housekeeping telemetry. No response from MCS	PASSED
V-HK-07	FOP Not Validated, TC parameter number inconsistency with MIB at MIB consistency Check level	FOP - MIB Consistency Check: NOT PASSED, FOP Not Validated.ERROR: Inconsistency between TC parameters number required (2) and reported (1)	PASSED

Table B.3: V-HK Test Cases for VIPER Validation: expected and obtained results

Test ID	Expected Results	Obtained Results	Status
V-SCHEDULE-00	FOP Validated	FOP Validation Status: Validated	PASSED
V-SCHEDULE-01	FOP Not Validated, MCS Server connection error at FOP analysis with MCS level	FOP Analysis with MCS: NOT COMPLETED, FOP Not Validated. ERROR: Timeout activated in TMTC_STATUS. No MCS response	PASSED
V-SCHEDULE-02	FOP Not Validated, process interrupted by the operator during the MCS analysis	FOP PLUTO Preliminary Check: NOT PASSED. REQUIRED INPUT ERROR. USER INPUT: Interrupt the FOP validation	PASSED
V-SCHEDULE-03	FOP Not Validated, wrong Delay construction, error at PLUTO Preliminary Check level	FOP PLUTO Preliminary Check: NOT PASSED, FOP Not Validated. ERROR: The Consistency Check highlights an ERROR in the DELAY command definition in Delay_1	PASSED
V-SCHEDULE-04	FOP Not Validated, TM packet inconsistency at MIB consistency check	FOP - MIB Consistency Check: NOT PASSED, FOP Not Validated. ERROR: Error in TMPkt name in Query	PASSED
V-SCHEDULE-05	FOP Not Validated, time-tagged TC not elaborated at MCS analysis level	FOP Analysis with MCS: NOT COMPLETED, FOP Not Validated. ERROR: Timeout activated in Telecommand_1. No response from MCS	PASSED
V-SCHEDULE-06	FOP Not Validated, wrong PLUTO time-tagged TC structure at PLUTO Preliminary Check level	FOP PLUTO Preliminary Check: NOT PASSED, FOP Not Validated. The Consistency Check highlights an ERROR of With_inconsistency inside the step Telecommand_1	PASSED
V-SCHEDULE-07	FOP Not Validated, process interrupted by the operator during the MCS analysis with invalid input	FOP Analysis with MCS: NOT COMPLETED. WARNING: OFF-nominal TM in OPMODE. Ex: SAFE_MODE, Ob: COMMUNICATION_MODE. USER INPUT: Not valid user input, ERROR: Not valid user input. FOP Validation Interruption	PASSED

Table B.4: V-SCHEDULE Test Cases for VIPER Validation: expected and obtained results

List of Figures

1.1	IRIDE: European Earth Observation satellite constellation	1
1.2	NASA Johnson Space Center's Mission Control Centre	4
1.3	High Level FOS Architecture	6
1.4	FOS Architecture Comprehending a Satellite Simulator and a Deconstructed MCC	8
2.1	FOP Validation method for a manual case [5]	13
2.2	FOP validation method for an automated case [5]	14
2.3	OPSVAL Objectives and context of work [20]	14
2.4	Comparison between the OPSVAL Conceptual Map [20] and the procedure evaluation process map of [5]	15
2.5	Example of OPSVAL test cases definition [20]	16
2.6	OPSVAL Execution and Validation specifications [20]	17
2.7	ARVALIS FOP Validation Conceptual Map [22]	18
2.8	ARVALIS ECA Process [22]	19
2.9	OPSVAL Rule Description [20]	20
2.10	OPSVAL Validation Report summaries [20]	21
2.11	ARVALIS Validation Report in PDF format [22]	21
3.1	FOP Example in PRL [19]	26
3.2	SPELL Procedure Code View [23]	27
3.3	FOP Example in PLUTO [11]	28
3.4	PLUTO FOP Structure [11]	29
3.5	PLUTO FOP Execution Stages [11]	31
4.1	VIPER Requirements	42
4.2	VIPER and FOS units interfaced	44
4.3	VIPER Interfaces	46
4.4	VIPER Architecture	48
4.5	VIPER pluto_mib Mode Working Principle	50
4.6	VIPER mcs_interface Mode Working Principle	51

5.1	VIPER Python Functions	54
5.2	FOP MIB Consistency Check Functioning Logic	62
5.3	MQTT Broker - Client Connection [4]	65
5.4	VIPER-MCS-Satellite Interfaces	66
5.5	Analysis of MCS Interface Responses Handling logic: (a) mcs_interface function, (b) mcs_interface_input function	69
6.1	Satellite Simulator Use during VIPER Validation Test	92

List of Tables

3.1	PLUTO FOP Standard Commands	40
4.1	VIPER Operational Modes Summary	52
5.1	JSON FOP Step Dictionary	56
5.2	JSON FOP Required Inputs Dictionary	57
5.3	JSON FOP TC Dictionary	57
5.4	JSON FOP TC Parameters Dictionary	58
5.5	JSON FOP TM Dictionary	59
5.6	JSON FOP TM Event and If Dictionary	60
5.7	JSON Interface TM and TC Dictionaries	64
5.8	JSON Interface TC Parameter and TC Option Dictionaries	64
5.9	JSON Interface Responses from MCS	68
6.1	VIPER RoD Requirements	75
6.2	TMTC Configuration Change Test Cases	80
6.3	Operational Mode Change Test Cases	82
6.4	House Keeping Report Request Test Cases	84
6.5	On-Board Schedule Update Test Cases	86
A.1	VIPER Functional Requirements (First Set)	105
A.2	VIPER Functional Requirements (Second Set)	106
A.3	VIPER Functional Requirements (Third Set)	107
A.4	VIPER Interface Requirements	108
A.5	VIPER Operational Requirements	109
B.1	V-TMTC Test Cases for VIPER Validation: expected and obtained results	119
B.2	V-OPMODE Test Cases for VIPER Validation: expected and obtained results	120
B.3	V-HK Test Cases for VIPER Validation: expected and obtained results . .	121

B.4 V-SCHEDULE Test Cases for VIPER Validation: expected and obtained results	122
---	-----

Acknowledgements

Firstly, I would like to thank my academic supervisor, Professor Michèle Lavagna, for allowing me to delve deeper and deeper into her subject, increasing my already strong passion for space missions.

My heartfelt gratitude goes to my tutor, Edoardo Bruno, for his passionate guidance and support throughout this entire journey in Argotec. His unvaluable insights and mentorship have been a true reference point for me.

Furthermore, I would also like to express deep appreciation to Biagio Cotugno, for the trust placed in me and for encouraging me to strive for continual improvement.

In addition, I feel thankful to the entire Argotec Flight Control Team: in a world that previously seemed unknown, you have welcomed me not only as a colleague, but also as a friend. Especially, I warmly thank Francesco and Teodoro for their invaluable comradeship and support during the last months.

Sincere thanks go to all those friends who have accompanied me on this journey: to those who have shared this experience with me day by day, to those who have always supported me from afar.

Finally, the most significant and indispensable thanks is reserved to my Family. You are the mightiest sword and the safest shield of my life.

