



**Politecnico
di Torino**

Politecnico di Torino

Corso di Laurea Magistrale in Ingegneria Aerospaziale

A.a. 2021/2022

Sessione di Laurea Marzo/Aprile 2022

Application of MBSE to reverse engineering a rendezvous and docking space mission

Relatrice:

Prof.ssa Sabrina Corpino

Candidato:

Giuseppe Dinolfo, s271410

Correlatore:

Ing. Giorgio Ammirante

A mia madre e a mio padre

Abstract

Model-based Systems Engineering (MBSE) is a turning point for companies in the manufacturing and engineering sectors that in recent years have increasingly understood the need to digitize themselves, satisfy the growing demand to produce new technologies, supply quality products and services in a short time with low costs. Its use could also bring benefits to the development of CubeSat missions, which until now has been done through document-based approaches. This thesis aims to improve and enrich the first phases of the SROC (Space Rider Observer Cube) space mission project, a small satellite capable of rendezvous and docking, through the construction of models to describe its different areas and realize through reverse engineering an MBSE procedure for similar missions. To achieve this goal, an overview of the MBSE state of the art was initially outlined, researching the main methodologies, languages and tools, to understand what potential advantages or challenges could derive from this approach, focusing on the functionality of the software Capella, a tool that implements the Arcadia (Architecture Analysis and Design Integrated Approach) methodology and Valispace. These were selected after a simple trade-off study based on bibliography and practical experience and were used for the SROC mission. Capella was used to create models related to stakeholders and needs analysis, functional analysis, the construction of logical and physical architecture, Valispace, for the collection and management of requirements, to define the mission architecture, implementing the characteristics physical components, and to obtain, also thanks to the use of add-ons, mass, volume, power, energy and link budget. Finally, through a process of abstraction, a procedure was provided, usable as a guide, to carry out, in the best possible way, future model-based missions projects for small satellites.

The systems engineer, following this guide, will be able to develop, in a simple, quick and intuitive way, by making the appropriate modifications, the initial phases of his interest space project.

Table of Contents

1	Introduction	1
1.1	Context, objectives and outline of the research project.....	1
1.2	CubeSat	1
1.3	Rendezvous and docking	2
1.4	Space Rider	3
1.5	SROC Mission	4
1.6	ECSS and SROC design phases.....	4
1.7	Systems Engineering	7
2	Model-Based Systems Engineering.....	8
2.1.1	Definition of methodologies, process, tool, method	9
2.2	Methodologies, languages and tools.....	10
2.2.1	IBM Harmony for SE.....	10
2.2.2	Pattern-Based Systems Engineering (Systematica™ Methodology)	12
2.2.3	Vitech Model-Based System Engineering (MBSE) Methodology	12
2.2.4	NASA JPL State Analysis Methodology	14
2.2.5	The Systems Modelling Toolbox (SYSMOD).....	15
2.2.6	Object-Process Methodology.....	16
2.2.7	INCOSE Object-Oriented Systems Engineering Method (OOSEM)	18
2.2.8	The ARCADIA method	19
2.2.9	System Composer™	21
2.2.10	SYS-ML.....	21
2.2.11	Cameo System Modeler	22
2.2.12	MagicDraw.....	23
2.2.13	OCDT	23
2.2.14	Papyrus	24
2.2.15	Valispace.....	25
2.2.16	Choice of project implementation	26
3	SROC	28
3.1	Operational analysis	28
3.1.1	Stakeholders and needs phase o/A.....	28
3.1.2	Operational Analysis: Capella architectural elements and diagrams	30
3.1.3	Operational analysis phases delta A/B1	31
3.2	System Analysis	35
3.2.1	Concept of Operations, Mission Architectures and Functional Analysis phase o/A.....	35

3.2.2	System Analysis: Capella architectural elements and diagrams	38
3.2.3	System analysis phases delta A/B1.....	40
3.3	Requirements.....	47
3.3.1	Requirements phase o/A.....	47
3.3.2	Valispace Requirements	48
3.3.3	Requirements delta A/B phases	49
3.4	Logical Architecture.....	50
3.4.1	Logical Architecture phase o/A.....	50
3.4.2	Logical Analysis: Capella architectural elements and diagrams	51
3.4.3	Logical analysis phases delta A/B1.....	52
3.5	Physical Architecture	53
3.5.1	Physical Architecture phase o/A.....	54
3.5.2	Physical Analysis: Capella architectural elements and diagrams	54
3.5.3	Valispace components module	55
3.5.4	Physical architecture phases delta A/B1	56
3.6	Budgets	58
3.6.1	Power budget and Energy budget phase o/A.....	59
3.6.2	Link budget phase o/A	60
3.6.3	Power budget phases delta A/B1	62
3.6.4	Energy budget phases delta A/B1.....	63
3.6.5	Link budget phase delta A/B1	65
4	ModelSat	66
4.1	Advice for MBSE stakeholders and needs identification.....	66
4.2	Advice for MBSE Concept of Operations.....	68
4.3	Advice for MBSE requirements management	69
4.4	Advice for MBSE logical analysis.....	71
4.5	Advice for MBSE physical analysis	72
4.6	Power budget	75
4.7	Energy budget.....	77
4.8	Link budget	83
5	Conclusions	86
	Annex A: Capella diagrams	89
	Annex B: Energy budget script.....	94
	Bibliography	106

List of tables

Table 1: Typical project lifecycle [15].....5

Table 2: Stakeholders' needs [7]..... 29

Table 3: SROC Baseline Mission – ConOps 1 - Phases and Scenarios description [7] .. 36

Table 4: SROC mission architecture [7]37

List of figures

Figure 1: Space Rider [9]	4
Figure 2: The PMTE Elements and Effects of Technology and People [25]	10
Figure 3: OMG SysML™ Representation of Service Request-Driven Approach [25]	11
Figure 4: A summary view of the S* metamodel [30]	12
Figure 5: The state-based control architecture [34].....	15
Figure 6: Zooming into System Developing [25]	17
Figure 7: OOSEM Activities and Modeling Artifacts [25]	19
Figure 8: The main engineering levels of Arcadia [24]	20
Figure 9: System Composer workflow [42]	21
Figure 10: OCDT Architectural Overview [47]	24
Figure 11: Difference between classical MBSE with SysML (left) and MBSE with Arcadia/Capella (right) [24].....	27
Figure 12: Stakeholders' mapping [7].....	28
Figure 13: From the left symbols of “Operational Capabilities”, “Operational Activity”, “Operational Entity”, “Operational Actor”, “Operational Interaction”, “Operational Process” [51]	30
Figure 14: OEB.....	31
Figure 15: OCB.....	32
Figure 16: Expand CubeSat mission portfolio OAIB.....	32
Figure 17: Development of key technology OAIB.....	32
Figure 18: ESA Dissemination ES	33
Figure 19: Governments Dissemination ES	33
Figure 20: OAB.....	34
Figure 21: Functional tree phase o/A [7]	38
Figure 22: From the top left symbols of “System”, “System Actor”, “System Mission”, “System Capability”, “System Function”, “Functional Exchange”, “Component Exchange”, “Physical Link”, “Functional Chain” [51]	39
Figure 23: CSA.....	40
Figure 24: MB.....	41
Figure 25: LEOP Capability property window	42
Figure 26: Observe and retrieve ConOps MCB	42
Figure 27: Observe ConOps MCB.....	42
Figure 28: End of Mission (observe and retrieve) - Decommissioning (EOM) SDFB...	43
Figure 29: End of Mission (observe) - Decommissioning (EOM) SDFB	43
Figure 30: Interfaces SAB	44
Figure 31: Mission SAB (1)	45
Figure 32: Mission SAB (2)	46
Figure 33: Requirements examples.....	49
Figure 34: Valispace organization and example of requirements.....	50
Figure 35: From the left symbols of “logical function”, “logical component”, “functional exchange”, “component exchange” [51]	52
Figure 36: LAB.....	53
Figure 37: From the left symbols of “physical function”, “behavioral physical component”, “node physical component”, “physical connection”, “physical path”[51]..	55
Figure 38: PAB	57
Figure 39: Valispace physical tree (left), satellite mass and volume budget (right)	58
Figure 40: Example of Tyvak Power Budget during Downlink orbits phase o/A [52] ..	59

Figure 41: Tyvak link budget phase o/A [52]	62
Figure 42: SROC flow chart for energy budget phase delta A/B1	64
Figure 43: Flow chart general procedure stakeholders and needs analysis.....	67
Figure 44: From the left Valispace organization of proposal 1, proposal 2 and proposal 3	71
Figure 45: Valispace mission architecture (left) and thruster alternative container (right).....	73
Figure 46: Electric cold gas thrusters alternatives	74
Figure 47: Cold gas and Electric thrusters comparison	75
Figure 48: Mode example for components (left) and mission phases (right).....	76
Figure 49: Mode selection matrix	76
Figure 50: Import data from STK (Matlab script)	77
Figure 51: Example codes for vector selection (Matlab script)	78
Figure 52: Comparisons for vector selection (Matlab script).....	78
Figure 53: Valispace connection.....	78
Figure 54: Cosine loss variation and Sunpointing input power computation (Matlab script).....	79
Figure 55: Maximum Sunpointing input power computation (Matlab script)	80
Figure 56: Nadir pointing input power computation (Matlab script).....	80
Figure 57: Total state of charge plot (Matlab script)	81
Figure 58: Valispace parameters export (Matlab script)	81
Figure 59: Flow chart for energy budget determination	82
Figure 60: Example of Valispace architecture for the ground segment (left), space segment (centre), operations (right)	83
Figure 61: Cost down OAIB	89
Figure 62: Dissemination OAIB	89
Figure 63: Safety up OAIB.....	90
Figure 64: Cost down OES.....	90
Figure 65: Safety up OES.....	91
Figure 66: Launch and Early Operations Phase (LEOP) SDFB	91
Figure 67: Commissioning and Performance Verification Phase (CPVP) SDFB	92
Figure 68: Proximity Operations Phase (POP) SDFB	93
Figure 69: Docking and Retrieval Phase (DRP) SDFB.....	93

Abbreviations

A	Area
ADCNS	Attitude Determination & Control Navigation System
ADCS	Attitude Determination & Control System
AIT	Assembly Integration & Test
API	Application Programming Interface
ARCADIA	Architecture Analysis & Design Integrated Approach
ASI	Agenzia Spaziale Italiana
avg	average
BCE	Battery Charge Efficiency
BDE	Battery Discharge Efficiency
BER	Bit Error Rate
<i>C</i>	Carrier
CDH	Command & Data Handling
CEO	Chief Executive Officer
ConOps	Concept of Operations
COTS	Off-The-Shelf Component
CPVP	Commissioning and Performance Verification Phase
CSA	Contextual System Actor
DEP	Deployment Phase
DMP	Docking and Mating Phase
<i>DOD</i>	Depth of Discharge
DoD	Department of Defence
DOCKS	Docking System
DRP	Docking and Retrieval Phase
DST	Domain Specific Tool
<i>E</i>	Energy
E	Environment
ECSS	European Cooperation for Space Standardization
EIRP	Effective Isotropic Radiated Power
EOL	End of Life
EOM	End of Mission
EOP	Early Operations Phase
EPBS	End Product Breakdown Structure
EPS	Electric Power System
ES	Entity Scenario
ESA	European Space Agency
<i>f</i>	frequency
<i>F</i>	Noise figure

FFBD	Functional Flow Block Diagram
G	Gain
GPS	Global Positioning System
HDR	High Data Rate
HOP	Holding Phase
HP	Holding Point
HTML	Hypertext Markup Language
IBC	Initial Battery Capacity
IBE	Initial Battery Energy
ID	Identity Document
IDE	Integrated Development Environment
IMU	Inertial Measurement Unit
IRL	Isotropic Receive Level
INCOSE	International Council on Systems Engineering
JPL	Jet Propulsion Laboratory
KSAT	Kongsberg Satellite Service
<i>L</i>	Loss
LAB	Logical Architecture Blank
LDR	Low Data Rate
LEO	Low Earth Orbit
LEOP	Launch and Early Operations Phase
LVLH	Local Vertical Local Horizontal
M	Method
max	maximum
MB	Mission Blank
MBSE	Model Based System Engineering
MCB	Mission Capabilities Blank
MCC	Mission Control Center
MDD	Model Driven Development
MDR	Mission Definition Review
min	minimum
MPCB	Multi-Purpose Cargo Bay
MPCD	Multi-Purpose CubeSat Dispenser
MPPT	Maximum Power Point Tracking
<i>N</i>	Noise
N	Number
NASA	National Aeronautics and Space Administration
OAB	Operational Architecture Blank
OAIB	Operational Activity Interaction Blank

OAP	Orbital Average Power
OBS	Organizational Breakdown Structure
OC	Operational Capabability
OCB	Operational Capabilities Blank
OEB	Operational Entity Breakdown
OMG	Object Management Group
OO	Object-Oriented
OOSEM	Object-Oriented Systems Engineering Method
OPD	Object Process Diagram
OPL	Object-Process Language
OPM	Object-Process Methodology
<i>P</i>	Power
P	Process
PAB	Physical Architectural Blank
PBSE	Pattern-Based System Engineering
PCBA	Printed Circuit Board Assembly
PDF	Portable Document Format
PDR	Preliminary Design Review
POP	Proximity Operations Phase
PRNS	Proximity Relative Navigation Subsystem
Prop	Propulsion
PRR	Preliminary Requirements Review
REP	Retrieval Phase
Rev	Revision
RF	Radio Frequency
RIA	Rich Internet Application
RTF	Rich Text Format
RV&D	Rendezvous and Docking
RVP	Rendezvous Phase
RW	Reaction Wheel
SA	State Analysis
SAB	System Actor Blank
SFBD	System Function Breakdown Diagram
SDFB	System Data Flow Blank
SDL	System Definition Language
SE	System Engineering
SOC	Sum of Children
<i>SOC</i>	State of Charge
SQL	Structured Query Language

SR	Space Rider
SROC	Space Rider Observer Cube
SROP	Space Rider Observation Phase
SRR	System Requirements Review
SSO	Sun Synchronous Orbit
STK	Systems Tool Kit
SysML	Systems Modeling Language
SYSMOD	System Modeling Toolbox
<i>T</i>	Temperature
T	Tool
TBD	To be determined
TCS	Thermal Control System
TT&C	Telemetry, Tracking and Command
UHF	Ultra High Frequency
UML	Unified Modeling Language
V&V	Verification & Validation
WBS	Work Breakdown Structure
WSE	Walking Safety Ellipse

1 Introduction

1.1 *Context, objectives and outline of the research project*

The thesis is carried out in the framework of the Space Rider Observer Cube (SROC) mission, funded by the European Space Agency (ESA) and by the Agenzia Spaziale Italiana (ASI). The mission consists of a CubeSat that will be carried into orbit stowed inside Space Rider (SR), new reusable ESA space transport, and will perform, after the deployment, formation flying, observation of the mothercraft, and at the end, rendezvous and docking with the mothercraft herself.

The research aims at assessing whether and how the Model-Based Systems Engineering (MBSE) approach can help during the early design phases of a space mission. The thesis tries to achieve this through five chapters.

The first chapter focuses on providing to the reader pieces of information relating to the context in which the thesis will be located, the world of CubeSats, SR, SROC missions and systems engineering.

The second chapter will show an overview of the main methodologies, languages, and tools that support MBSE, analysing what potential benefits or challenges might come from this approach. The environments that best fit the mission features to develop, after a trade-off study, based on bibliography and practical experience, will be selected.

In the third chapter the thesis, furthermore, will focus on the development of an MBSE model, through the selected tools, for a rendezvous and docking mission between a small satellite (SROC), which will be the main character of the study, and an orbiting platform (SR). The system will be implemented considering requirements, operational, functional, and physical aspects.

The fourth chapter will be based on the creation of a model-based procedure, provided through a process of abstraction from the previous model, that can be understood as a canvas or advice, to carry out in the best way future missions designs for small satellites during the first phases of the project.

Finally, the fifth chapter will concern the conclusions that will be drawn from the work carried out.

1.2 *CubeSat*

A CubeSat is a miniaturized spacecraft belonging to the class of nanosatellites, born in 1999 from the collaboration between Professor Jordi Puig-Suari, of California Polytechnic State University at San Luis Obispo, and Professor Bob Twiggs, of Stanford University, with the purpose of “creating a platform for education and space exploration” [1], [2]. For their construction, a standard is followed, called the “CubeSat standard”, which defines the external dimensions of the satellite within several cubic U units of $10\text{ cm} \times 10\text{ cm} \times 11.5\text{ cm}$ [3]. Each unit has on average a mass that can reach up to 1.33 kg [1].

Today we can speak of a real “CubeSat movement” as universities and companies collaborate in the design of these satellites trying to integrate increasingly technological scientific payloads and to draw as much information as possible from the missions previously carried out [2].

The CubeSats are modular and integrated systems whose components are constituted most of the time by COTS, which guarantee to access economically and in a simple way in space through small sizes, lower costs, lower production and development times [3], [4]. Generally, a project for CubeSats is carried out, according to ESA estimates, in two years. The entire mission, on the other hand, can have a cost ranging from fifty thousand dollars up to several million dollars depending on its complexity [2], [3].

Space agencies are investing in this class of vehicles as, in addition to the various benefits listed above, it allows to do in-orbit demonstrations of technological components and new flight techniques (formation flight, rendezvous and docking), to make scientific investigations, to lead to miniaturization of components and the creation of highly integrated subsystems, to carry out measurements and observations in the vicinity of other orbiting bodies, to increase the possibility of exploring the solar system and to generate redundancy of functionality when in the fleet, to be sent into orbit through containers inside a mothercraft reducing risks and increasing the chances of launch [1], [3].

The presence of many subsystems and payloads in small spaces, the need to limit consumption and the reduced budget means that systems engineering, also through the “model-based”, can be of great importance for the development of the mission [2], [4].

1.3 Rendezvous and docking

The rendezvous and docking phases take place between a tracker satellite, which in most cases plays an active role, called “chaser”, and a satellite (usually passive) or a celestial body, which can be reached, called “target”, and are carried out through different orbital manoeuvres and controlled trajectories [5].

After having brought the chaser to the desired orbital plane (launching), to perform the rendezvous manoeuvre, a “phasing”, a far-range rendezvous, a close-range rendezvous and a final approach manoeuvre are usually accomplished. In most cases, a reference system centred on the earth is used for launching and phasing phases, while for the others a relative reference system Local-Vertical-Local-Horizontal (LVLH) is used, with the origin coinciding with the centre of mass of the target [6].

The phasing manoeuvre is realized by the chaser activating the navigation sensors. The goals of this phase are to adjust the phase angle between the two satellites, to start relative navigation by reducing any differences between the orbital planes and to decrease the difference in altitude between the two vehicles, at diverse altitudes at the beginning of the phase [6].

The far-range rendezvous manoeuvre, also known as “homing”, is necessary for the chaser to achieve the target orbit, reducing its relative speed and “trajectory dispersions”, “synchronizing the mission timeline” [6], [7]. At the end of this phase, an “hold point” is reached, a point in which the satellite can remain without consuming propellant. The distance between the two satellites and the position of the chaser in the hold point depends on the requirements imposed by the target [5].

The close-range rendezvous phase is performed to further reduce the distance between the two satellites, reaching a position, an attitude, a relative speed, and an angular rate necessary to execute the final approach. The main differences with the far-range rendezvous manoeuvre are related to the position and final speed. This phase can be carried out in different ways by manoeuvres along the V-bar or R-bar axes [5],[7].

After these stages, the “final approach” begins, in which the chaser follows a straight or almost straight-line trajectory, using closed-loop controls, inside a safety conical region called “approach cone”, to reach the mating necessary conditions (position, speed, attitude and angular speeds) [6].

Depending on the satellites docking or berthing can be executed to mate the vehicles. Docking if the two vehicles come into contact with capture interfaces, berthing if the mating occurs thanks to a robotic arm, which one of the two satellites can be equipped with [5].

In the case of docking, any misalignments and rebound effects related to the impact between the two interfaces must be taken into account. When the satellites come into contact rightly, rigid mechanical and electrical connections can be made [5].

1.4 Space Rider

Space Rider is an “automated”, “reusable”, “independent” and “economical” integrated space transport vehicle without a crew, with a load capacity of 800 *kg*, useful to reach low Earth orbit and return to Earth [8],[10].

SR has been conceived as the heir of the IXV shuttle thanks to an agreement signed by ESA, Thales Alenia Space and the European Launch Vehicle consortium, of which Avio and ASI belong as well as manufacturers, research centres and universities [8], [9].

SR will reach low Earth orbit through the ESA VEGA C vector starting from the Kourou Spaceport, located in French Guiana and will be able to remain in orbit for up to two months and then re-enter the Earth's atmosphere and land. The vehicle could be refurbished and used up to six times [8],[10].

Following the “User Guide” released by ESA, the vehicle was designed to guarantee, for payloads in the cargo bay, “prolonged exposure to space environments in microgravity presence, an opening hold able to point payloads towards the Earth or deep space, very accurate pointing capability (Nadir and Zenith), reduced safety constraints with respect to manned operations, the possibility of using different flight and ground services, lower flight times and costs than competitors”.

Furthermore, the main applications for which it has been designed are:

- demonstration and validation of technology for robotic exploration, Earth observation, in-orbit infrastructure maintenance, telecommunications, and scientific studies;
- experiments in the presence of microgravity for pharmaceutical, biomedical, and biological purposes;
- satellite inspections;
- educational missions;
- inquiry on access to and return from space [8], [10], [11].

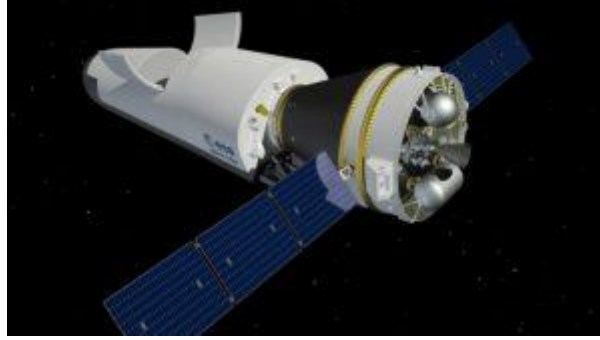


Figure 1: Space Rider [9]

1.5 SROC Mission

The SROC (Space Rider Observer Cube) mission statement explains how this consists in the distribution and management of “a CubeSat in LEO to support Space Rider operations through multispectral and visual observations, taken in the vicinity of the vehicle, during the orbital phase” with the additional purpose of “improving the CubeSats capabilities in the domain of proximity operations”[7].

As highlighted in the document “Patrioli L., Corpino S., D1.1 Assessment of the SROC mission and preliminary functional specification, 2020”, the mission, therefore, has the following main objectives: observation through unprecedented images of SR and the demonstration of technologies and functions to carry out formation flight through a focus on “proximity navigation, guidance and control capabilities and communication architecture”. In addition, the mission proposes to achieve an optional objective which consists of the “ability to recover and reuse the CubeSat in orbit” [7].

In general, the concepts behind the SROC mission are the transport by Space Rider and the release of a 12U CubeSat equipped with multispectral cameras and advanced technology to carry out the mission safely, a subsequent phase of mothercraft observation and an optional phase of re-entry, through the use of a docking mechanism, into a dispenser installed on SR, called MPCD (Multi-Purpose CubeSat Dispenser), the return to the ground, avoiding any collisions throughout the mission [7].

The development plan for the entire project foresees a flight opportunity in mid-2023 [12].

1.6 ECSS and SROC design phases

Project planning and implementation encompass all the carried-out processes to plan and execute a space project from initiation to completion at all levels in the customer-supplier chain in a coordinated, efficient, and structured manner. Usually, a space project involves several actors with different characteristics, and it lasts for many years. For these reasons, it is necessary to implement a structured process to execute the program [13].

The term “Systems engineering life cycle” is used in SE to describe all the processes involved in the evolution of a system, starting from its conception and ending with its disposal.

The INCOSE SE Handbook defines a life cycle as “the series of stages through which something (a system or a manufactured product) passes”. “The life cycle of any system must encompass not only the development, production, utilization and support stages

but also provide early focus on the retirement stage when the decommissioning and disposal of the system occurs”[14].

Various life cycle models have been defined by key players like the European Space Agency (ESA), the National Aeronautics and Space Administration (NASA), the US Department of Defense (DoD).

Following the ECSS-M-ST-10C_Rev.1 (6 March 2009) by ESA a life cycle mission is composed of seven phases, closely linked to activities on system and product level:

- phase O – Mission analysis/needs identification;
- phase A – Feasibility;
- phase B – Preliminary Definition;
- phase C – Detailed Definition;
- phase D – Qualification and Production;
- phase E – Utilization;
- phase F – Disposal;

As can be seen in Table 1 project phases include project reviews (e.g. MDR, PRR, etc.) that determine the readiness of the design to move to the next phase [15].

Activities	Phases						
	Phase O	Phase A	Phase B	Phase C	Phase D	Phase E	Phase F
Mission/Function	MDR		PRR				
Requirements	SRR			PDR			
Definition			CDR				
Verification				QR			
Production					AR ORR		
Utilization						FRR CRR ELR LRR	
Disposal							MCR

Table 1: Typical project lifecycle [15]

Phases O, A and B comprise the delineation of system functional and technical requirements and the identification of system concepts based on stakeholders' needs, an elaboration of initial risk assessments, the description of the activities and resources to be used, and the beginning of pre-development activities.

Phases C and D are focused on the development and qualification of the space and ground segments. A key point of this stage is to implement and integrate the initial systems and verify and validate systems such that they can be produced.

Phases E comprises the activities related to launching, commission, utilisation and maintenance of the orbital elements and associated ground segment elements.

Phases F comprises activities to safely dispose all products launched into space as well as ground segment [13].

The Vee model, defined in INCOSE SE Handbook as “a sequential method used to visualize various key areas for SE focus”, provides a useful illustration of the SE activities during the life cycle stages [14].

Phase O/A studies have already been faced for the SROC mission. They were conducted with the objectives of “identifying the needs and requirements for a mission composed by one or more nanosatellites flying in the vicinity of a target vehicle, determining new technological solutions, architectures and mission concepts, defining the development plan of the project until completion” [12].

The entire work led to the definition of the mission and the system through the identification of the main challenges to realize (proximity navigation, propulsion, observation, deployment, and recovery mechanism) [12].

The thesis, instead, focuses on the new phases that must be developed by the project: a delta phase A and a phase B1, trying to propose “model-based” solutions for the management of data, files, requirements, interfaces, components and budgets.

The main tasks that must be performed by the entire design team, considering the ECSS-M-ST-10C_Rev.1 and the SoW published by ESA, are:

- “Confirm technical solutions for the system and operations concepts and their feasibility with respect to programmatic constraints.
- Conduct trade-off studies and select the preferred system concept, together with the preferred technical solutions.
- Establish a preliminary design definition for the selected system concept and the retained technical solutions.
- Determine the verification program including model philosophy.
- Identify and define external interfaces.
- Prepare the next level specification and related business agreement documents.
- Initiate pre-development work on critical technologies or system design areas.
- Conduct reliability and safety assessment.
- Update the risk assessment” [12], [15].

The project reviews associated with Phase B are:

- The SRR (System Requirements Review) where updated technical requirements specification will be released, the preliminary design definition and the preliminary verification plan will be assessed.
- The PDR (Preliminary Design Review) where it will be verified that the preliminary design meets the project requirements and a series of deliverables are realized, such as system requirements and final management, engineering, and product assurance plans, product tree, WBS, the specification tree and the verification plan [15]. This review will be faced at the end of a subsequent phase of the project, other than the B1.

1.7 Systems Engineering

The INCOSE SE Handbook defines Systems Engineering (SE) as “an interdisciplinary approach and a means to enable the realization of successful systems” that “deals with designing and managing systems over their lifecycle, starting from the system conceptualization phase till its disposal. It focuses on defining customer needs and required functionality early in the development cycle, documenting requirements, and then proceeding with design synthesis and system validation while considering the complete problem: operations, cost and schedule, performance, training and support, test, manufacturing, and disposal. SE integrates all the disciplines and speciality groups into a team effort forming a structured development process that proceeds from concept to production and operation. SE considers both the business and the technical needs of all customers with the goal of providing a quality product that meets the user needs”[14].

Reading these sentences, it is necessary to clarify the term “system”. A “system” can be defined as a combination of different elements that work together, to perform a certain required function, producing results not obtainable by the single element. The aerospace systems, for example, are complex “machines” composed of thousands of parts with millions of interactions that operate to fulfil some necessary tasks for society (i.e. transmit messages, provide energy, transport people and cargo) [13], [16].

The origin of systems engineering is to be found in “systems thinking”, a matchless perspective of reality that leads to approaching a problem or a simple question as if this were a system. A perspective thanks to which a “system thinker”, through discovery, learning, diagnosis, and dialogue, can identify and manage in everyday life the parts and the relationships that make up the present systems and also the systems as a whole [14], [16].

Systems engineering is therefore a holistic approach, based on tradeoffs, that wants to favour the optimization of the project as a whole and not one system rather than another, which aims to achieve the right balance between operational, economic, and logistical factors. And this happens to reach an economically advantageous and quality solution, respecting the imposed requirements, being able to satisfy the customer, not neglecting the business and technical needs [2], [16], [17].

As NASA Systems Engineering Handbook states, the contributions of various disciplines such as electrical engineering, structural engineering, energy engineering, mechanism design, human factors engineering, and others are evaluated and equilibrated “to produce a coherent whole that is not dominated by the perspective of a single discipline” [18].

Systems engineering consists, in the end, in a succession of phases that include numerous iterations of functional analysis, synthesis, optimization, definition, design, testing, and evaluation; to ensure the compatibility of the different interfaces between the various subsystems and between the system and the environment in which it will operate.

Today SE is used in various fields and manages increasingly complex systems that, over the years, have required the use of a new formalized method, different from document-based (where the leading actors are physical documents used to manage and pass system information), to carry out the activities: the model-based approach. A method that has its roots in software engineering and uses computerized documents rather than physical [16].

2 Model-Based Systems Engineering

In the last twenty years, the digitalization, the need to satisfy the technological demands of industries, the competition between different companies in the engineering and manufacturing sector have increasingly pushed the various organizations to improve themselves, to provide products and services of higher quality and to launch products on the market quicker reducing costs. These objectives have prompted system engineering, as previously mentioned, to invest in the advantages brought by the model-based approach rather than the traditional document-based one [19], [20].

The document-based method, in particular, focuses on the generation of information or specifications concerning the system in the form of documents, which can reach very large numbers for complex projects, i.e., a satellite design. The vastness of documents and the complexity of the pieces of information they contain lead in most cases to complications related to their management and their updating. Indeed, it is necessary to avoid different people working simultaneously on different update releases of the same document, eliminating communication problems [14], [21].

As E. Burger, in his doctorate thesis “A conceptual MBSE framework for satellite AIT planning”, writes: “sometimes different documents carry the same information, and sometimes both pieces of information do not match, which may cause unexpected outcomes”.

The model-based system engineering, instead, is defined by INCOSE Systems Engineering Vision 2020 as “the formalized application of modelling to support system requirements, design, analysis, verification, and validation activities beginning in the conceptual design phase and continuing throughout development and later life cycle phases”[14].

MBSE formalizes the practice of systems engineering using coherent, integrated, and complete computerized models [22] of physical systems and processes [23], that represent an abstract vision of reality [21] and lead to several improvements.

The advantages associated with the MBSE are different, among them we can find:

- the optimization of communication between designers with different backgrounds but also between designers and customers thanks to better data sharing and better exchange of information [14], [21];
- an improvement in the management of system complexity, with a consequent understanding of the impacts associated with any changes, minimizing the risks facing solution limitations [14], [21], [24];
- the possibility of providing a unique system model that guarantees higher product quality [14].;
- an improvement of productivity, with a reduction in terms of time and costs, thanks to the recycling of the created models and the traceability of requirements [14], [24];
- the ability to teach and learn MBSE concepts more easily thanks to the use of standardized models [14].

2.1.1 Definition of methodologies, process, tool, method

The implementation of MBSE, to design successfully a project, is based on applying the right methodology, consisting of “processes”, “methods” and “tools”, for which, to understand the meaning, a clear definition of all these terminologies is needed.

Indeed, with reference to the study by A. Albers and C. Zingel, “Challenges of model-based systems engineering: A study towards unified term understanding and the state of usage of SysML”, it is essential to define the significance attributed to these terms as there are many “ambiguities” and “semantic discrepancies” between “academia” and “industry” [16], [25], [26].

Clear definitions of these terms are provided in J. Estefan, “Survey of Model-Based Systems Engineering (MBSE) Methodologies, 2008” according to which:

- “A Process (P) is a logical sequence of tasks performed to achieve a particular objective. A process defines “WHAT” is to be done, without specifying “HOW” each task is performed. The structure of a process provides several levels of aggregation to allow analysis and definition to be done at various levels of detail to support different decision-making needs” [25].
- “A Method (M) consists of techniques for performing a task, in other words, it defines the “HOW” of each task. At any level, process tasks are performed using methods. However, each method is also a process itself, with a sequence of tasks to be performed for that particular method. In other words, the “HOW” at one level of abstraction becomes the “WHAT” at the next lower level” [25].
- “A Tool (T) is an instrument that, when applied to a particular method, can enhance the efficiency of the task; the purpose of a tool should be to facilitate the accomplishment of the “HOWs”. In a broader sense, a tool enhances the “WHAT” and the “HOW” [25]. MBSE is mainly based on tools, which in most cases are software, allowing creating models using appropriate languages. Today the market permits to select, given the great variety, the tool that best suits designers’ needs, considering the “strengths” and “weaknesses” of the software. It is fundamental to consider there is no possibility to adopt the MBSE without a specific tool [21], [27].

Thanks to these definitions, it is possible to characterize the methodology as the set of processes, methods and tools that allows model-based techniques to be applied in the different project phases.

A methodology is defined as “a “recipe” to be applied to a class of problems that have something in common” [25].

J. Estefan also remembers how the concept of “environment” (E) is related to the definitions previously provided. The environment is understood as everything that surrounds us: “external objects”, “conditions” (i.e. “social”, “cultural”, “personal”, “physical”, “political”) and “factors” that can influence the choices of a community [25]. Therefore, the MBSE environment must allow the integration of the tools and methods used in a project, enabling the “WHAT” and the “HOW” [16], [25].

Figure 2 is a visual diagram showing the PMTE elements (process, methods, tools, environment) and their links with technology and people, elements to always consider in every project [25].

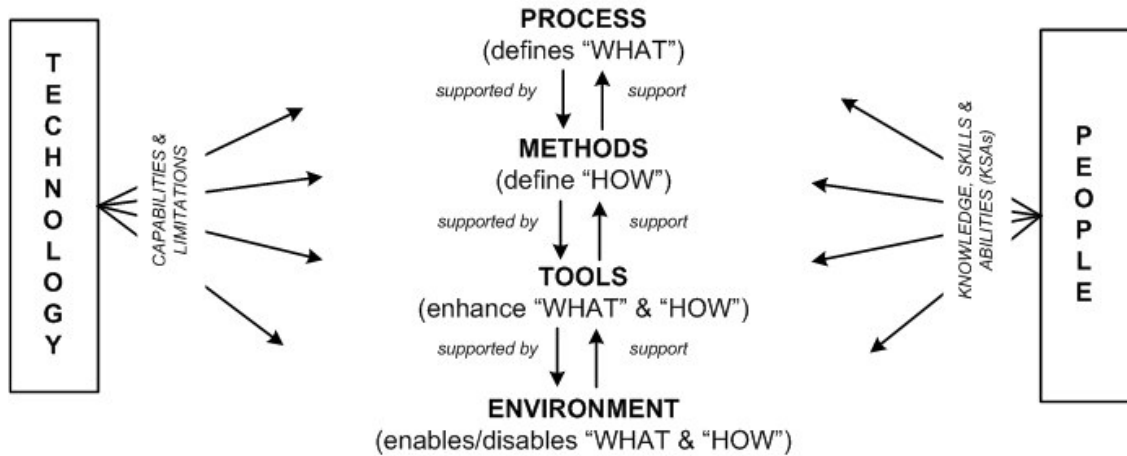


Figure 2: The PMTE Elements and Effects of Technology and People [25]

2.2 Methodologies, languages and tools

After having made the necessary clarifications and definitions, it is now possible to describe the characteristics of the main methodologies, tools and languages present in the literature.

2.2.1 IBM Harmony for SE

IBM Harmony-SE is a “subset” of the broader software development methodology IBM Harmony. IBM Harmony-SE and IBM Harmony were created at I-Logix Inc., a brand now acquired by IBM Corporation, which was a leading provider of embedded modelling tools [25].

The key objectives of Harmony for Systems Engineering, following the manual “Systems Engineering Best Practices with Rational Solution for Systems and Software Engineering”, are:

- “Identification and derivation of the required system functions”.
- “Identification of the associated system modes and states”.
- “Allocation of the identified system functions and modes/states to the subsystem structure” [28].

The modelling takes place through iterative cycles following the “Vee” life cycle model for the design of systems, carrying out phases of requirements analysis, functional analysis, and architectural design [25], [27].

The methodology modelling approach is based on the “service request” between blocks. The system structure is described employing SysML “structure diagrams” where the blocks are the fundamental elements of the structure and the communication among them takes place via messages (“service requests”) [25]. An explanatory example can be seen in the next image (Figure 3).

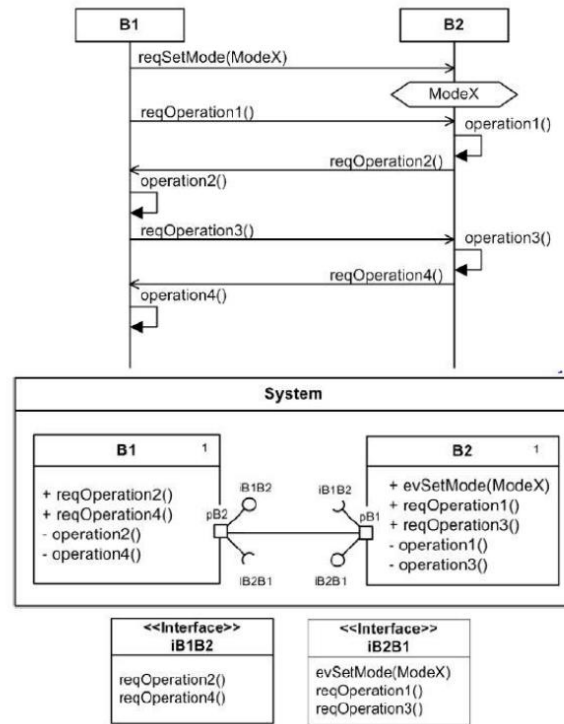


Figure 3: OMG SysML™ Representation of Service Request-Driven Approach [25]

“The main emphasis” of IBM Harmony-SE “is on the identification and allocation of a needed functionality and state-based behaviour, rather than on the details of its functional behaviour”.

In the requirements analysis phase, the inputs provided by the stakeholders are transformed into system requirements that indicate what the system must do and how well it must work. The functional analysis phase has its focus on describing the operations that the system will have to carry out based on the functional requirements previously identified. The design synthesis, on the other hand, focuses on the definition of a physical architecture, which has to respect the imposed constraints and the requirements, performing the desired functions [28].

IBM Harmony was designed to be vendor and tool neutral, although the primary tool supporting this methodology is IBM Telelogic Rhapsody.

To help systems engineers and project managers use this methodology, the developers have published a Harmony-SE / SysML Deskbook [25].

The tool supports UML and SysML languages and allows different organizations to manage the complexity in the design of models and systems. This opportunity is provided as it is possible to share the models, created during the entire life cycle, among all the designers and to validate them continuously, carrying out rapid simulations that can solve the produced errors. The tool also offers the possibility of integrating the MathWorks Simulink software.

In general, the main functions offered by IBM Rhapsody products are:

- the analysis and processing of design requirements using a low-cost environment that allows the creation of system specifications, parametric evaluations, interface design documents, and test cases;

- a rapid transition from the project to its implementation through the use of a graphic language and codes such as C++ and C or Java, providing the possibility of their reuse;
- the automation of project reviews through a common repository between designers, suppliers and customers, speeding up “decision-making processes” and improving product quality;
- the creation of prototypes and simulations to validate the requirements and the architecture;
- a highly integrated environment [29].

2.2.2 Pattern-Based Systems Engineering (Systematica™ Methodology)

Following the description provided in the document “Model-Based System Patterns for Automated Ground Vehicle Platforms”, Pattern-Based Systems Engineering (PBSE) is an MBSE methodology capable of producing, through the use of complete and coherent models, systems ten times more complex with ten times less effort on modelling, through the work of a community of people ten times larger than that made up of systems experts alone. These advantages are possible because the PBSE uses the concept of “metamodel” which is “a model of other models, a framework or a plan governing the models it describes, which can be represented in SysML, database tables or other languages”. The used models refer to already existing models, previously made by users, which exploit a “learning curve” already underway, which always tends to improve thanks to the results learned over time.

In particular, the Systematica™ Methodology uses S* Metamodel (Figure 4), a model useful for the “description of requirements, projects and other information such as verification and failure analysis”, and S* Pattern, “formal” and “large scale” models which are based on S* Metamodel with the ability to be reusable and configurable [30].

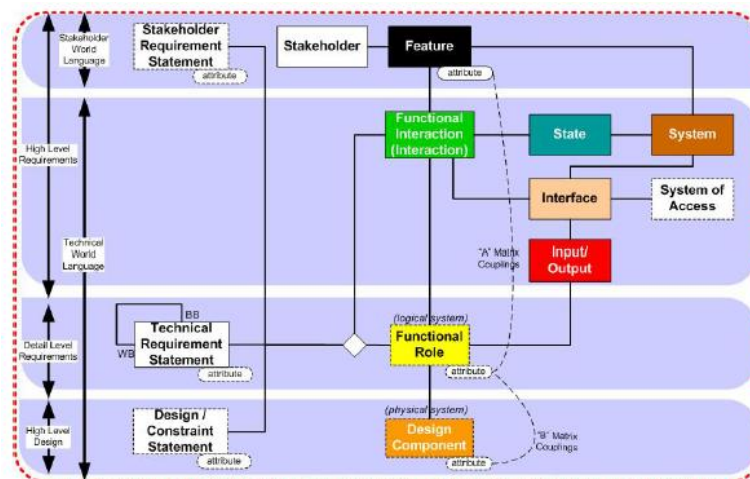


Figure 4: A summary view of the S* metamodel [30]

2.2.3 Vitech Model-Based System Engineering (MBSE) Methodology

The STRATA MBSE or Vitech MBSE is a methodology, developed by the company Vitech Corporation, offered and explained to the public, together with the CORE® product suite, through various tutorials produced by its CEO James E. Long [25].

This methodology focuses in the early phase on the analysis and comprehension of the more abstract levels of design by studying the interactions between the requirements, the system, the environment and, architecture, from which “functional behaviours” are

extracted. The latter are assigned to the physical components constituting the architecture that is tested to verify compliance with the requirements [16], [31].

The high-level vision allows defining the framework within which the project will evolve, providing to the team useful information for making intelligent detailed decisions [31]. Precisely, as Estefan writes, there are four primary areas of interest, contained in a common repository, which are simultaneously dealt with by STRATA MBSE:

- functional/behavioural analysis;
- analysis of origin requirements;
- validation and verification of the project;
- architecture/synthesis;

And for each of these, it is possible to identify an associated domain of activities called “Process Domain” [25].

The artefacts of the model are managed in the Vitech MBSE methodology through a common structured language that deals with syntax and semantics: the MBSE System Definition Language (SDL). This helps improve models and documents traceability and also the communication between team members.

A key concept for the Vitech methodology is how to carry out the project that shall be executed first horizontally and then vertically, by the means of a process called the “Onion Model”. An “incremental process” in which the different activities, through a simultaneous execution level after level, are gradually implemented passing from a maximum stage of abstraction to that of greater detail.

This model takes this name precisely because every time “the SE team successfully completes one level of system design, they “peel off a layer of the onion” and start to explore the next layer, when the team reaches the desired level of detail (the centre), their design is complete” [25].

The fundamental characteristics of this approach are therefore completeness and convergence; when these are not achieved it is, in fact, necessary to review the work executed by the designers [25].

The STRATA Vitech methodology despite being considered “independent of the tools” can be implemented thanks to a set of tools included in the CORE® suite or Vitech Genesys [16], [25].

The CORE® suite integrates systems engineering along the entire life cycle of the project, starting from the collection of requirements up to the tests, identifying the present interfaces and managing the relationships between the system data.

It provides a common place in which to group the various critical aspects of the engineering activity, i.e. requirements, architecture and tests, defining their behaviour, carrying out verifications and validations. The work through simulations, which can always be carried out, is subject to continuous reviews, more and more in detail during the life-cycle, capable to identify problems and errors at every level of the project.

The tool is based on the “model-centric” idea, therefore all data and files are saved in a common repository, where users can collaborate on multiple platforms in real-time and can add different types of diagrams contained in a special library. These include, for example, activity diagrams, N2 diagrams or sequence diagrams as well as traditional diagrams and SysML graphs.

CORE®, as the Vitech website specifies, was created specifically for systems engineers by systems engineers and is naturally geared towards their needs in model making. Furthermore, due to its simplicity, it is not necessary to refer to programming and modelling specialists [32].

Genesys is the latest tool produced by Vitech and is recommended for large organizations. This faces every aspect of systems engineering by exploiting connections with Microsoft Excel, Power Point and Project but also with industrial tools such as Matlab, Simulink, ModelCenter and DOORS to create a “connected engineering flow”.

Similarly to the CORE® suite, it allows performing V&V at every level of the project in order to limit and reduce risks and problems. It also lets to simulate the behaviour of the system and evaluate its performance. The tool “uses a full suite of nine SysML diagrams supported by a proven systems metamodel”, automatically generating and updating the diagrams [33].

2.2.4 NASA JPL State Analysis Methodology

State Analysis is an MBSE methodology developed by the California Institute of Technology Jet Propulsion Laboratory (JPL) [27].

SA uses a control architecture based on model and state, in which the state is defined as “a representation of the momentary condition of an evolving system (the value of the state variables at a precise moment)” and the model as “the state evolution” [27], with the aim of producing system and software requirements “in the form of explicit models” that can reduce the gap between the requests made by software engineers and systems engineers [25].

The state and models together ensure system operation by predicting the future state, checking the desired state, and evaluating performance [25], [27].

As M.D. Ingham explains in “A Model-Based Approach to Engineering Behavior of Complex Aerospace Systems”, the SA allows rigorously and uniformly to discover and study the different states of the system, their behaviour, the different interactions between them to clearly describe the mission objectives through specific scenarios, how the objectives will be satisfied and to outline the constraints and rules that must be considered during the design [34].

The SA architecture can be represented by the “control diamond” shown in Figure 5 for it the key characteristics are described below:

- The state is explicit and constantly evolving. The state to be described at any time is defined through a collection of state variables that vary according to continuous time state functions [34].
- The state estimation, a process based on making measurements to determine the real state, and the state control, a command useful for correcting the current state to achieve the desired state, are two separate acts of the methodology. This separation is useful as it simplifies software design, its implementation and allows for an “objective state assessment” [34].
- The only interfaces between the control system and the one under control are the hardware adapters and data collections. The former allow the various measurements to be made and the commands required for control to be provided, while the latter are viewed to determine the status of the system [34].
- Models are used for both “execution” and “higher-level planning” and are “ubiquitous” throughout the architecture. They can be documented in whatever form is most convenient for their application [34].
- The architecture of the methodology consists of a closed-loop that aims to achieve the desired behaviour of the system. The goals can be “the desired quality of knowledge of the state that the estimators must achieve”, “the operational constraints” and “the monitored conditions” [34].
- “The control diamond elements can be mapped directly into components in a modular software architecture” [34].

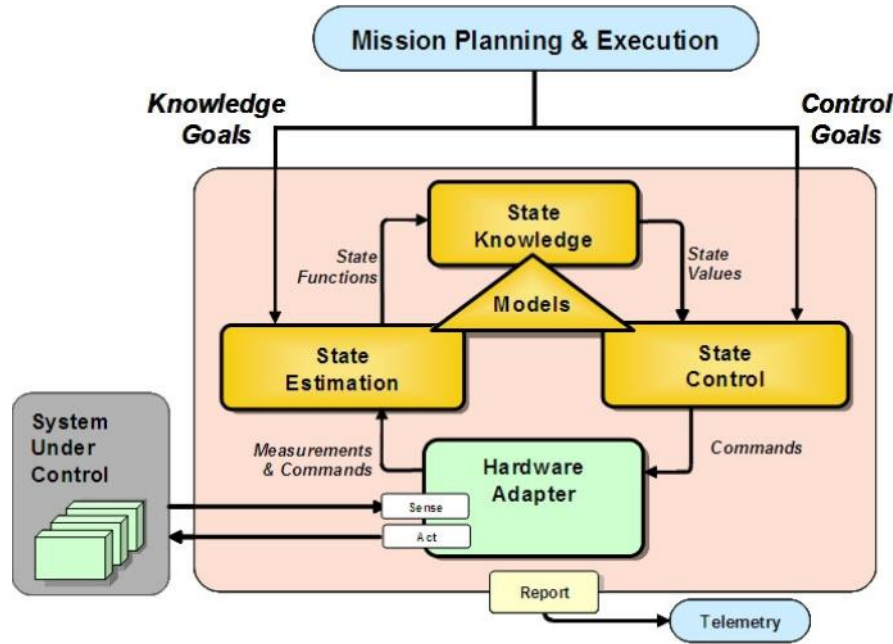


Figure 5: The state-based control architecture [34]

The State Analysis methodology is also based on three fundamental principles:

- The distinction between the system under control and the control system is clear. All the aspects related to the functioning of the system constitute the control that can be exercised using models of the system under control [16], [25].
- All control system models must be explicitly identified and must be accepted by all systems engineers [16], [25].
- The understanding of system states is critical to modelling success [16].

The SA methodology defines an iterative process to iteratively characterize the model throughout the entire life cycle. The steps of this process are the identification of the high-level objectives, the identification of the state variables, the definition of the state models, the definition of the measures to be carried out to determine the state variables, the definition of measurement models, the identification of the commands for the control of the state variables, the definition of the command templates [34].

The tools that use this methodology are based on relational database systems, among these, we find, for example, Oracle® with a front-end interface. SpecTRM-RL and SpecTRM-GSC are also used for specification and requirements management [25].

SpecTRM focuses on the initial phases of the project related to system requirements and specifications where safety decisions are made. SpecTRM uses new languages called “intent specification languages” which explain the behaviour of system and software engineering activities, clarifying “the what”, “the how” and the logic followed. One of these languages is SpecTRM-RL to specify requirements, based on simple formalisms to carry out an automated analysis. SpecTRM is also characterized by an intuitive editor that can be used by non-experts [35].

2.2.5 The Systems Modelling Toolbox (SYSMOD)

Weilkiens' System Modeling Toolbox (SYSMOD) is a methodology that uses the SysML modelling language for system architecture development and requirements traceability [16], [27]. Particularly, according to “Weilkiens T., Lamm J., Roth S., Walker M., Model-

Based System Architecture, 2016”, it allows to perform the requirements identification and definition, to model the system context from which it is possible to derive the requirements, to define the system use cases, to describe the use case flows and to model the system domain [36].

The system architecture is defined by creating processes, internal structure and parameters useful for studying its behaviour [27]. The approach followed by the methodology is a top-down type where methods, products and roles of the players involved are the main artefacts [16].

SYSMOD can be used with any modelling tool [16].

2.2.6 Object-Process Methodology

Object-Process Methodology (OPM) is a methodology developed at the Massachusetts Institute of Technology by Dr. Dov Dori that represents a holistic approach to model natural or artificial complex systems [25], [37].

Dori describes OPM as “a formal paradigm for systems development, lifecycle support and evolution” [25], [38].

The methodology is based on the concept according to which “the whole universe is an object or a process” and therefore it is possible to build any system through the relationships between three entities: objects, processes, and states [25].

The object is defined as what exists or can exist in a physical or computerized way, the process is an object “transformation model” between one state and another, the state is the object condition in a given instant of time [16], [37].

Objects and processes together are known as “things” and can be involved in the fundamental relations of the OPM “generalization, aggregation, instantiation and exposure”. OPM allows modelling a system from a “structural”, “functional” and “behavioural” point of view combining a graphic model called Object Process Diagram (OPD), able to represent the three entities described above and their relationships, and a language (a subset of the English language), consisting of bound sentences, known as Object-Process Language (OPL), capable of providing a textual duality to the created OPD [25], [37].

The models in OPM have a hierarchical structure, starting from the system diagram, a high-level diagram that represents the main functions of the system, up to increasingly complex and detailed OPDs. To pass from one level of detail to another, unfolding/folding mechanisms are used, necessary to “refine/abstract the structural hierarchy of a thing”, and zooming/out-zooming mechanisms to show or hide “internal details of one thing”. In this way, it is possible to define a system at any level of detail without ever losing sight of the “big picture” [25], [37].

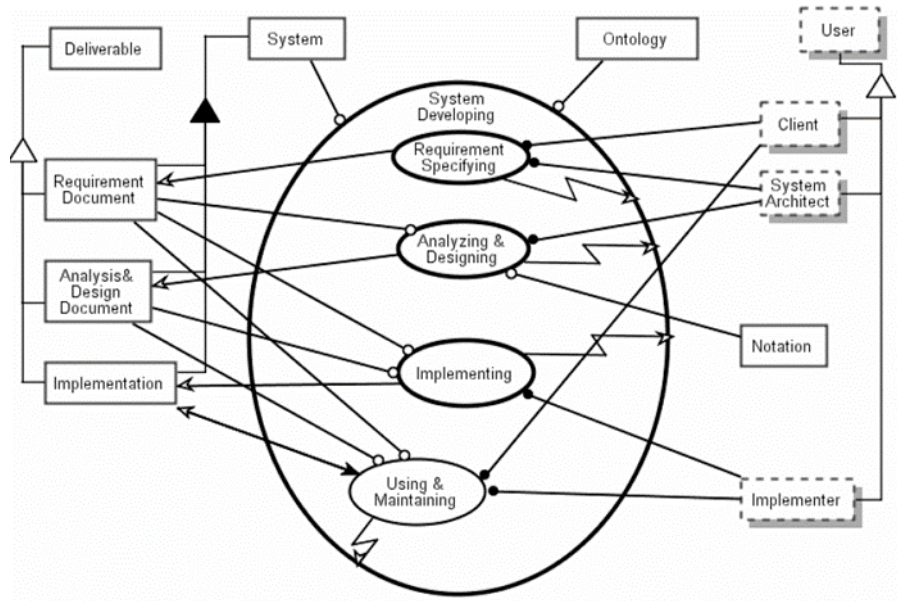


Figure 6: Zooming into System Developing [25]

The image above (Figure 6) allows observing the development phases of the system used by the OPM methodology:

- Requirements specification: managed by the system architect, through OPM, and by the customer, thanks to OPL, that work together creating a requirements document capable of identifying the problems to solve with the system. In order to reduce development and debug times, the methodology allows reusing the previously created requirements.
- Analysis and design: in which the “skeleton” of the system architecture is created, taking the requirements document as input, identifying the behavioural and structural aspects.
- Implementation: where a target language (i.e., Java, C++, SQL), the implementation profile and a directory for placing the artefacts are chosen. During this phase, the implementer can make changes to the behaviour and structure of the system.
- Use and maintenance: where the system is checked against the requirements document, eliminating any discrepancies [25].

The methodology does not have a software-oriented linguistic semantics as it was intended for general use [37].

Today OPM can be used through the OPCAT tool that allows the management of artefacts, “system requirements management, including interconnections and traceability to model entities, animated model simulation, code generation and automatic document generation” [25], [37].

OPCAT supports the entire life cycle of the system step by step by “implementing, on a single repository, the graphic and textual bimodal expression of the OPM model”. The suite creates configurable models for the system's current state, future goals, project constraints and hardware and software requirements. It is also possible, as mentioned above, to produce animated simulations in order to manage requirements or other advanced features [39]. The tool generate also automatically documents. To model a

complex system, the user must first create an OPD that corresponds to the System Diagram, to which OPCAT will associate an OPL, secondly, he must define the type of project, physical or IT, and the origin, systemic or environmental. So he can progressively add new parts to the project by detailing it [39].

2.2.7 INCOSE Object-Oriented Systems Engineering Method (OOSEM)

The Object-Oriented Systems Engineering Method (OOSEM) was developed through a collaboration of Lockheed Martin Corporation and Software Productivity Consortium in the second half of the 1990s and since then, thanks to further refinements carried out by the INCOSE OOSEM Working Group, it has been applied for the development of hardware systems, software, databases and manual procedures [25], [40], using for modelling originally Unified Modeling Language (UML) and later Systems Modeling Language (SysML) [41].

From S. Friedenthal, A. Moore, and R. Steiner, “Residential Security System Example Using the Object-Oriented Systems Engineering Method”, it is clear that the methodology, by generating system models as output, has two main objectives: “to facilitate the integration of systems engineering with object-oriented (OO) software engineering”, integrating top-down concepts, in order to ensure greater product flexibility, and “apply OO modelling in a way that benefits the systems engineering process”.

OOSEM allows the acquisition, analysis and specification of the system and its components guaranteeing a consistent visualization and ensuring the reuse of previously created models for future applications [25].

The development of the system takes place according to the following activities:

- Stakeholder needs analysis: to identify the needs of interested parties to develop the requirements “which are specified in terms of mission/company, effectiveness measures and cases of superior use”.
- System Requirements Analysis: to specify system requirements that support mission requirements. Functional, interface, data and performance requirements are identified for the system that is modelled as a black box capable of interacting with other systems and with different users. All the requirements are monitored through a database able to keep track of updates and to evaluate the impact of the changes.
- Definition of Logical Architecture: an activity that includes breaking down and partitioning the system into logical elements which, performing the assigned functions, “interact” to satisfy the requirements. How the architecture is divided is decided by following the guidelines provided by the methodology according to criteria such as “cohesion, coupling, design for change, reliability, performance and other considerations”.
- Synthesis of Candidate Physical Architectures: useful for expressing the relationships between the physical components of the system. These are represented as “system nodes” equipped with different features for the hardware, the software, data and procedures.
- Optimization and evaluation of alternatives: an activity that, through different evaluation parameters, allows to make comparisons between different architectural alternatives to choose the best, always considering the

requirements imposed, the effectiveness of the solutions and the impact of the risks.

- Requirements traceability management: always useful to have a total consideration of all mission and component requirements.
- Verification and validation of the system: a final task that is carried out to verify that all the needs and requests of the stakeholders are met. “The verification system can be modelled using the same activities and artefacts used for modelling the operating system”. The outcome of the verification may lead to the need to change the requirements. To carry out this activity it is necessary to develop plans, procedures and verification methods (such as inspection, demonstration, analysis, tests), integrate the components, verify them and finally analyze the results, producing verification reports [25], [40].

In Figure 7 a summary graph of the various activities is shown.

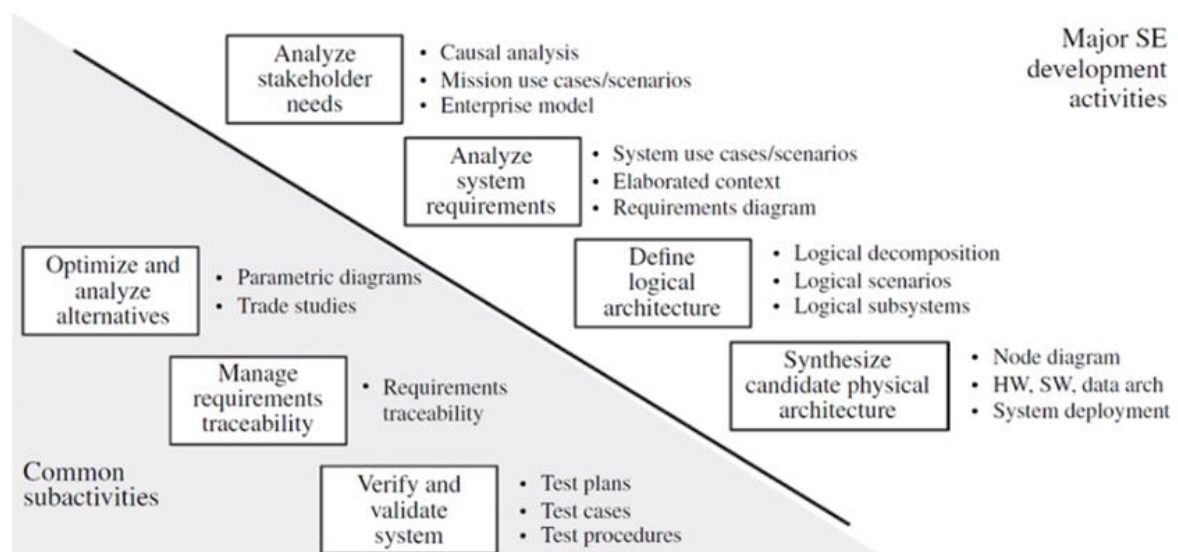


Figure 7: OOSEM Activities and Modeling Artifacts [25]

These activities are performed during the project for each system individually and are applied recursively and iteratively in the different hierarchical levels in a similar way to a Vee development process [40]. Furthermore, for their execution, risk management, configuration management, planning, measurement and the use of multidisciplinary teams must always be considered [25].

A specific tool for OOSEM methodology does not exist but tools based OMG SysML can be used. These allow working on COTS and associated requirements [25].

2.2.8 The ARCADIA method

Architecture Analysis & Design Integrated Approach (ARCADIA) is a methodology developed by Thales Airborne Systems in 2007 with the aim of “defining and validating” the architecture of complex multidomain systems, using engineering activities that are based on the formalization of needs and the functional analysis [16], [24].

ARCADIA is presented as a very flexible methodology that can be used according to top-down, bottom-up or middle-out approaches, capable of providing support for the collaborative work of the various users, interested in the design, and stakeholders [16].

In Thales it is used for the realization of complex systems in different fields, i.e. Defense, Space, Aeronautics, Land Transport, Security [24].

This methodology covers all project activities at different levels, allowing stakeholders to share the same information, integrates co-engineering by supporting the joint elaboration of models (“not only descriptive”), that can also interact with each other, and formalizes any type of specialist engineering concerning the requirements [24].

ARCADIA uses a series of diagrams with interconnected elements that create the structure of the system and its behaviour [16].

The ARCADIA work levels are:

- Operational analysis: the actors are identified, and the problems of the users and their requests are analysed. It represents the highest level of the methodology and is necessary so that the system can be adapted to expectations.
- System analysis: the functions, that the system must perform to meet the users' requests, are identified.
- Logical architecture: the logical components that make up the system, their sub-functions, their relationships, and their content are recognized, considering any imposed constraints.
- Physical architecture: the structure of the system is defined and how it will be executed. In this phase, behavioural components, that can perform the required functions, are identified.
- End Product Breakdown Structure (EPBS): the design limitations of the architecture are established and the constraints to be met are deduced [24].

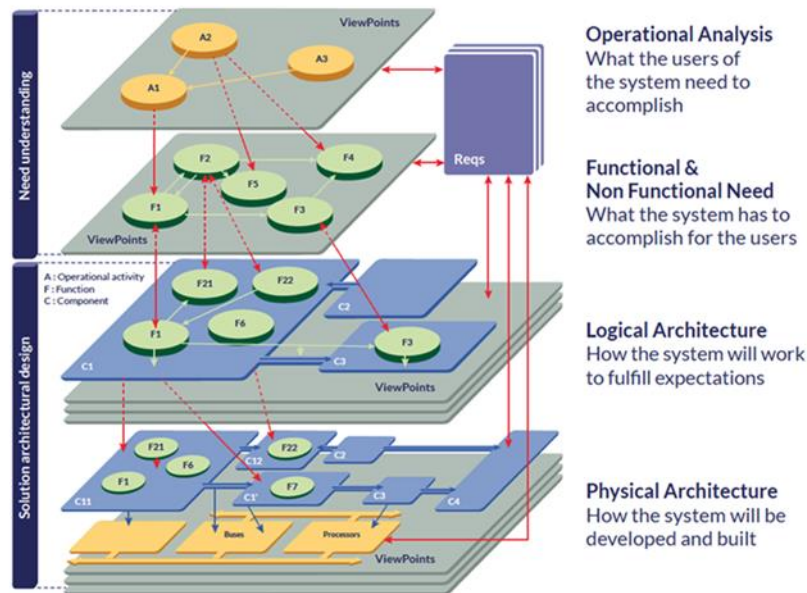


Figure 8: The main engineering levels of Arcadia [24]

A peculiarity of the methodology is that not all the project levels must be performed: the Operational Analysis, the Logical Architecture and the EPBS are considered optional. These may or may not be implemented based on design difficulties [24].

In conjunction with ARCADIA, Thales also developed the only modelling tool to work on, initially known as Melody Advance and after becoming open source as Capella. The latter is a modelling hybrid tool, strongly inspired by SysML, that allows tackling System Engineering and System Architecture problems. It lets to build seven types of main diagrams for each project level: “data flow diagrams, scenario diagrams, architecture diagrams, mode and state diagrams, distribution diagrams, class diagrams and capacity diagrams” [16], [24].

2.2.9 System Composer™

System Composer™ is an MBSE tool produced by MathWorks that executes “specification, analysis and design” of systems through one or more architectural models, consisting of “structural elements” characterized by “behavioural descriptions”, which can be presented in the form of diagrams to facilitate communication between interested parties. The diagrams can be “sequence diagrams, state charts, or Simulink models”.

The tool consents to follow an engineering process, through the use of Simulink, which involves the identification of the stakeholders’ needs, the definition of requirements and “use cases”, the iterative design of the system through hierarchical structures, the definition of the performed functions and their execution order, the definition of interfaces, verification and validation of requirements [42].

Below (Figure 9) followed workflow it is shown.

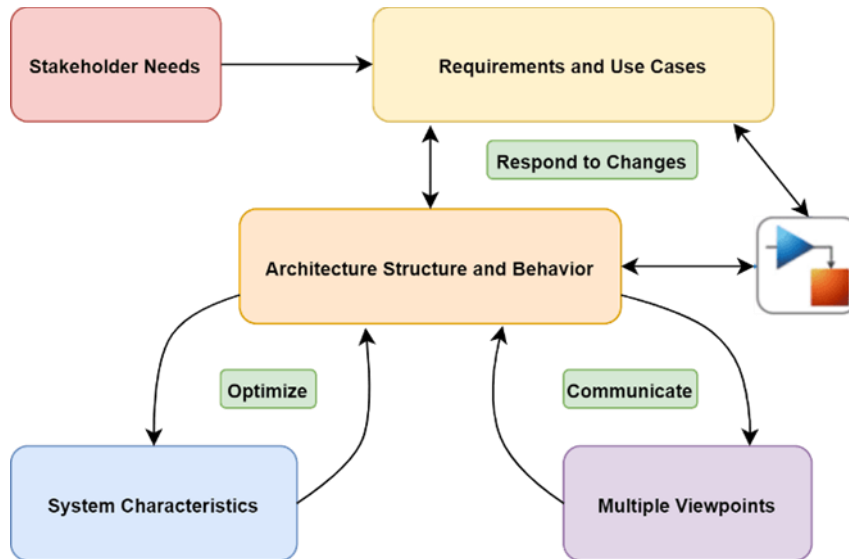


Figure 9: System Composer workflow [42]

As we read on the MathWorks web page, the models can be created directly or can be imported from other tools, and “can be used to analyze requirements, capture properties via stereotyping, perform trade studies, and produce specifications and interface control documents (ICDs)”[42].

2.2.10 SYS-ML

As reported in “E. Cambulut, V. Aydingül, B.Yaglioglu, Application of Model-Based Systems Engineering with SysML in a small satellite project, 2019” “a modelling language specifies the elements that can be used in a system model and the set of possible

relationships between those elements. In the case of a graphical modelling language, the language defines the notations and symbology that should be used to present the elements of the model in diagrams” [4].

SysML is a graphical modelling language developed in collaboration between Object Management Group (OMG) and INCOSE, which derives, and at the same time seeks to enrich by adding new features, from the UML, used for software development and production [4].

SysML carries out, through a set of diagrams, activities related to the analysis, design, specification, validation and verification of complex “physical”, “technical” and “sociological” systems (for example hardware, software, data, personnel, procedure) [2], [4], [16], [37].

Estefan affirms that “UML and SysML together make it possible to bridge a communication gap between systems engineering and software engineering” [25].

SysML can represent systems, components and other entities as follows:

- composition, interconnection and classification of the structure;
- behaviour based on functions, messages and status;
- constraints on physical and performance properties;
- allocations between behaviour, structure and constraints;
- requirements and their relationship, design elements and test cases [2].

Overall, SysML allows the creation of nine types of diagrams: four concern the “Structural Diagrams” structure, four concern the behaviour of the system and components “Behavioral Diagrams” and one the requirements. Parametric relations must also be considered [37].

The “Structural Diagrams” provide a static view of the system and characterize its logic and architecture, physical connections, content. The “Behavioral Diagrams”, on the other hand, describe the functionality and behaviour of the system under all conditions providing a dynamic view [4]. Requirements are entered in the text but can be observed in different formats (graphic, tabular, tree structure). Parametric relationships make available identifying links between system properties to analyze the performance and reliability of the system [37].

SysML inherited many ideas from UML that is an unintuitive language for engineers who have no knowledge in the field of software design. Furthermore, it does not provide a working guideline leading to ambiguity in modelling and therefore to a delay in learning how to use it [16].

2.2.11 Cameo System Modeler

Cameo System Modeler is an MBSE platform that provides tools to create SysML models and allows to perform design decision analysis, requirements and model consistency verification, to track changes made. Artefacts are managed in remote repositories.

The main peculiarities of Cameo are:

- Requirements management: it is possible to create SysML templates for requirements, keep track of verification methods and their satisfaction, create prioritizations.

- Function traceability for Gap Analysis: it keeps track of the work among the different project levels easily finding specific elements through dependency matrices, allocation matrices, interaction matrices, traceability matrices of functions and activities.
- Direct Web Publishing: where users can create customized reports guided by suggestions and can instantly check for errors.
- Resolution of parametric models and MoE (Measure of Effectiveness) of the system: this allows to identify and establish the project constraints but also to carry out risk, performance, and reliability analysis. The Simulation Toolkit is used for these analyses, and it is possible to interface with other programs such as Excel, Matlab, Mathematica and OpenModelica.
- Development and configuration management: that allows different users to work simultaneously on the same project through a single server connected to the internet.
- Security: for which different users can access only the project levels for which they are enabled according to authorizations.
- Interoperability: that gives the opportunity to interchange different UML, SysML and Unified Profile models and diagrams [43].

2.2.12 MagicDraw

MagicDraw is a modelling environment that supports the UML 2 and UML metamodel, able to integrate different applications, thanks to an open API, consenting the creation of IDEs, requirements, tests, estimates, MDDs, databases and more. It offers various features such as the creation and editing of diagrams in a very short time, the automatic completion of attributes and parameters, the verification and validation of models.

The environment enables following the entire life cycle of the project and provides a centralized, continuously verified, reference place in which to model the different processes. It automatically produces reports and documentation (in PDF, HTML, RTF format) on the requirements and the project.

MagicDraw offers the possibility of approaching the project at any stage as it is not bound to a specific point of the architecture. It also allows multiple users to work simultaneously on the same model using the Teamwork Cloud functionality and ensures fast navigation within the models, thanks to hyperlinks.

The platform is presented as an easy-to-access place where learning design methods is done easily through simple commands [44].

2.2.13 OCDT

Open Concurrent Design Tool is a software package produced by ESA to manage and design a system, during the first phases of the life cycle, through a simultaneous sharing of the work done and the data produced.

The platform was developed in conjunction with the ECSS-E-10-TM-25 Annex A standard which provides a semantic model to be followed as a guideline for design.

ConCORD, the software user interface, is an Excel add-on and allows simultaneous access by twenty users with a data update every two minutes. OCDT also offers the possibility of integrating other tools to carry out analyzes and simulations [45].

On the whole, ConCORD enables the creation of parameters, elements or engineering models as “Things” to describe the product and carry out analyses. For each system, the category, its parameters type and unit measures can be defined. It is possible to reuse for different applications the created data [46].

The software is distributed by ESA as open-source [45].

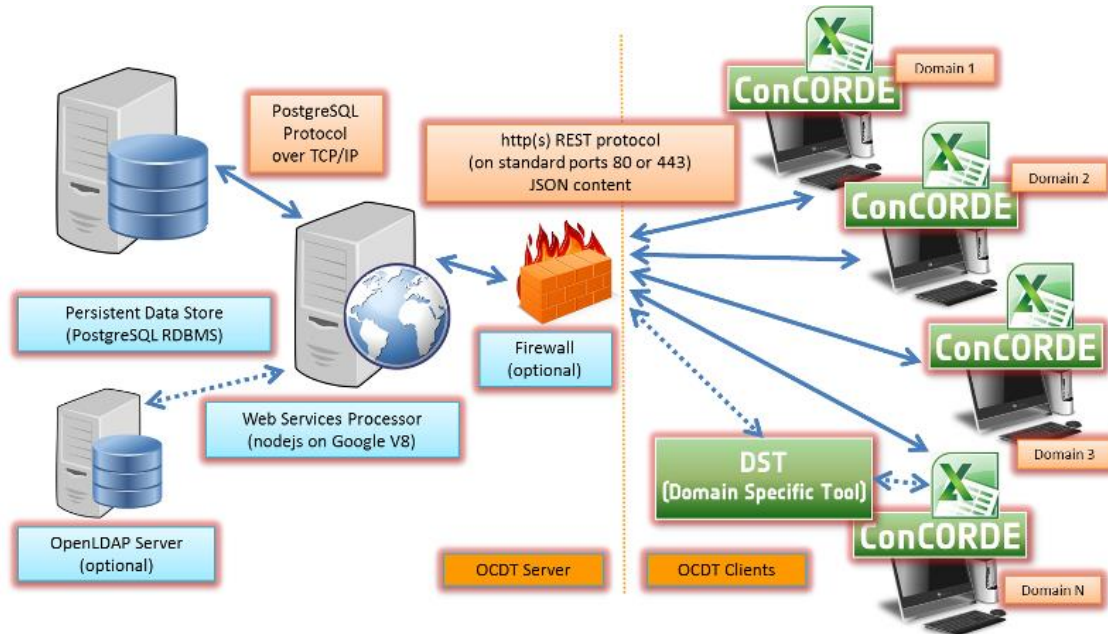


Figure 10: OCDT Architectural Overview [47]

2.2.14 Papyrus

The Papyrus platform offers a methodology based on a corporate architecture that documents any type of process, for which objectives to be met are set using models. These processes create relationships between information, people, applications and can be easily designed by business experts thanks to the fact that Papyrus guides them in the definition of processes, taking into account the business rules and the achievement of the established goals.

Papyrus is described on the company's website with reference to the following features:

- Process analysis by the Executive: the executive defines the processes indicating the owners and objectives. Diagrams are used only for process dependencies.
- Modelling by business users: through a web repository, users can model “entities” of data and content, which can be used to create “collaborative processes”.
- Business rules: the platform defines rules for modelling using a “natural language”.
- Company objectives: the objectives and their achievement are monitored during the process.
- Process management: processes are described utilizing “execution chains” and not rigid flowcharts.

- Involvement and collaboration of users: Papyrus allows defining users and interfaces in order to have little limited “input forms”.
- Social Networking: the platform offers chats, blogs, RIAs and wikis through which you can interact.
- Portal applications: the offer includes an “integrated” and “powerful” server that does not require Java programming.
- Monitoring and control: management has the opportunity to track performance indicators during work.
- Reporting and optimization: thanks to digital registers, the server collects processes information in chronological order to subsequently create reports.
- Execution: processes are run on multiple levels and multiple servers in case they are of large volume.
- Content services: Papyrus provides the delivery of content of any type or format (e.g. print, e-mail, mobile, web, fax).
- Incoming content: the platform manages to acquire and process all incoming content.
- Outgoing content: documents and resources can be created for marketing or business purposes and can be provided through different channels.
- Archiving: the files generated are placed and protected within secure servers.
- Project management and implementation: all projects are implemented within the central repository called Papyrus WebRepository.
- Infrastructure: which consists of eleven different operating systems installed in a cloud.
- Integration: the platform allows easy integration of existing applications, devices and databases.
- Privacy: the privacy of the products created is guaranteed [48].

2.2.15 Valispace

Valispace is an intelligent browser platform that allows engineers to collaborate in real-time during the realization of a project by making calculations, keeping track of changes throughout the project life cycle, storing, and analyzing the collected data [49], [50].

Valispace is now used for the construction of complex hardware systems such as satellites or aircraft since it easily integrates with engineering tools already in use “providing an API that makes available automation, scripting and optimization” [49], [50].

Among the interesting features proposed by Valispace are the change of running calculations in a single “container”, instead of using multiple Excel sheets, the opportunity to keep track of the chronology of the performed work, allowing a comparison with values or graphs of the past, the ability to receive alerts in case of changes [50].

The platform is based on the following modules:

- Components: a section where “properties” that consist of elements constituting the system can be created. These can be ordered hierarchically through tree relationships and for them, the user can define different characteristics. For example, masses or volumes are implemented in this area, characterized by values, units of measure, margins, requirements of maximum and minimum. The properties can be used to obtain new ones through equations.

- Requirements: area in which the requirements are collected and managed in a “simple and efficient way”. Requirements can be divided into folders and can be linked to components or components values in order to be updated with every change and always have the correct version of the requirements. A hierarchy can also be defined for requirements through parent-child relationships. The verification method can be defined and the status can be tracked.
- Analysis: a repository where reports, technical budgets and graphs can be stored.
- Time Sequence: allow to make system time predictions based on changes over time of “variables” or “operating modes”. The latter let to create time sequences and monitor indicators for the system, while the former, on the other hand, represent the behaviour of the components in the different phases of the mission.
- Simulations: it is a work region that allows, through the definition of inputs and outputs, to carry out calculations of greater complexity that cannot be done within the component area. The GNU Octave framework is used, which has a syntax similar to Matlab.
- Tests: are procedures for verifying the requirements. Each procedure can be composed of different phases and sub-phases according to the needs that can be characterized by attached files and tags [50].

In general, as ESA states, “it aims to simplify the engineering process of hardware projects by allowing even small teams to design highly complex systems quickly and economically” [49].

2.2.16 Choice of project implementation

The thesis proposes an implementation procedure consisting of a combination of tools: Capella with the Arcadia methodology and Valispace. This choice was strongly guided by the possibility of being able to use methodologies and tools with a license already available to the project team or open source and the goal of moving from a document-centric approach to a model-centric one.

Capella has been used to define the SROC system with the operational analysis by identifying the stakeholders and their needs, the system analysis, the logical and physical architecture. Valispace has been taken into consideration to keep track of the requirements, to create a complete and extended mission architecture, to carry out mission budgets through specific add-ons, such as those for Matlab and STK, to keep track of the documents.

The field of choice was initially restricted precisely due to the economic availability limit, the possibility of being able to implement large systems considering multiple levels of design and the availability of documentation to learn how to use the software. The area from which the tool was selected included Capella with the Arcadia methodology, Papyrus with the use of the SysML language and System Composer.

The latter has not been evaluated because, unlike the other two, despite being able to keep track of the requirements, starting from the requests of the users, it does not allow to guarantee connected and unified modelling, for the stakeholders and their needs, with the rest of the system.

The choice thus fell on Capella since, compared to Papyrus, it offers a “modelling approach” that “guides” the user in designing the system at various levels of detail, through the “Activity Explorer” function, avoiding ambiguity. Such ambiguities can occur using, for example, tools that make direct application of SysML, where the user is free to choose the modelling strategy, freedom that could lead him to misunderstandings if he approaches the tool for the first time. The concept of “usability” was therefore evaluated, which is linked to the reduction of time and costs to learn how to operate the software and the possibility of being able to use the tool. Capella, in relation to this, presents an effective and efficient user interface and is equipped with numerous intuitive functions (e.g. automated transitions of elements from one project level to another or operational and functional chains that will be discussed later).

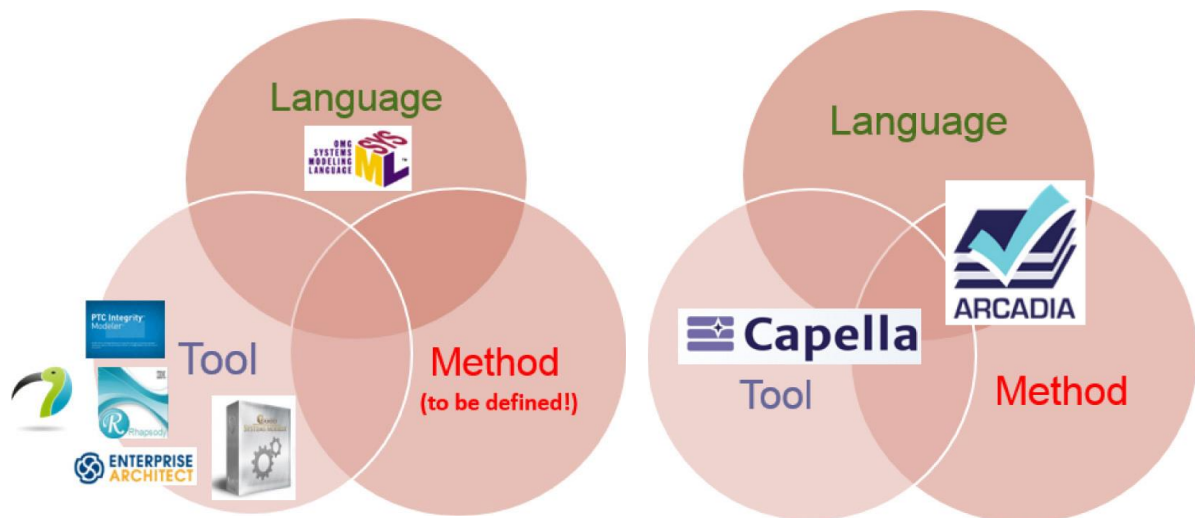


Figure 11: Difference between classical MBSE with SysML (left) and MBSE with Arcadia/Capella (right) [24]

Capella allows to clearly describe the design choices and to create constructs characterized by a close relationship between the architectural and behavioural aspects of the system, unlike SysML, as found by various bibliographic researchers, in which to describe the functions semantically ineffective blocks must be used.

Requirements management can be carried out on Capella after downloading specific add-ons but it is not possible to create traceability matrices. This led to the use of the Valispace tool, which the project team already owned a license for. Valispace has a user-friendly interface and allows to keep track of the requirements, carry out their verification, to be able to associate them with the various components inserted, create flags capable of indicating errors in non-compliance with them and indicate their status. The platform, as written in the previous paragraph, has great potential, allowing for example the creation of a mission architecture by implementing all the elements, that are part of it, with their characteristics, the traceability and collection of documents, the possibility of budgeting internally or through the function of “import” and “export” through links with other tools.

It is therefore expected that the combination of the two tools will cover every aspect of project management in its early stages.

3 SROC

We now move on to describe the work carried out for SROC in phase delta A and B, by analysing first what was carried out in phase O/A of the project.

3.1 Operational analysis

The goal, that the systems engineer sets during the design, is to satisfy the stakeholders and their needs. Precisely for this reason in the planning phase of the project (Phase O), it is necessary to carry out an analysis of the interested parties and needs to correctly set up the development of mission objectives, requirements, and constraints.

In the following paragraphs, the process performed at the beginning of the SROC Phase O study is described in detail.

3.1.1 Stakeholders and needs phase O/A

The identification of stakeholders was carried out through brainstorming sessions trying to identify the figures able to play key roles in the project, for example, those who will pay for the execution of the mission, who will use the system, who will judge the suitability for the use of the system, who will take care of regulating the system to be implemented, who is involved in the realization and construction of the system, who will study the data produced, which entities can be negatively affected by the mission and so on.

Every stakeholder was identified as either a consumer, customer, or supplier.

After these sessions, as shown in Figure 12 the identified stakeholders were subdivided into four categories and prioritized relying on their level of interest and power, and for each of them, it was investigated what it would expect by the mission in terms of needs to derive mission objectives, drivers, and constraints.



Figure 12: Stakeholders' mapping [7]

The Governments, the European Space Agency, and the Space Industry understood as “suppliers” are classified as the most powerful and most interested in the SROC mission, so their needs must be managed closely. CubeSat developers, the research community, and the Space industry understood as “users”, should be informed about the SROC programme and mission outcomes, and their needs must be taken into account already

in the early phases to reduce design iterations in later stages. Regulators must be kept satisfied, developing the mission according to ECSS and international regulations. The general public and education community can be considered minimally.

Table 2 summarises the SROC stakeholders and their needs.

Stakeholder(s)	Needs
European Space Agency (Customer)	To execute a CubeSat mission with limited risk for the Space Rider system
	To foster technology development for CubeSats and small satellites
	To enable innovative mission objectives with CubeSats and small satellites (e.g. inspection of orbiting objects)
	To enable reusable on-demand missions with CubeSats (SROC as demonstrator of a flexible reusable system)
	To involve general public and universities in ESA activities
Space industry - SROC Developers (Supplier)	To develop advanced technology for market competitiveness
	To keep production costs as low as possible
	To have a robust and effective development roadmap for innovative CubeSat applications
Space industry - SR Developers (Supplier/Consumer)	To have an independent source of information about SR effectiveness (e.g. health status during operations and before re-entry) (Consumer)
	To demonstrate SR as flexible payload delivery system in orbit (Supplier)
	To develop and assess safety constraints/requirements applicable to SROC mission (Supplier)
Regulators (Supplier/Consumer)	To fulfil European and international standards for space mission design (Supplier)
	To develop new standards for future missions (Consumer)
Governments involved in SR programme (Customer)	To limit risk for the SR programme due to SROC mission
	To enhance the potential of the SR programme
	To keep mission cost as low as possible
	To provide economic and industrial return to the country
	To provide visibility of national space activities with unprecedented mission
CubeSat developers (Consumer)	To enable new technologies for CubeSats applications
	To validate CubeSat concepts for advanced mission objectives
	To foster CubeSat applications for future space missions
Research community (Consumer)	To acquire new knowledge in all the areas involved in the SROC mission
Education community (Consumer)	To use SROC mission as case study for educational and outreach purposes
General Public	To be involved in space missions related activities
	To be informed about ESA space missions' activities

Table 2: Stakeholders' needs [7]

In the next paragraphs, it will be explained initially the followed MBSE method for the implementation of SROC stakeholders and needs, and then through “reverse engineering” it will be given some tips or pieces of advice on a methodology that systems engineers could follow during the early phase of a project.

3.1.2 Operational Analysis: Capella architectural elements and diagrams

This first phase of the project was implemented using Capella's "Operational Analysis" section following the Arcadia methodology, working at a higher level of abstraction, considering the project extent, rather than the examples shown on the website of the tool and the manual written by Pasqual Roques from which the work was inspired. As previously explained in the paragraph dedicated to the general description of the methodology, the "Operational Analysis" was conceived to study "what the users of the system need to accomplish" by identifying at first the actors who interact with the system and subsequently their requests. In general, the main architectural elements (Figure 13) that can be used in this phase are:

- Operational Capabilities: the ability of an organisation to provide a service that supports the achievement of high-level operational goals.
- Operational Activity: process step or function performed toward achieving some objective by "Entities" that could necessitate using the future system.
- Operational Entity: a real-world entity (other systems, device, group or organisation), interacting with the system (or software, equipment, hardware) under study or with its users.
- Operational Actor: a non-decomposable "Operational Entity".
- Operational Interaction: set of operational services invocations or flows exchanged between "Operational Activities".
- Operational Process: a logical organization of "Interactions" and "Activities" to fulfil an "Operational Capability".
- Operational Scenario: it describes the behaviour of a given "Operational Capability" [51].

From a graphical point of view, instead, the following diagrams have been utilized:

- Operational Entity Breakdown (OEB): a breakdown diagram of the "Entities".
- Operational Capabilities Blank (OCB): it allows the creation of "Operational Capabilities", "Operational Entities" and "Actors", and the numerous relations between them.
- Operational Activity Interaction Blank (OAIB): a diagram that shows a set of "Activities" linked together by "Interactions".
- Entity Scenario (ES): it shows the vertical sequence of the messages passed between "Capabilities".
- Operational Architecture Blank (OAB): graph useful to allocate the "Activities" [24], [51].

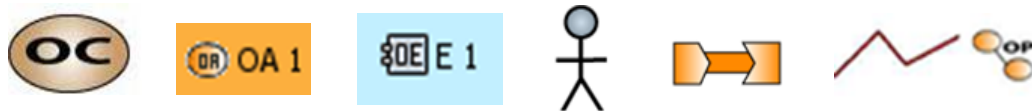


Figure 13: From the left symbols of "Operational Capabilities", "Operational Activity", "Operational Entity", "Operational Actor", "Operational Interaction", "Operational Process" [51]

3.1.3 Operational analysis phases delta A/B1

The chosen category, as previously discussed, to realize the stakeholders and needs analysis was the “Operational Analysis” of Capella.

Following the previously phase work, the initial step was the representation of identified stakeholders, so the first graph created was the Operational Entity Breakdown (OEB) where the different stakeholders were created as “Operational Entities”.

As shown in Figure 14 since two different kinds of stakeholders were identified for the Space Industry, i.e., SROC Developers (Supplier) and SR developers (Supplier/Consumer) it was mandatory to create two operational entities contained in the Space Industry one.

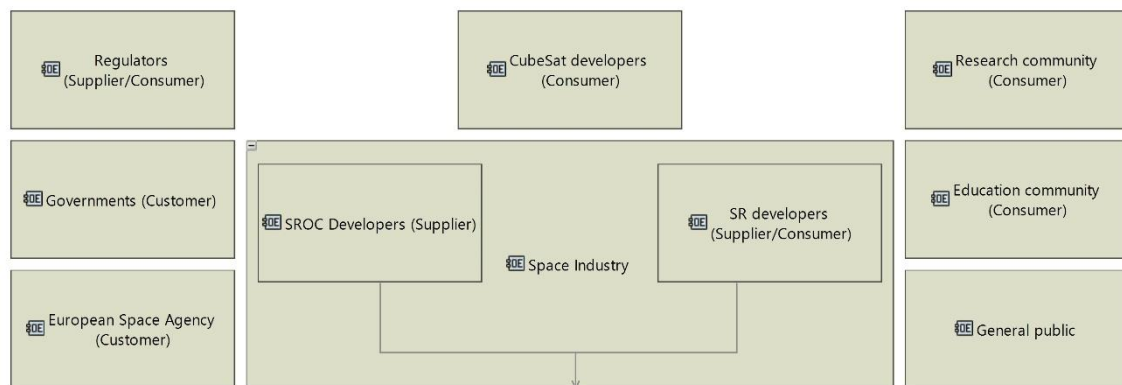


Figure 14: OEB

Later an Operational Capabilities Blank (OCB) diagram was realized, by inserting the operational entities previously added and creating operational capabilities. It was created to highlight high-level needs (“the capabilities”) common to the different stakeholders and their relationships with them.

Five capabilities were recognized:

- Development of key technology: in order to encompass needs related to the improvement or the production of new technology for space application;
- Expand mission portfolio: to englobe the needs linked with the desire to innovate missions in the field of space exploration;
- Safety up: to include the necessities connected to safety;
- Cost down: to include the necessities connected to cost;
- Dissemination: to include the needs of management, sharing, and distribution of data, documents, and deliverables among the various stakeholders.

The diagram was then completed linking this high-level objective to the involved stakeholders (Figure 15) and creating for each of them an Operational Activity Interaction Blank (OAIB).

Colours were used to identify entities with different interests and powers, in order to see the classification in the model (i.e., promoters, defenders, latentes, apathetics).

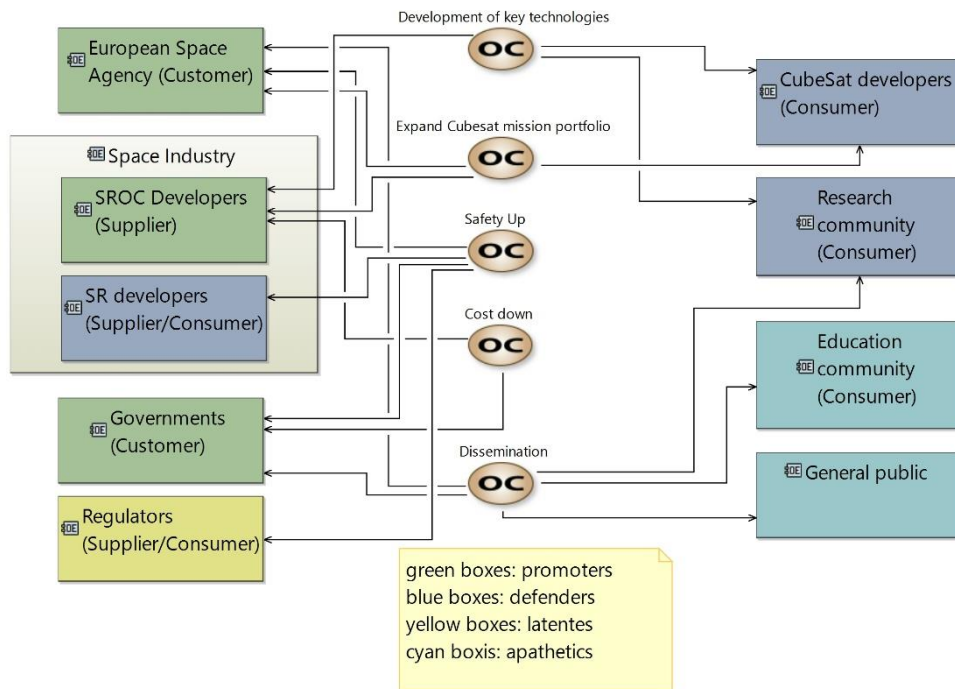


Figure 15: OCB

OAIBs were created to locate the identified needs as Operational Activity and to identify the related interactions.

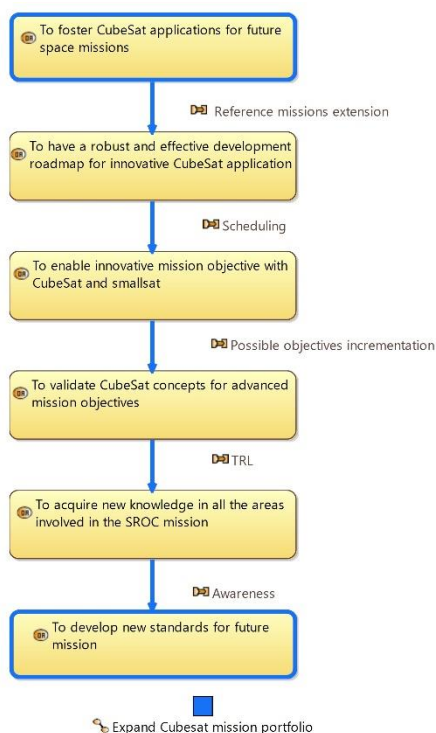


Figure 16: Expand CubeSat mission portfolio OAIB

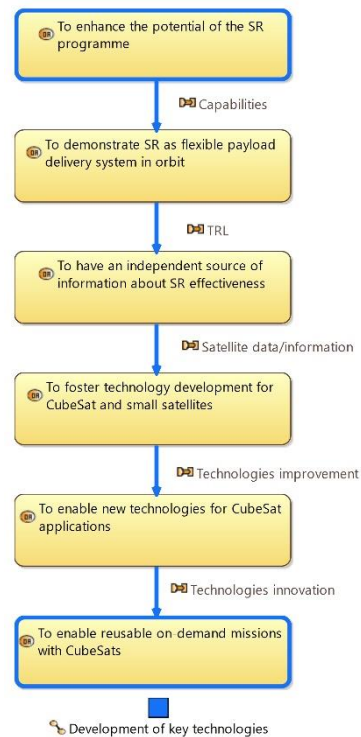


Figure 17: Development of key technology OAIB

In the images above (Figure 16, Figure 17) it is possible to observe examples of two OAIBs created, “Expand CubeSat mission portfolio” and “Developed of key technology”, where it has been tried to assign a certain logical succession at the different needs.

The stakeholders' analysis for the SROC project was then completed creating two other kinds of graphs: Entity Scenario (ES) and Operational Architecture Blank (OAB).

The Entity Scenario diagrams have permitted to organize chronologically the operational activities, belonging to different operational entities, for the defined capabilities. In this case, it has been chosen to create one ES for the high-level need “Cost down”, one for “Safety up”, and to split up the “Dissemination” capability in order to consider separately the distribution of data for ESA and Governments.

The next images, Figure 18 and Figure 19, show the ESs for the “Dissemination” capability.

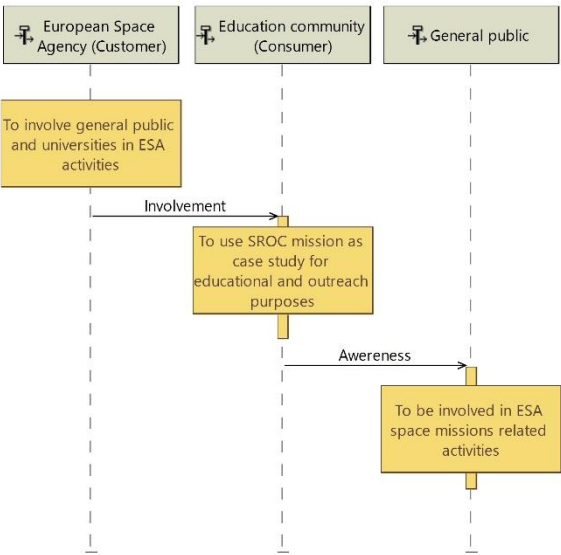


Figure 18: ESA Dissemination ES

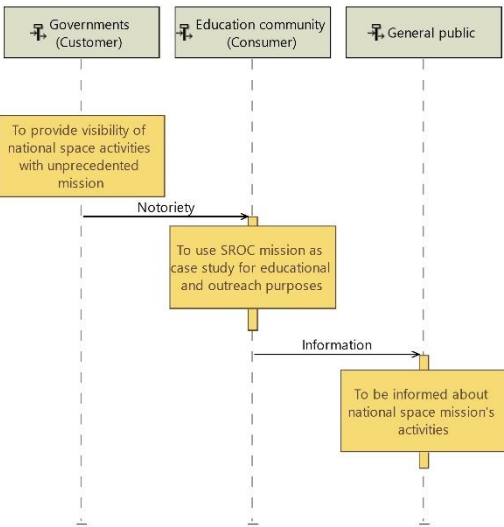


Figure 19: Governments Dissemination ES

The reasons that led to this separation are to be seen in the exclusive communication between the scientific community and the ESA, since the latter will not communicate directly with national governments, and also in the different necessities of the general public that in the first case would be involved in ESA space missions' activities, otherwise, in the second case, it would be informed about national space missions activities.

The OAB, Figure 20, was created to show in a single diagram all the identified stakeholders and their related needs. Therefore, it allows to see globally the logical interactions among the different needs and also to observe through “chains” (“Operational Processes”) how the needs of the different stakeholders are part of equal capabilities.

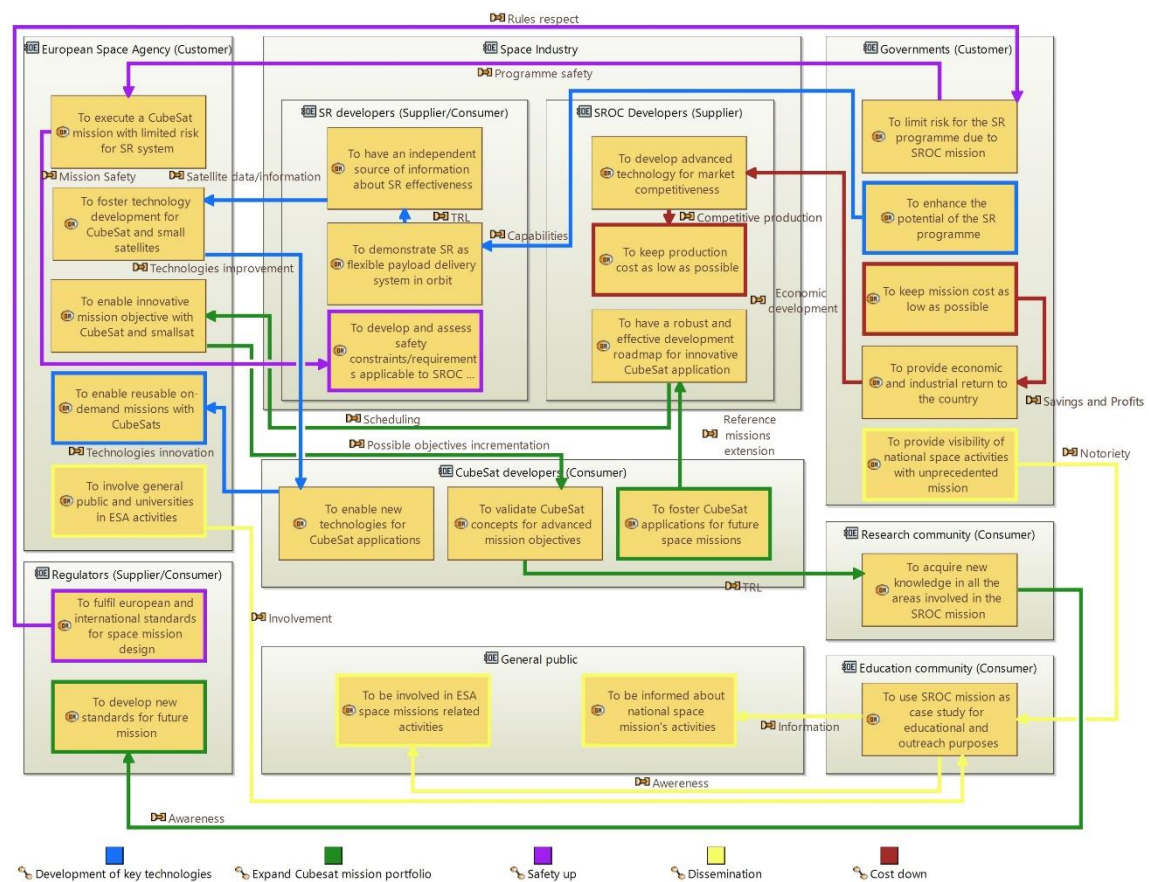


Figure 20: OAB

3.2 System Analysis

The stakeholders' needs identification with a subsequent definition of the mission objectives, the system, and the actors, led to determine, through a high-level functional analysis, which tasks they must perform, regardless of the physical components that will perform them, leading to the production of “concept of operations” and a high-level mission architecture where the functional interactions between the elements play a leading role. Thanks to the identification of the main characters, it is also possible to define the necessary interfaces.

3.2.1 Concept of Operations, Mission Architectures and Functional Analysis phase o/A

During the o/A phase, after defining the mission statement, the system, i.e., SROC, and the main actors involved in the mission, i.e., SR, the ground stations for communication with SROC and SR, the deployment, docking and retrieval mechanisms, were identified.

From the mission statement: “To deploy a CubeSat in LEO to support operations of Space Rider through multispectral and visual observations taken in the proximity of the vehicle during the orbital phase. To enhance CubeSats' capabilities in the proximity operations domain”, the primary objectives of the mission have been derived:

- To observe Space Rider with unprecedented imaging.
- To demonstrate critical technologies and functions related to formation flight missions.

And the optional secondary one:

- To demonstrate CubeSats in-orbit retrieval and reuse capabilities [7].

The critical areas to face during the missions were subsequently identified. These were:

- imaging capabilities;
- guidance and control capabilities;
- proximity navigation;
- docking;
- communication architecture;

In these early stages, three mission concepts were developed:

- “Observe & Retrieve mission (SROC baseline mission): the SROC CubeSat is deployed by the Space Rider and it is retrieved into its cargo bay before re-entering Earth, pursuing both primary (observation of the vehicle and demonstration of technology for formation flight) and the secondary (testing retrieval and reuse capabilities) mission objectives”.
- “Observe mission (SROC reduced mission): the SROC CubeSat is deployed by the Space Rider to pursue (only) the two primary mission objectives (observation of the vehicle and demonstration of technology for formation flight) without retrieval of the CubeSat into the Space Rider cargo bay” in which the most critical phase of the mission is therefore excluded. This concept of mission can be considered as an off-nominal condition, as it can be considered as a mission in which due to an unforeseen event it is not possible to carry out the recovery of the satellite.

- “Observe & Reuse mission (SROC enhanced mission): the Observe & Retrieve concept could be pushed further considering multiple deployment and retrieval, thus demonstrating the actual reuse of SROC within a single mission of Space Rider. The CubeSat is retrieved in the SR cargo bay, prepared for another observation mission, deployed and retrieved again a certain number of times before final retrieval/stowage/return to Earth” [7].

These different mission concepts were considered as incremental steps towards achieving the full capabilities of an RV&D mission between a CubeSat and a mothercraft.

Table 3 shows an example of the first concept of operations with different mission phases, durations, and scenarios.

Mission phase	Duration	Mission Scenario
Deployment Phase (DEP)	TBD hours	SROC system preparation SROC spacecraft separation
Early Operations Phase (EOP)	5 days (best case) 10 days (worst case)	Link acquisition Detumbling Attitude acquisition Appendices deployment Checkout post-deployment Calibration of thrusters Calibration of cameras Test of critical equipment
Holding phase 1 (HOP1)	4.5 hours	SROC is in hold point 1 (HP#1)
Rendezvous Phase (RVP)	4.5 days (best case) 9 days (worst case)	SROC performs a series of manoeuvres to follow a safe path from HP#1 towards SR to get in the operative orbit Rendezvous from HP#1 to SR -> Out-Of-Plane trajectory from HP#1
SR Observation Phase (SROP)	8 days	Insertion into the Walking Safety Ellipse (WSE) Observation in the Walking Safety Ellipse (WSE) Free Flight Approach to SR (except for the last Inspection cycle)
Holding Phase 2 (HOP2)	4.5 hours	SROC is in hold point 2 (HP#2)
Docking & Mating Phase (DMP)	10-15 hours	Fly-around trajectory from HP#2 to R-bar Close approach Mating
Retrieval Phase (REP)	5 hours	Capture Post docking check out SROC Retrieval
End of Life (EOL)	TBD	SROC is stored in the SR cargo bay and re-enters with SR

Table 3: SROC Baseline Mission – ConOps 1 - Phases and Scenarios description [7]

The mission architecture was outlined considering the previous steps by identifying:

- subjects: SR and the retrieval mechanisms;
- launch segment: Vega C;
- space segment: the CubeSat SROC with its retrieval mechanism;
- payload: a multispectral camera;

- orbit for formation flight: an SSO (Sun Synchronous Orbit) midday-midnight;
- communication architecture: store & forward;
- ground segment: the Ground station network and MCC;
- operations: with professional operators of the CubeSat Control Center in Turin.

The next table details the mission architecture proposed during phase A.

SROC proposed mission architecture – Phase A		
Mission element	Description	Trade-offs/Add-on/Comment
Subject	Space Rider Observations CubeSat Retrieval capabilities	Multi-spectral Observation of SR (or only visual or VIS+NIR observation, depending on SR needs) Single deployment and retrieval of SROC
Space Segment	1 CubeSat 1 Retrieval Mechanism	12U form factor baseline Architecture is compliant with the application for the enhanced scenario (Observe & Reuse mission) Deployment and Retrieval Mechanism
Payload	Multispectral imager	Multispectral imager, with increased FOV and adjusted wavelengths in the following bands: <ul style="list-style-type: none"> • Visual • Near InfraRed • Thermal InfraRed
Orbit & Constellation	Formation flying with respect to SR	SSO midday-midnight assumed as the baseline Rendezvous trajectory: in-plane + out-of-plane segments Walking Safety Ellipse with relative inclination change for observation
Communication Architecture	Store & Forward architecture	Direct link to Earth
Ground Segment	Ground station network MCC	Network of 6+ UHF ground stations, 1 S-band main ground station (+ 1 S-band ground station back-up) Compatibility with Estrack network is guaranteed MCC in Torino. Link with SR MCC
Operations	Professional operators	CubeSat Control Centre (C3) @Turin
Launch Segment	Vega C	Launch assumed Mid 2023

Table 4: SROC mission architecture [7]

In this phase, a high-level functional analysis was carried out through the creation of a functional tree (Figure 21: Functional tree phase O/A [7]Figure 21), realized to satisfy the stakeholders' requests and identify the critical functions that must be enabled by the different elements of the proposed mission architecture. In particular:

- To integrate SROC into the Space Rider cargo bay [Allocation to deployment mechanism].
- To deploy SROC without risk for the Space Rider [Allocation to deployment mechanism].
- To maintain formation with the Space Rider [Allocation to GNC subsystem]
- To mate with the Space Rider [Allocation to retrieval mechanism and GNC subsystem].
- To demonstrate CubeSat reuse capabilities [Whole space segment is interested].
- To take multispectral observations [Allocation to observation payload] [7].

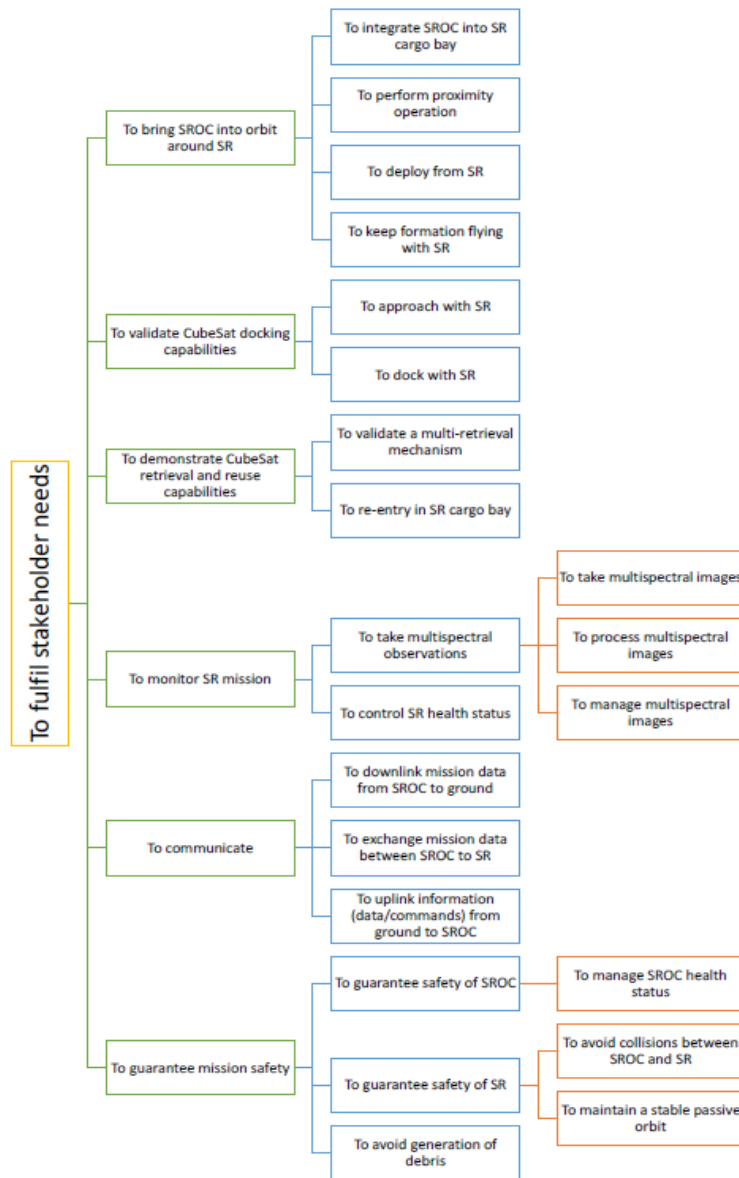


Figure 21: Functional tree phase O/A [7]

3.2.2 System Analysis: Capella architectural elements and diagrams

The functional analysis has been conducted using the Capella “System Analysis” section. According to the manual “Systems Architecture Modeling with the Arcadia Method”, this allows analysing “what the system has to accomplish for the users” by identifying “system functions needed by its users” [24].

From the examples provided in the book, it can be seen that between “Operational Analysis” and “System Analysis” there is a translation of some blocks, from one level to another, through the use of the “Transition” function. In our case, instead, we have chosen to keep the two levels separated, mainly using the first to define the high-level needs of the stakeholders and the second to identify the high-level functions that the system and the various involved actors must perform to satisfy the users’ requests [24].

The main elements of the architecture for functional analysis are:

- System: an organized set of elements functioning as a unit or an aggregation of end products, enabling products to achieve a given purpose.
- System Actor: external actor interacting with the system via its interfaces.
- System Mission: a high-level functionality of the system (system goal).
- System Capability: the ability of a system to provide a service that supports the achievement of high-level goals.
- System Function: an action, an operation or a service fulfilled by the system or by an actor interacting with the system.
- Exchange and Port: an interaction between some entities such as actors, the system, functions or components, which is likely to influence their behaviour. The “connection point” of an “Exchange” on an entity is called a “Port”.
- Functional Exchange: a piece of interaction between functions that is composed of data, events, signals, etc.
- Component Exchange: that identifies the interactions between system components and/or system actors.
- Physical Link: that allows depicting physical relationships between system components and/or system actors.
- Functional Chain: element of the model that enables a specific path to be designated among all possible paths [51].

The charts used to outline the procedure and the purpose of their use are:

- Contextual System Actor (CSA): to create and identify the “Actors”.
- Mission Blank (MB): it allows creating system “Missions”, the involved “Actors” and the links between them.
- Mission Capabilities Blank (MCB): to define “Functional Capabilities”, “Functional Mission” and “Actors”, and the numerous relations between them.
- System Actor Blank (SAB): his main goal is to show the allocation of “Functions” to “Components”. In “System Analysis”, these diagrams contain a box that represents the “System” under study and the “Actors” surrounding it.
- System Data Flow Blank (SDFB): it represents the information dependency network between “Functions”. These diagrams provide a diverse set of mechanisms for managing complexity: simplified links calculated between the “high-level Functions”, the “categorization of Exchanges”, etc.

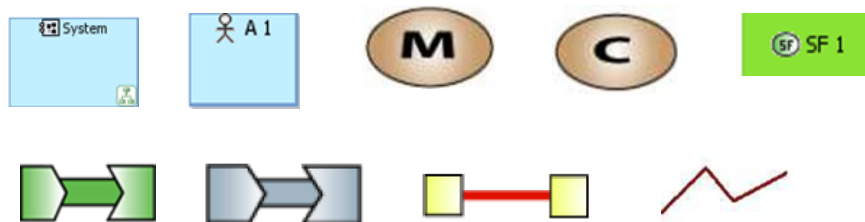


Figure 22: From the top left symbols of “System”, “System Actor”, “System Mission”, “System Capability”, “System Function”, “Functional Exchange”, “Component Exchange”, “Physical Link”, “Functional Chain” [51]

3.2.3 System analysis phases delta A/B1

At the beginning of phase delta A, revisions were made that gave rise to the necessity to modify and enrich the previously systems analysis. In this regard, the use of Capella has allowed, retracing the steps taken in phase A, enlarging the vision and the links between the various analyses that can be performed.

The first step was the definition of the main “System” and the “Actors” involved during the mission and their relationships, necessary for the mission architecture. A Contextual System Actor (CSA) graph was created in Capella (Figure 23). The identified system was SROC, instead, the actors, with whom it interacts, were SR, as the mean that carries and deploys the small satellite in orbit, the Space Rider MCC (Mission Control Center) and the SROC MCC, for operations, the MPCD dispenser, in which the CubeSat will be stowed during the launch and for the retrieve, and the docking device DOCKS, SROC subsystem that must be considered as a separate actor given its relevance for the mission objectives and its numerous interfaces.

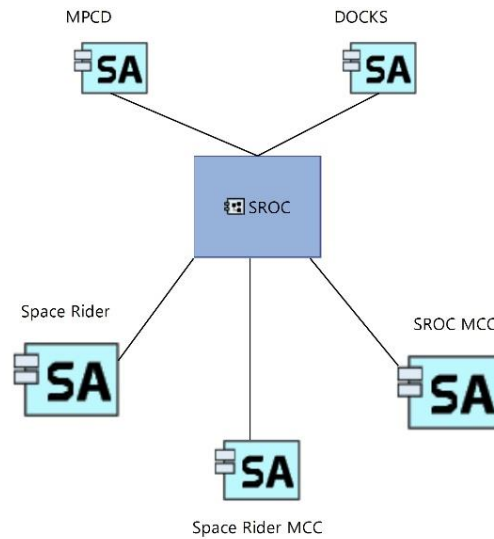


Figure 23: CSA

This graph started the creation of the system analysis, as after it, through the use of a Mission Blank (MB) diagram, it was possible to implement the three “Missions” (in orbit goals) carried out by SROC, set out in the previous paragraph, and define the interactions with the involved actors to achieve them (Figure 24). One can observe how the mission “To demonstrate CubeSats in-orbit retrieval and reuse capabilities” has become a primary objective in the delta A phase. Using the “note creation” feature differentiation between the primary and secondary missions was carried out within the diagram, also through a play of colours. The diagram (Figure 24) was conceived considering that all missions are executed by SROC. Visible actors, on the other hand, as the subjects of the missions were considered. For this reason, the arrows go from missions to actors.

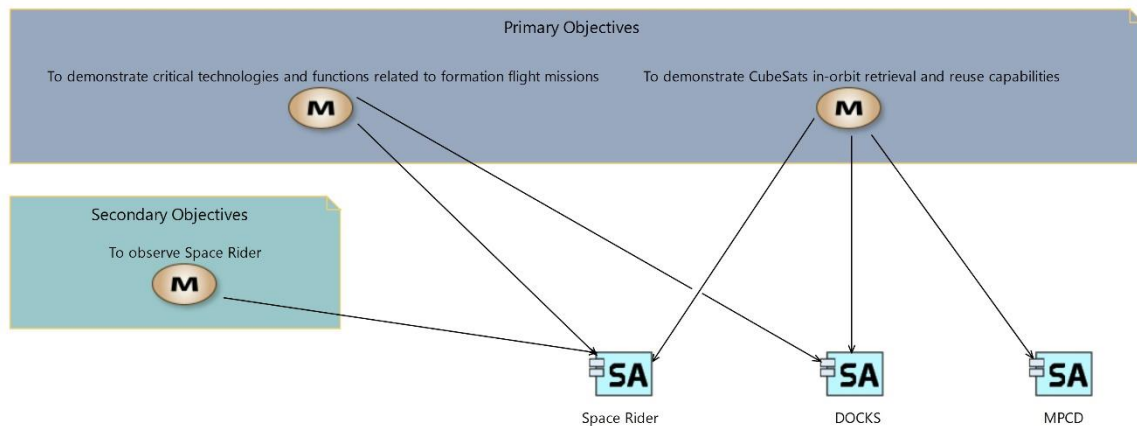


Figure 24: MB

To carry out the missions it was considered appropriate to refer them to the phases within which they can be performed. The Mission Capabilities Blank (MCB) made it possible to define SROC nominal and off-nominal concepts of operations, the mission phases and to create the appropriate links with them. The phases were defined as “System Capabilities” that the satellite must be able to accomplish, through a set of combined functions, to achieve its purpose. For each of them, thanks to the “properties window”, the initial and final conditions for their development have been entered (Figure 25). From an MBSE perspective, this functionality played a relevant role in summarizing all the relevant information in a single model.

The images below show the two MCBs made. The first (Figure 26) is dedicated to the description of the nominal ConOps in which observation and retrieving are carried out. It comprises the “Capabilities”:

- Launch and Early Operations Phase (LEOP);
- Commissioning and Performance Verification Phase (CPVP);
- Proximity Operations Phase (POP);
- Docking and Retrieval Phase (DRP);
- End of Mission (observe and retrieve) - Decommissioning (EOM);

The second one (Figure 27) concerns the representation of ConOps considered off-nominal, in which the retrieving is not done, consisting of:

- Launch and Early Operations Phase (LEOP);
- Commissioning and Performance Verification Phase (CPVP);
- Proximity Operations Phase (POP);
- End of Mission (observe) - Decommissioning (EOM);

The main difference between the two graphs involves the distinct functions performed during the EOM phases.

Capella

Management

Description

Extensions




Name :

Launch and Early Operations Phase (LEOP)

Summary :

Pre-condition :

SROC/MPCD ready for integration into SR

Post-condition :

SROC separated from SR at a certain distance from it








Figure 25: LEOP Capability property window

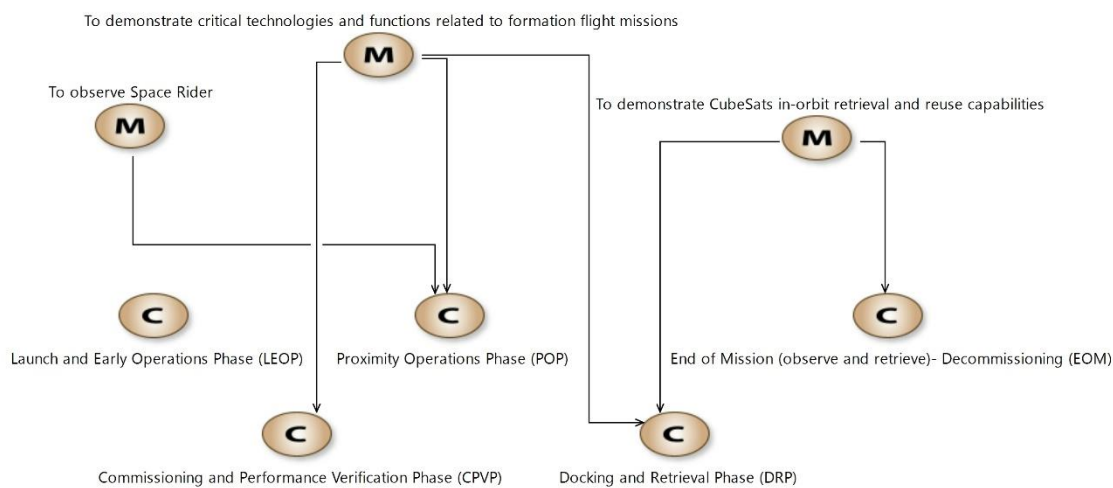


Figure 26: Observe and retrieve ConOps MCB

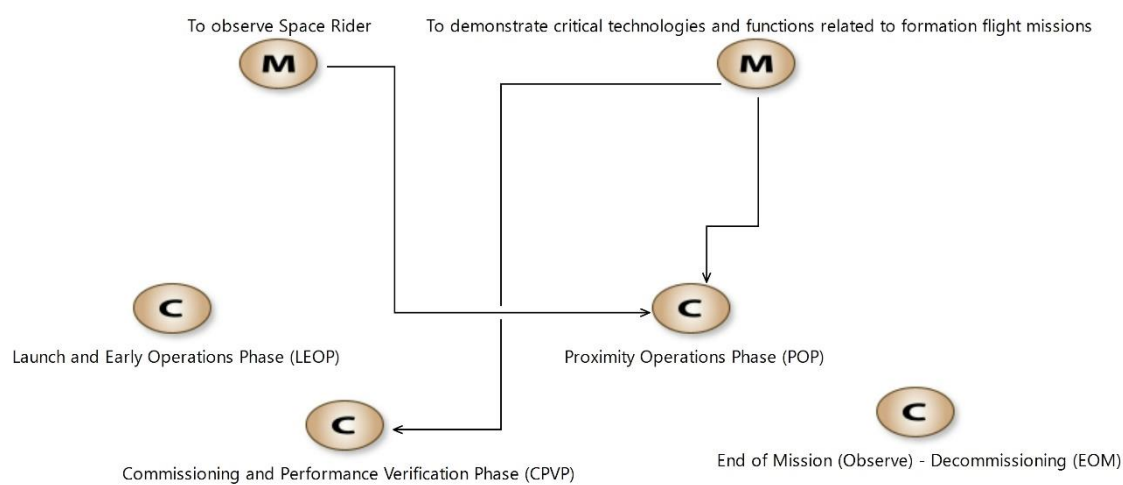


Figure 27: Observe ConOps MCB

In a similar way to the Operational Analysis, to accomplish the Capabilities, functions were defined.

The functions that form the capabilities have been grouped and arranged following an execution order using System Data Flow Blank (SDFB) diagrams. The same graphs let to identify the functional exchanges to accomplish the capabilities and show high-level functions (part of the mission phase) and subfunctions (the actual functions to be performed) in the same graph. Furthermore, functional chains with the name of the phase were created to be able to identify these sets of functions in other graphs. The SDFBs of the two EOM capabilities (with and without retrieve) are shown below to highlight their differences. The remaining graphs were placed in the annexes.

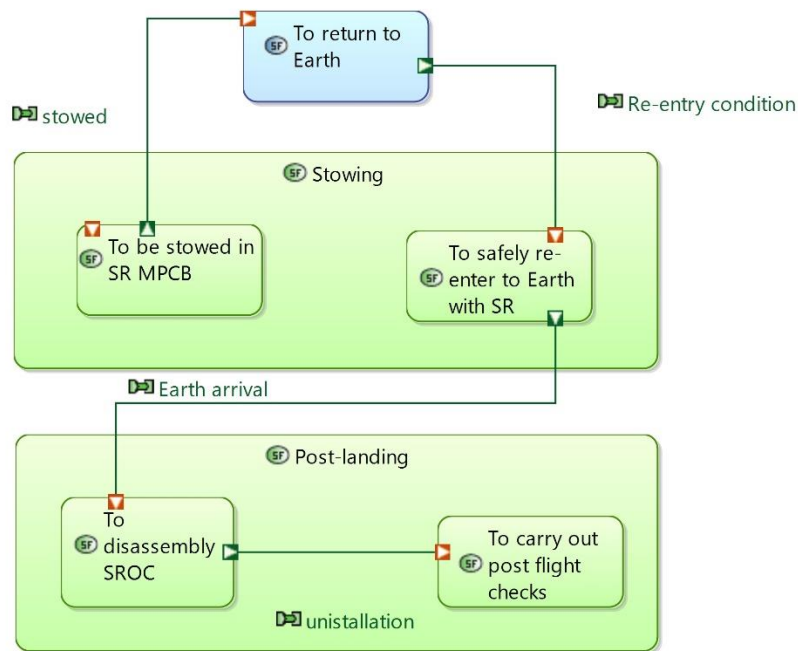


Figure 28: End of Mission (observe and retrieve) - Decommissioning (EOM) SDFB

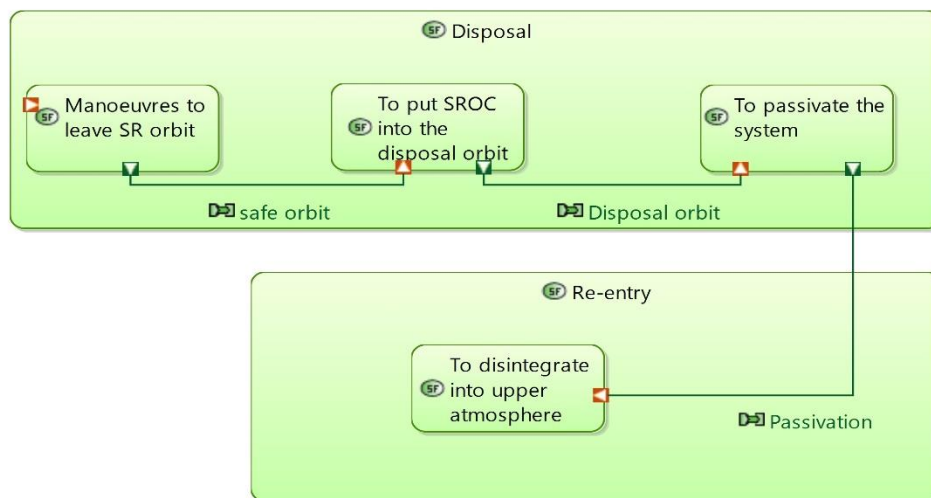


Figure 29: End of Mission (observe) - Decommissioning (EOM) SDFB

The work was completed by designing two SABs (System Actor Blank). The first one was created to identify the interfaces for the system and the actors. In red “Physical Interfaces” have been represented, which correspond to tangible connections between the graph elements (e.g., cables, pins), instead, in blue “Component Interfaces”, to indicate certain detached constraints (e.g., performances, operational quantities etc.) (Figure 30). The second one was created to provide an overall view of the system and the actors involved, the functions that they perform, concerning the different capabilities (highlighted through functional chains), considering both the nominal and off-nominal conditions (Figure 31, Figure 32).

Looking at the charts, it can be seen how the functional tree created in phase A has been broken down and addressed in this paragraph only as regards the functions relating to the mission, while it will be faced in the paragraph on logical architecture as regards the relative functions to subsystems.

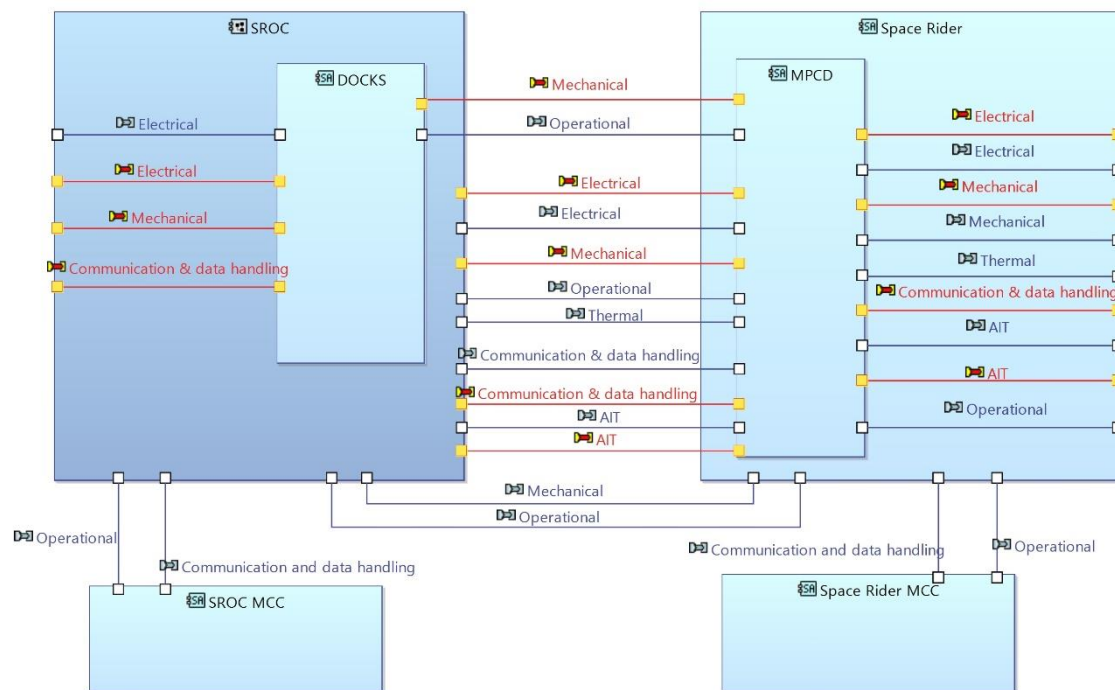


Figure 30: Interfaces SAB

3.3 Requirements

Stakeholders' requests collected at the beginning of the project are translated into system requirements that the system itself must comply with in order to fulfil their needs. These requests may initially be described by interested parties in non-technical language. The role of the systems engineer, in this case, is to transform the requests into sentences that respect certain rules (e.g., they must contain a single "shall" in the statements and must be verifiable) and organize them appropriately. Following the ESA ECSS standards, the requirements can be divided into the following categories:

- **Mission:** requirements and constraints that derive directly from the mission objectives and answer the questions "Why?", "When?", "Where?", "What?", "How many satellites", "For how long?"; for example, they deal with the orbit, duration, assembly, scientific objectives or mission success criteria.
- **System:** requirements that affect the high-level characteristics of the system and derive in most cases from those of the mission.
- **Functional:** they are obtained from the functional analysis and define how well the system must perform functions.
- **Environmental:** requirements that take into account the environment in which the product acts, for example considering vibrations, radiation or temperatures during the life cycle.
- **Operational:** they refer to the operation of the system, they include the different operating modes of the product (i.e. on the ground, in orbit) or general operations (i.e. communications, frequencies).
- **Interface:** category of requirements that traces the requests relating to the interfaces between different components of the product or between the product and the external environment.
- **Physical:** class of requirements deriving from requests regarding geometry, mass and inertia properties of the product, etc.
- **Configuration:** requirements concerning the parts that make up the product, their assembly and their organization.
- **Product guarantee:** class concerning product characteristics such as reliability, availability, maintainability, safety, quality.
- **Design:** requirements design, material standards, safety margins.

Requirements can be verified through testing, inspection, analysis, and demonstration [13].

3.3.1 Requirements phase 0/A

The designers have derived the requirements considering, in addition to the analysis of the stakeholders, the mission objectives, functional analysis, identification of scientific objectives, the study of critical technologies, mission elements and specifications imposed by ESA. They intended to define high-level requirements by detailing those for critical subsystems and technologies.

Already in the first phase of work, the Valispace tool was used for the collection and management of requirements, in order to take advantage of some model engineering features, such as the creation of Excel spreadsheets with the requirements to return to a document-centric approach.

During the writing the designers set, following ESA standard, that the requirements must be:

- trackable backwards and forwards, to determine their parental relationship;
- unique;
- unequivocal;
- verifiable;
- expressed through a sentence containing a “shall” statement, as mentioned in the foregoing paragraph.

The requirements were first organized through an ECSS-type subdivision, considering the different categories and then, when necessary, based on the product tree [52].

3.3.2 Valispace Requirements

The Valispace requirements module allows managing the project requirements by archiving, organizing and keeping track of them.

Several features can be exploited in this area, in particular, each requirement can be characterized through:

- A section: within which, through a drop-down menu, it is possible to indicate a class of requirements to which it belongs.
- A specification: to provide a further subdivision with respect to the section to which it belongs and identify a specific group. This is defined as “a type of dynamic document that contains and manages requirements” and can also be assigned to components.
- An identifier: which consists of a unique name for the requirement usually composed of a group of letters useful to indicate the kind of requirement and three or more numbers.
- A title: for briefly describing it.
- The text: to describe the requirement. “Valis” can be inserted inside it, i.e., numerical values linked to the components properties that can be updated automatically in case of changes.
- A rationale: to insert a comment or explanation.
- An image or attachments: to insert additional information in image format and attach documents relating to the requirements.
- Parent-child relationships: to indicate, utilizing lists, which requisites he derives from or to indicate which requisites derive from him.
- Type: to identify an additional membership field with respect to the specification, it can be used to provide information about the characteristics of the requirement.
- Status: to indicate whether the requirement is concluded, a draft or to be reviewed.
- The verification status, the date of the event and the author: to see if and when the requirement has been verified and indicate the user who carried out the verification.
- Verification methods: to indicate how the requirement is to be verified. The methods that can be selected are analysis, inspection, review, rules, tests.
- Components: to which requirements can be linked to indicate their membership.

- Closeout reference: to create a check through numerical values to report errors if not respected.
- Tag: to indicate particular conditions of the requirement [50].

In phase O/A the designers used:

- title;
- unique identifier (ID);
- level (tree of requirements compliance);
- statement (or text);
- traceability (parent/children);
- item (with respect to the product tree);
- owner;
- type;
- verification level;
- verification method.

3.3.3 Requirements delta A/B phases

The classification of requirements in the new phases has been changed from the previous ones. In this case, two folders were created through Valispace to identify the internal requirements, which are over the responsibility of the designer for the development of SROC, and external requirements, to be provided to external actors that are responsible for the MPCD, DOCKS, Vega and the MCC of SR. In this way, the subdivisions occur at the contractual level, facilitating the checks to be carried out.

For both folders, “specifications” were created to represent elements of the mission architecture or the product tree that have already been identified or will be identified through subsequent analyzes (e.g., spacecraft, mission, propulsion, etc.). Subsequently, the “sections” were used to identify topics in order to classify the requirements (i.e., thermal, radiation, proximity operations, mechanical). The “types” were employed to indicate the classification of the requirement according to the ECSS regulation. The requirements were linked to the individual components thanks to Valispace functionalities.

IDENTIFIER	↑	TEXT	SPECIFICATION	SECTION	TYPE
SROC-INT-SC-010		The SROC/DARM mechanical (static, sine and shock) loads shall be compliant with the maximum Space Rider loads [AD 1].	Spacecraft	Mechanical	Environment
SROC-INT-SC-020		The maximum de/re-pressurisation to which SROC/DARM shall be subjected to during respectively launch ascent and re-entry, is maximum 2200 Pa/s (TBC).	Spacecraft	Mechanical	Environment
SROC-INT-SC-030		SROC/DARM shall be compatible with the thermal environment of the Space Rider MPCB [AD 1].	Spacecraft	Thermal	Environment
SROC-INT-SC-040		The interface heat flux between SROC/DARM and the MPCB baseplate shall be less than 2 kW/m ² (TBC).	Spacecraft	Thermal	Environment
SROC-INT-SC-050		The SROC/DARM, when in the MPCB or in the proximity (< TBD km in-plane) of SR, shall keep its electro-magnetic radiation emission levels specified in the Space Rider User Manual [AD 1]. Note: Short-period temporary deviations from this requirement, e.g. during rendezvous and docking, shall be subject to prior approval by ESA.	Spacecraft	Radiation	Environment
SROC-INT-SC-060		SROC shall have a Collision Avoidance Manoeuvre (CAM) capability that places it in a trajectory that does not cross a predefined Keep Out Sphere (KOS) around Space Rider.	Spacecraft	Safety	Mission

Figure 33: Requirements examples

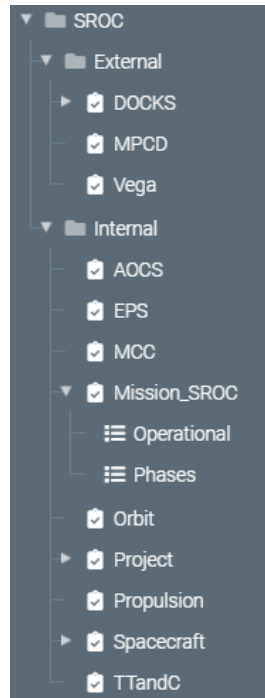


Figure 34: Valispace organization and example of requirements

3.4 Logical Architecture

The goal of this project phase is to outline how the system will work to meet customer expectations.

After defining the high-level functions through functional analysis, it is necessary to carry out a subsequent step and identify, by breaking them down into sub-functions, how they can be performed. These sub-functions are associated with logical components that interact through logical relationships and are independent of the physical components. This stage of work is necessary to move from the definition of the problem to that of the solution.

3.4.1 Logical Architecture phase 0/A

The designers in the initial phase of the project identified, considering the mission objectives, the outlined requirements and the functional analysis, the logical areas that SROC had to possess to carry out the mission. A list was created for them and then published in the document “D3.1 SROC Feasibility and Preliminary Specification” [52].

In particular, initially, the spacecraft was divided into two macro-regions, one dedicated to the payload and one to the support platform, subsequently, the logic components connected by electrical, data or mechanical interfaces were identified.

The following elements have been recognized for the platform:

- Command and data handling subsystem (CDH);
- Attitude Determination & Control, and Navigation subsystem (ADCNS);
- Propulsion subsystem (PROP);
- Proximity Relative Navigation subsystem (PRNS);
- Space to Earth communication subsystem;

- Space to Space communication subsystem;
- Docking subsystem;
- Power supply subsystem (EPS);
- Thermal control subsystem (TCS);
- Mechanical structural subsystem;

For the payload area:

- Primary payload;
- Secondary payload (when applicable) [52].

3.4.2 Logical Analysis: Capella architectural elements and diagrams

The logical architecture is defined by the Arcadia methodology with the statement “how the system will work to fulfil expectations”.

The main elements for the definition of the architecture are:

- Logical Component: a structural element that allows “a notional decomposition of the system independently of the technological solution”. The logical component can be divided into logical subcomponents and one or more logical functions can be associated with it. It can be linked to logical components and actors.
- Logical Actor: any element that does not belong to the system but that interacts with it.
- Logical Function: representing a “behaviour” or “service” provided by a logical component or actor. The user can connect it via the input/output flow ports to other logical functions and can be broken down into sub-functions.
- Functional Exchange: element of the architecture that is created by connecting two logical functions and represents a “one-way” exchange of data or matter.
- Component Exchange: that identifies the interactions between logical components and/or logical actors and allows “the circulation of functional exchanges”.
- Logical Scenario: dynamic sequence diagram that “describes the interactions between logical components and logical actors in the context of a capability”.
- Functional Chain: an element that, as seen in the system analysis, allows the identification of function paths [24], [51].

The logical architecture was realized through:

- Logical Architecture Blank (LAB): an architecture diagram that shows the constituent elements of the system at a logical level and how the functions are allocated within the logical components, contained within the system or actors. Unidirectional or bidirectional components exchanges and function exchanges can be produced, always connecting one door to another, with the option of highlighting chains of logical functions.



Figure 35: From the left symbols of “logical function”, “logical component”, “functional exchange”, “component exchange” [51]

3.4.3 Logical analysis phases delta A/B1

The construction of the logical architecture in this second phase of the project was carried out with the aim of enriching and further deepening, through small conceptual changes, the analysis previously performed.

In order to ensure continuity with the system analysis, Capella autonomously makes a “transition” to the new level of the SROC system. It is also possible to make a voluntary transition of the actors, with which the system interacts during the mission, but in this place, it was chosen to focus only on the CubeSat.

The LAB diagram made it possible to define two logical macro-areas within the system: the payload and the platform. The first is necessary to achieve the objectives regarding the SR observation analysis and the second to support the functioning of the payload and the achievement of the remaining missions.

The platform was subsequently broken down, after several brainstorming sessions, into further logical blocks capable of covering all fields of interest to complete the mission. They, differently from phase O/A, have not been defined as “subsystems” to try, as much as possible, to remain free from the physical world.

The identified blocks were:

- “Power Collection and Control” to generate, regulate, store, and distribute power to the spacecraft.
- “Command and Data Handling”, further divided into a part dedicated to the commands to receive, store, process, and distribute them and one to the management of data to collect, store, process and downlink the mission data and spacecraft telemetry.
- “Attitude & Orbit Determination and Control”, to determine the spacecraft rotational and translational motion, to maintain and reorient the spacecraft or change the spacecraft orbit or trajectory.
- “Structure and Mechanisms”, which contains both the structural function and the docking function considered separately in phase O/A, to provide mechanical support and house spacecraft payloads and subsystems.
- “Propulsion” to provide, monitor, and control spacecraft thrust.
- “Proximity and Relative Navigation” in order to determine the spacecraft rotational and translational motion, maintain and re-orient the spacecraft and or change the spacecraft's orbit or trajectory with respect to SR.
- “Thermal Determination and Control” to regulate spacecraft temperatures and ensure operability.
- “Communication” to provide the communication link between the spacecraft and the ground system or with another spacecraft.

Subsequently, the logical functions performed were allocated within the different areas in order to have an overall view of the different roles. And finally, functional exchanges between the functions and the exchanges between the components were created. The former identifies the outputs of each logical region, the last describes the different types of interfaces between logical components. In general, three types of interfaces have been identified: mechanical support, energy and data.

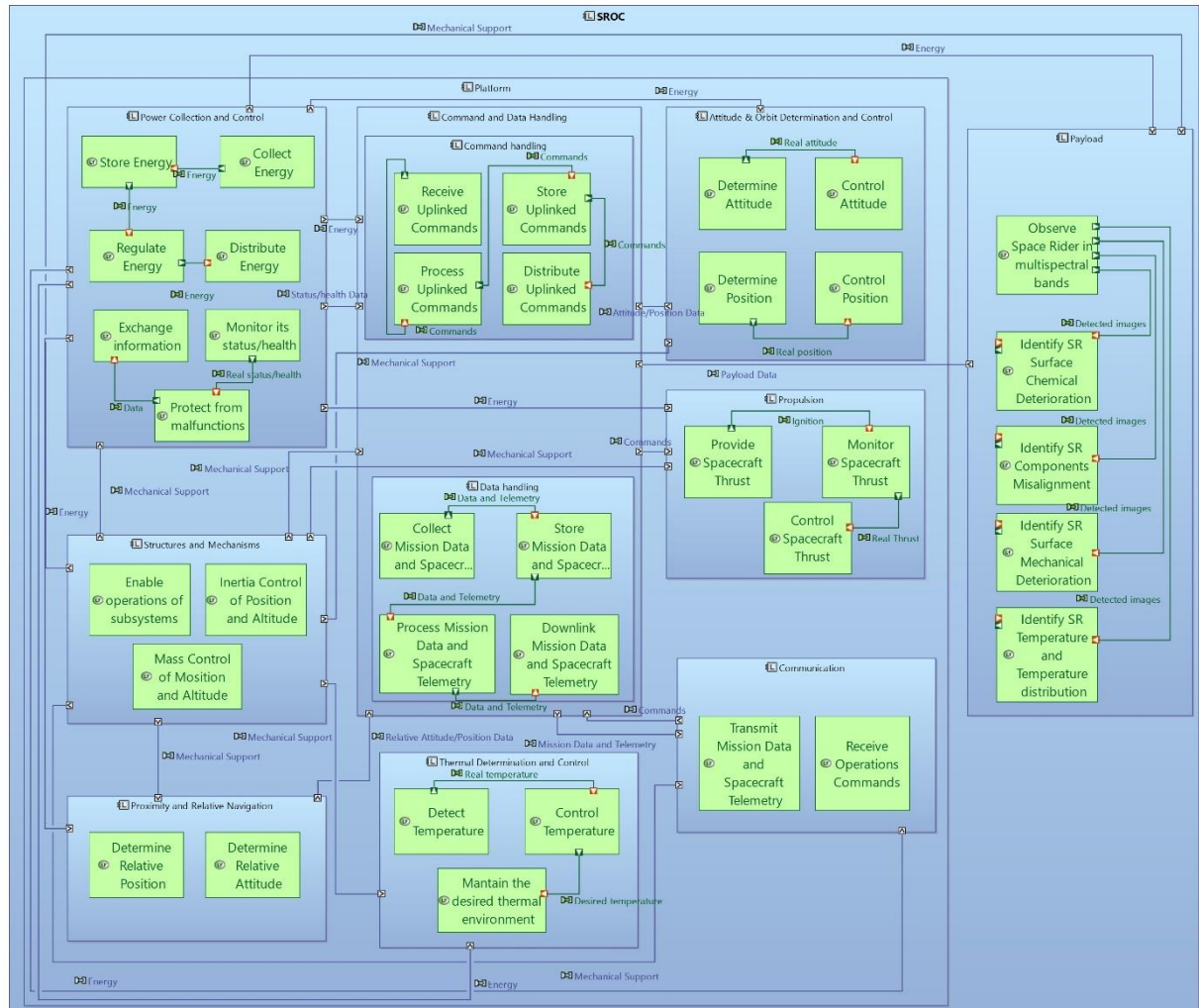


Figure 36: LAB

3.5 Physical Architecture

Physical architecture represents the physical elements that constitute the system and its physical interfaces. The designer in creating it aims to develop a technical solution to the logical architecture, remembering that, despite the physical components derived from the logical ones, very often a system function performed by a single logical component can be executed by one or more physical components.

In the thesis, this architecture was created in both tools, on Capella to define the interfaces and the functions performed by the products, while on Valispace to implement the characteristics of the different components and to carry out evaluations and budgets.

3.5.1 Physical Architecture phase 0/A

From a physical point of view, the designers, after assessing trade-off analyzes, created a “shopping list” of spacecraft components.

For the bus, the following have been identified:

- a self-standing avionics module, consisting of an on-board computer, a coprocessor for the ADCS, sensors and actuators to perform a fine control of the attitude (i.e. 3 reaction wheels, 2 star trackers, 1 triaxial IMU);
- a backplane that constitutes the printed circuit board assembly (PCBA) to ensure the interconnection between the subsystems;
- sensor modules arranged inside the spacecraft to measure temperatures and for the ADCS system (i.e. magnetometers and sun sensors);
- a thrusters module to carry out the propulsion;
- a set of cameras in the visible and infrared range to perform proximity detections, a proximity lidar and an Optical Navigation Board;
- the GPS module, with a patch antenna;
- a torque rods module consisting of 3 ferrite solenoids that operate as magnetorquers to desaturate the RWs and to accomplish an attitude control;
- an LDR radio for TT&C satellite-earth communication with a UHF transceiver with L-dipole;
- an HDR radio for earth satellite communication in S-band;
- a battery module, consisting of 6 lithium ion cells managed by a control board;
- a module dedicated to solar panels with an MPPT power controller;
- 12U structure with aluminium panels;

A Hyperscout 2 hyperspectral camera was chosen for the payload [52].

With the characteristics determined for the different components, different budgets have been made (i.e. power budget, energy budget, link budget, etc.) which will be shown later.

3.5.2 Physical Analysis: Capella architectural elements and diagrams

The physical architecture is defined in “P. Roques, Systems Architecture Modeling with the Arcadia Method, A practical guide to Capella, 2018” as “how the system will be developed and built” and allows defining, as previously written, the architecture of the system.

The main elements to create this architecture are:

- Physical component: artefacts that describe the structure of the physical system. There are two types of components in Capella: “behaviour physical components” to perform the assigned physical functions (i.e., operating software, radar antenna, etc.), “node physical components” which provide the material resources for the behavioural components (i.e., processor, router).
- Physical port: called “connection port” when it belongs to a “node physical components” or “structural port” when created on a behavioural component.
- Physical function: “function applied at a physical level”.
- Physical connection: a non-oriented physical connection between components of the architecture (e.g., ethernet cable, USB cable, etc.).
- Physical path: an “organized set of physical links” between several different components [24], [51].

The modelling of the architecture took place through the use of the following diagram:

- **Physical Architectural Blank (PAB):** similar to the previous cases, this architecture diagram allows associating the performed functions with each component, to create component exchanges and functional exchanges. It also offers the capability to allocate behaviour physical components within node physical components and create functional chains.



Figure 37: From the left symbols of “physical function”, “behavioral physical component”, “node physical component”, “physical connection”, “physical path”[51]

3.5.3 Valispace components module

The system architecture can be also defined in Valispace “components” module to elicit numerical values.

Components, as indicated in the tool documentation, are “final product elements” which can be physical parts, such as solar panels, or logical parts, such as operations or orbits. The user can hierarchically structure them in a components tree, identifying the superior system and the sub-components of which it is composed.

A section is dedicated to each component defining details such as the “Id”, which let the tool uniquely identify it, an image and documents, for which different versions can be loaded based on the update, a tag, a part number and properties.

The latter can be of different types:

- “Vali”: properties containing a single value, such as mass or volume;
- “Vali matrices”: groups of Vali arranged in rows and columns, e.g., used for components with multiple operating modes, such as power consumption;
- “Textvali”: property for inserting text;
- “Datevali”: that stores information about the date;
- “Dataset”: property to import or add a set of values and create a chart.

Furthermore, for the different Valis, users can define the unit of measurement, the type, a formula, the description, margins of error on the value, add the minimum and maximum requirements, which cause an error message to appear if they are exceeded. Within the component section, thanks to various mathematical functions, it is possible to perform calculations of various kinds.

Valis are called Valitypes if they are defined as default properties in a section dedicated to the setting of the tool.

In the initial stages of the project, users may have to make comparisons between different components, to choose a suitable solution. For this reason, Valispace allows the user to create and compare alternatives through the “alternative container” function and see how they would affect the project. Every time a different solution is chosen, the calculations are carried out again.

The user can also use the “copy-paste” function to create multiple similar components or the “copy and connect” function for identical components, to update them all at the same time in case of changes.

When a Vali is changed, it is saved in a database that can be accessed for a possible restore using the “Time machine” function. When changes are made by different users, a user can receive notifications to stay informed.

An important functionality offered by Valispace in the components section is to be able to create operating modes. A satellite can face different mission phases within which some components may have diverse operating modes (active, off, energy saving). When this happens, therefore, some properties will not remain constant throughout the mission but will vary according to the modes.

The tool thus allows the creation of the operating modes for the individual components and to associate them with the properties that can vary, to generate the operating modes of the overall system and to implement connection matrices through which to indicate the states of the components in the different system modes. Thanks to these matrices and to the “SOC” (sum of children) function, which make a summation of all the same type properties in an equal level architecture, it is possible to identify the overall value of that Vali for a given system mode in the whole.

“SOC” function allows making budgets that can be viewed not only through numerical values but also through pie charts and tables [50].

3.5.4 Physical architecture phases delta A/B1

The physical architecture was modelled in Capella only partially by re-proposing what was done in the work phase O/A since there had been no changes from a physical point of view compared to the previous phase and no new components or interfaces had been defined.

The model was built by designing a product tree thanks to the possibility of creating “behaviour components” and “node components” that helped to distinguish subsystems from equipment. The construction of the tree made it possible to translate the work on the Valispace tool where the different physical characteristics, useful for carrying out the budgets, were inserted.

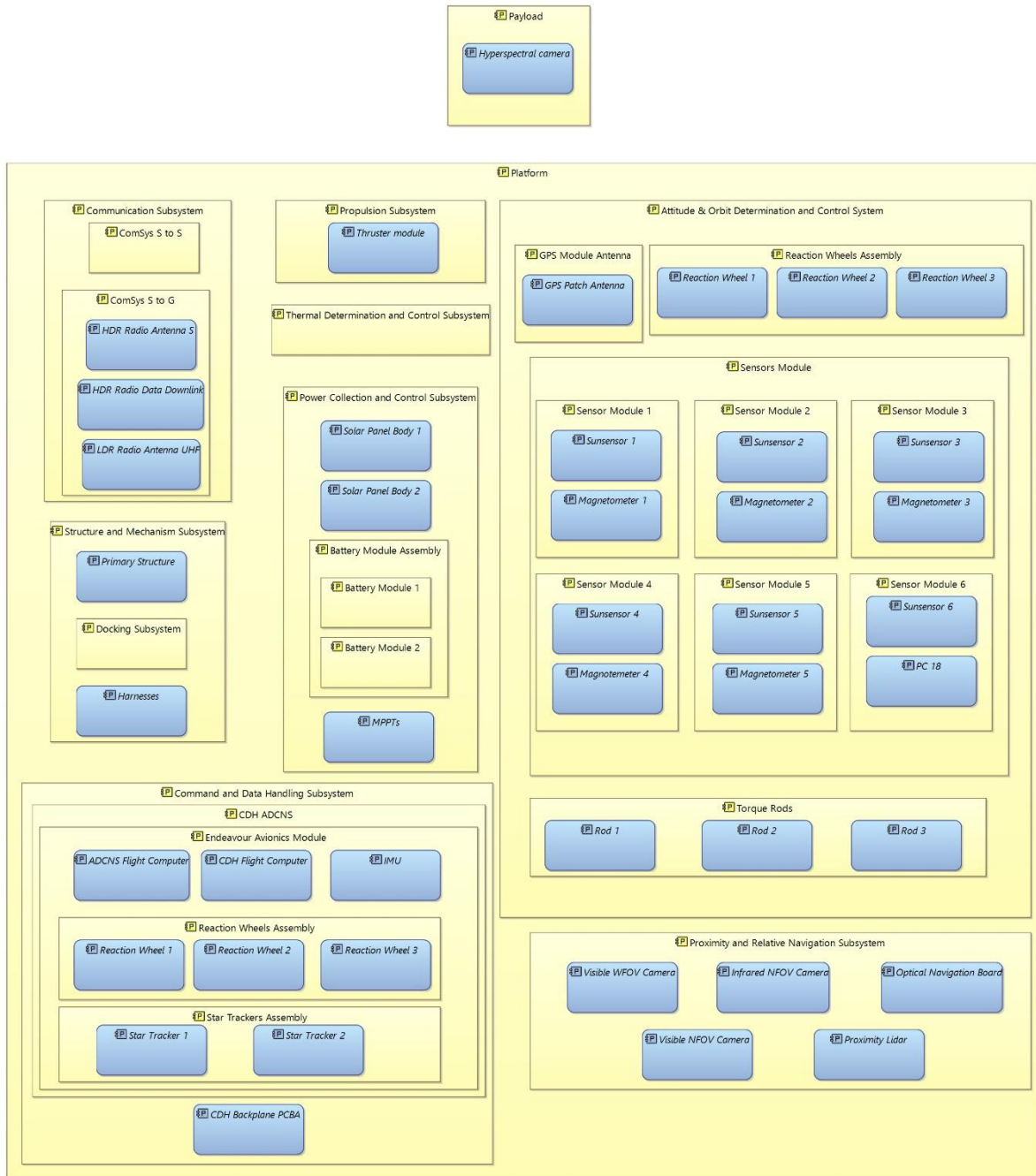


Figure 38: PAB

The work on Valispace was organized to reflect what was done in Capella and to do this the product tree was created through hierarchical relationships. The latter has thus allowed through the “SOC” function to create volume and mass budgets at each hierarchical level.

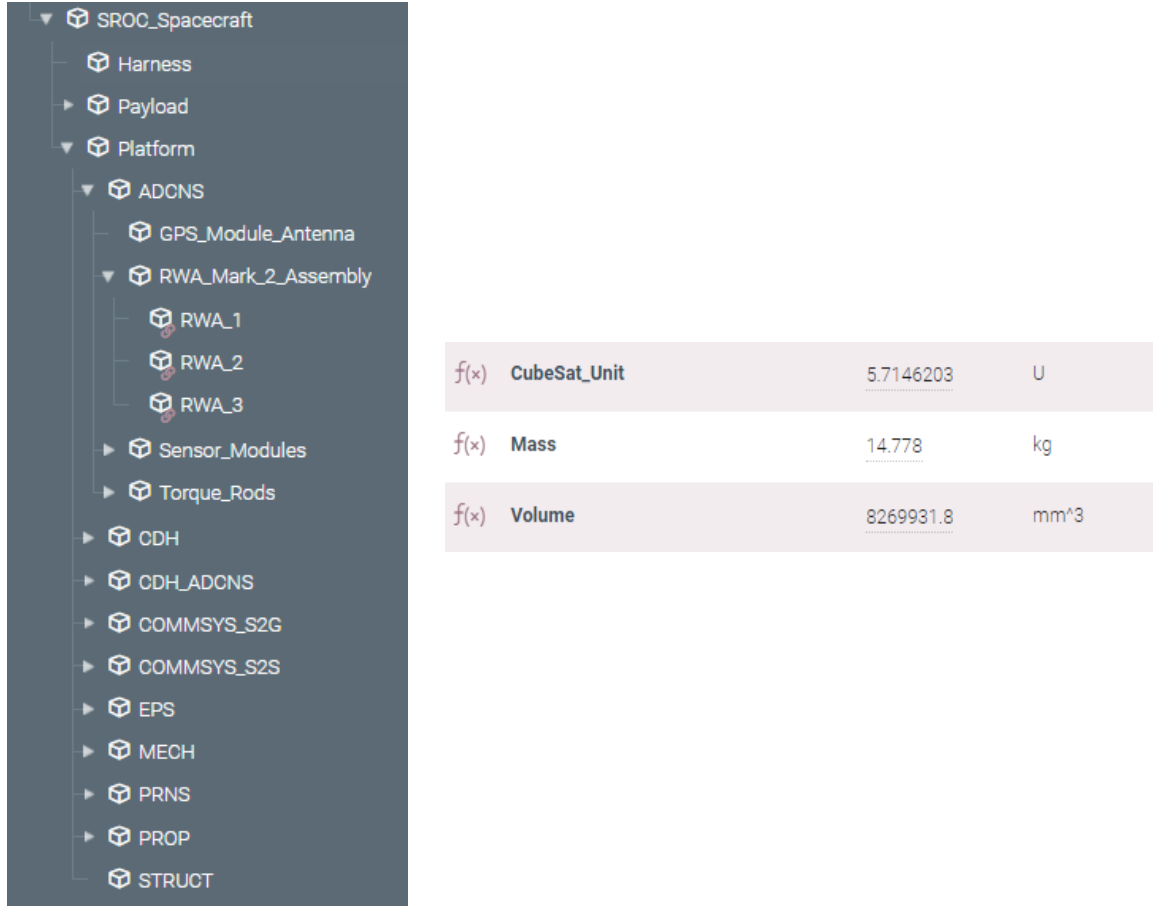


Figure 39: Valispace physical tree (left), satellite mass and volume budget (right)

3.6 Budgets

After considering the components that make up the system from a physical point of view, thanks to their characteristics it was possible to calculate different types of budgets. This study describes the power, energy and link budgets for the SROC satellite, implemented during the first project phase and the changes and improvements made by moving to an MBSE approach in delta A and B1 phases. The objective of this last phases was, indeed, to move from an approach document-centric, centred on many worksheets on which to make the desired estimates, as done in the previous phase of the project, to a model-centric one, where a single platform is used, within which several users simultaneously have the possibility of carrying out calculations, storing data and results.

Following this objective it will be possible to see how, the power budget, an evaluation of the total power produced by the CubeSat during the mission, and the link budget, “the capability of the communication system to guarantee an effective communication between space segment and ground segment by defining the relationship among data rate, antenna size, propagation path length, and transmitter power”, have been fully computed on Valispace while the energy budget, “the balance of energy income against energy expenditure”, has been implemented on Matlab connected via an appropriate add-on to Valispace which, in this case, has been used as a repository to export and import data [53].

3.6.1 Power budget and Energy budget phase o/A

In phase o/A of the SROC mission, the power and energy budgets were treated together through calculations made on Excel worksheets.

Following the ESA regulations, a margin error of 20% was considered for the system, while for the individual components margins of 5%, 10% or 20% were implemented; the first for components with flight heritage, the second for components that needed small changes and the last for components that needed major changes.

The designers to develop the analysis considered seven operating modes: safe, Sunpointing, downlink, free flight, observation, manoeuvres and docking, for which they evaluated the powers required by the individual components and their sum, in order to obtain a subtotal value, and the duty cycles (the operating percentage duration with respect to the total) of the components and of the individual modes.

On the subtotal power consumptions values, they applied the ESA margin to determine the power required during the various modes. Finally, through a sum-product operation between the values of the required powers and the modes duty cycles, they obtained an “average required power”.

The energy budget, instead, as reported in the document [52], was carried out, in a preliminary way, for the different operating modes, including Sunpointing, considering the degradation of the solar cells, their efficiencies and any misalignments, the effects of eclipses and an average incidence of the solar angle of the panels. The engineers chose to place the solar panels directly on the body, in order to reduce complexity, by arranging them on 3 faces. In general, in this phase, thanks to the parameters listed above, the designers evaluated the power generated in Sunpointing and the “orbital average power” OAP generated in the other modes, obtained by multiplying the power generated in Sunpointing by a parameter capable of considering any losses, known as the “rule of thumb”, and thanks to these they obtained the “average generated power” equal to $P_{EOL} \cdot dutycycle_{Sunpointing} + OAP (1 - dutycycle_{Sunpointing})$. This value was thus compared with the satellite “average required power” and with those required during the different modes in order to identify the parameters to be modified.

Power budget	Safe	Sunpointing	Downlink	Free Flight	Observation	Manoeuvre	Docking
Endeavour Avionics Module	0.00	0.00	0.00	0.00	0.00	0.00	0.00
- CDH Flight Computer	0.70	0.70	0.70	0.70	0.70	0.70	0.70
- ADCNS Flight Computer	0.80	0.80	0.80	0.80	0.80	0.80	0.80
- RWA	0.00	0.00	0.00	0.00	0.00	0.00	0.00
- Star Trackers	0.00	0.90	0.90	0.90	0.90	0.90	0.90
- IMU	0.25	0.25	0.25	0.25	0.25	0.25	0.25
Sensor Modules	0.10	0.10	0.10	0.10	0.10	0.10	0.10
GPS Module	0.50	1.00	1.00	1.00	1.00	1.00	0.00
Torque Rods	0.18	0.18	0.18	0.18	0.18	0.18	0.00
RWA Mark 2 Assembly	0.00	3.00	3.00	3.00	3.00	3.00	3.00
Thruster Module	0.00	0.00	0.00	0.00	0.00	8.50	3.30
Visible NFOV Camera	0.00	0.00	0.00	0.02	0.02	0.02	0.00
Visible WFOV Camera	0.00	0.00	0.00	0.02	0.02	0.02	0.10
Infrared NFOV Camera	0.00	0.00	0.00	0.06	0.06	0.06	0.30
Proximity Lidar	0.00	0.00	0.00	0.00	0.00	0.00	2.00
Optical Navigation Board	0.00	0.00	0.00	1.00	1.00	1.00	1.00
LDR Radio for S-G TT&C	0.25	0.25	0.25	0.25	0.25	0.25	2.45
HDR Radio for S-G Data	0.00	0.00	14.00	0.00	0.00	0.00	0.00
LDR Radio for S-G TT&C	0.00	0.00	0.00	0.25	0.25	0.25	2.45
Docking Subsystem	0.00	0.00	0.00	0.00	0.00	0.00	1.20
Battery Module Assembly	0.55	0.55	0.55	0.55	0.55	0.55	0.55
Hyperspectral Payload	0.00	0.00	0.00	0.00	12.00	0.00	0.00
Subtotal Power Consumption	3.33	7.73	21.73	9.07	21.07	17.57	19.10
Required Power Generation	3.84	8.92	25.09	10.48	24.34	20.29	22.06
Duty Cycle	0%	0%	10%	90%	0%	0%	0%
Average Required Power	11.94						
Power generated in Sunpointing	27						
OAP generated in other modes	12						
Average Generated Power	12.00						

Figure 40: Example of Tyvak Power Budget during Downlink orbits phase o/A [52]

3.6.2 Link budget phase o/A

In the first phase of work, Tyvak international designers calculated the link budgets, considering both the UHF and S bands, for communications between SROC and ground stations. The evaluations were performed using the E_b/N_0 method, in which this ratio corresponds to the energy required to transmit a bit (E_b) in presence of white noise (N_0), choosing a link margin threshold of 3 dB and assuming link-related losses. The Carrier to Noise ratio, which indicates the receiving quality of the antenna was also considered.

All calculations were done on different Excel worksheets.

For the UHF band, both downlink and uplink link budgets have been computed. The considered ground stations were Tyvak UHF stations equipped with Yagi antennas with outputs of 120 W RF and gain of about 16 dBi. The antenna considered for the spacecraft was instead an omnidirectional V-dipole transceiver with 1 W RF power. The assumptions related to the losses concerned: the atmospheric losses of 2 dB, the polarization losses of 3 dB and the losses due to misalignment of 3 dB.

For the S-band, the budget was made considering only the downlink toward KSAT ground stations capable of receiving on dishes with a G/T (gain over equivalent temperature) of 12.5 dB/K, and a satellite antenna with a transmitter output of 2 W RF power and 7 dBi gain. In this case, an atmospheric loss of 2 dB, a polarization loss of 3 dB and a misalignment loss of 6 dB were considered.

The other parameters used for the calculations were derived from the components' technical specifications.

The designers obtained margin values higher than 3 dBi, for the UHF band, already considering 2° of elevation and an orbit of 500 km, conservative compared to the nominal one of 400 km, for the S-band, instead, an elevation of 15° and orbits of 500 km.

The procedure carried out for making the budget in UHF band is explained below, considering the same equations for the downlink and the uplink by changing in the two cases the reference values for the transmitter and receiver, in the first case the transmitter for the spacecraft and the receiver for the ground station, in the second case the opposite condition. In the equations, the subscript “t” will be used to indicate the transmitter and the subscript “r” for the receiver.

The following data were used for the transmitter: the bandwidth in kHz, the gain G_t in dB, the transmission power (T_x) P_t in W, the signal frequency f in MHz, the data rate in bps, the losses along the transmission line between transmitter and antenna (proportional to the length of the cable and the type of cable) L_t in dB, the noise bandwidth of the transmitter calculated as $Noise\ bandwidth = 10 \log_{10} Bandwidth_t$ and the Effective Isotropic Radiated Power ($EIRP_t$) equal to $10 \log_{10}(P_t) - L_t + G_t$.

For the receiver, instead, the following were considered: a reference temperature T_0 in K, its actual temperature T_{r_a} in K, the noise figure in dB, the gain G_r in dB, the line loss L_r in dB, an additional loss called *Implementation losses* in dB (i.e., component efficiencies), a noise figure F in dB and the required ratio E_b/N_0 , determined by Bit Error Rate (BER that defines the probability that a wrong transmission of a bit occurs), modulation and the coding if any.

Other parameters useful for the calculations were the atmospheric losses L_a in dB , the polarization losses L_p in dB , the losses due to misalignment L_m in dB , the elevation angle δ in rad , the radius of the earth R_{Earth} in km , the average distance of the satellite from the ground station during communications h in km , the Boltzmann constant $k = -228.6 \text{ dBW/HzK}$, the slant range calculated with the following equation:

$$\text{Link distance} = S = R_{Earth} \left(\sqrt{\left(\frac{(R_{Earth}+h)^2}{R_{Earth}^2} \right) - (\cos\delta)^2} - \sin\delta \right).$$

In order the following parameters were computed:

- *Free Space Path Loss* $= 20 \log_{10} \text{Link Distance} + 20 \log_{10} \text{frequency}_t + 32,4$ that represents the sum of the link distance and the transmitter frequency in dB ; the $32,4 \text{ dB}$ value is linked in this case to the unit of measurement used for the frequency;
- The total loss due to the link *Propagation loss* $= \text{Free Space Path Loss} + L_a + L_p + L_m$;
- The isotropic receive level $IRL_{r_a} = EIRP_{t_a} - \text{Propagation loss}$ which corresponds to the power level it would expect to achieve using an isotropic antenna (uniformly omnidirectional antenna with 0 dB gain) [54];
- The signal level of the receiver *Rx signal level* $= IRL_{r_a} + G_r - L_r$;
- The equivalent temperature $T_e = T_{r_a} + T_0(10^{\frac{\text{Noise figure}}{10}} - 1)$;
- The ratio between gain and equivalent temperature T_e of the receiver in dB *Receiver antenna* $\frac{G}{T} = G_r - 10 \log_{10} T_e$;
- The white noise $N_0 = +k + 10 \log_{10} T_e$;
- The energy per bit equal to $E_b = \text{Rx signal level} - 10 \log_{10} \text{Data rate}_t$;
- The budget total noise as *Total Noise Power* $= N_0 + \text{Noise Bandwidth}$, considering the bandwidth for the noise bandwidth in Hz ;
- The ratio $\frac{E_b}{N_0} = E_b - N_0$;
- The carrier to noise density ratio, that determines if the receiver can lock on to the carrier and if the information encoded in the signal can be retrieved, considering noise presence in the received signal, $\frac{C}{N_0} = IRL + \text{Receiver antenna } G/T - k$;
- The Carrier to Noise ratio $\frac{C}{N} = \frac{C}{N_0} - \text{Noise Bandwidth}$, with N received noise power after filters;
- The *Link Margin* $= \frac{E_b}{N_0} - \left(\frac{E_b}{N_0} \right)_{\text{required}} - \text{Implementation Losses}$.

In the S-band downlink communication, instead, were used for the transmitter: the bandwidth in kHz , the gain G_t in dB , the transmission power P_t in W , the signal frequency f in MHz , the data rate in bps , the line losses L_t in dB , the noise bandwidth in dB and the $EIRP_t$ in dBW .

For the receiver: the required ratio E_b/N_0 and the implementation loss in dB . The designers considered, furthermore, as mentioned before, the receiver antenna ratio G/T of the ground station as known.

The link parameters used have been described previously.

In this case, to determine the link margin the G/T ratio was exploited through the following operations:

- $Free\ Space\ Path\ Loss = 20 \log_{10} Link\ Distance + 20 \log_{10} frequency_t + 32,4$;
- $Propagation\ loss = Free\ Space\ Path\ Loss + L_a + L_p + L_m$;
- $IRL_{r_a} = EIRP_{t_a} - Propagation\ loss$;
- $\frac{E_b}{N_0} = G/T + IRL_{r_a} - L_r - 10 \log_{10} Data\ Rate - k$;
- $\frac{C}{N_0} = EIRP - Propagation\ loss + G/T - k$;
- $\frac{C}{N} = \frac{C}{N_0} - Noise\ bandwidth$;
- $Link\ margin = \frac{E_b}{N_0} - \left(\frac{E_b}{N_0}\right)_{required} - Implementation\ Loss$.

Spacecraft			Link			Ground Station		
Tx Power	1	W	Link distance	2360	km	Tx Power	120	W
Antenna peak gain	0	dBi	Atmospheric loss	2	dB	Antenna peak gain	16.2	dBi
Line loss	0.5	dB	Polarization loss	3	dB	Line loss	1	dB
Noise figure	1	dB	Misalignment loss	3	dB	Noise figure	2	dB
Frequency (Downlink)	401	MHz				Frequency (Uplink)	401	MHz
Data rate (Downlink)	9600	bps				Data rate (Uplink)	9600	bps
Bandwidth (Downlink)	16000	kHz				Bandwidth (Uplink)	16000	kHz
Antenna temperature	290	K				Antenna temperature	290	K
Reference temperature	290	K				Reference temperature	290	K
Uplink required Eb/N0	12.6	dB				Downlink required Eb/N0	12.6	dB
Implementation loss	1	dB				Implementation loss	1	dB
Spacecraft EIRP	-0.50	dBW				Ground station EIRP	35.99	dBW
UPLINK						DOWNLINK		
Free space loss uplink	151.92	dB				Free space loss downlink	151.92	dB
Propagation loss uplink	159.92	dB				Propagation loss downlink	159.92	dB
IRL at spacecraft antenna	-123.93	dBW				IRL at ground station antenna	-160.42	dBW
Rx signal level	-124.43	dBW				Rx signal level	-145.22	dBW
Equivalent temperature	365.09	K				Equivalent temperature	459.62	K
Spacecraft antenna G/T	-25.62	dB/K				Ground station antenna G/T	-10.42	dB/K
N0	-202.98	dBW/Hz				N0	-201.98	dBW/Hz
Eb	-164.25	dBW/Hz				Eb	-185.05	dBW/Hz
Total Noise Power	-130.93	dBW				Total Noise Power	-129.93	dBW
Uplink Eb/N0	38.72	dB				Downlink Eb/N0	16.93	dB
Uplink C/N0	79.05	dBHz				Downlink C/N0	57.75	dBHz
Uplink C/N	7.00	dBHz				Downlink C/N	-14.29	dBHz
Link margin uplink	25.12	dB				Link margin downlink	3.33	dB

Figure 41: Tyvak link budget phase o/A [52]

3.6.3 Power budget phases delta A/B1

The power budget was implemented on Valispace retracing the work done by Tyvak designers.

The assessments previously created on Excel were easily reproduced on the new platform by exploiting the different potentials.

In detail, the possibility of creating modes was initially used both for the mission, considering its different phases, and for the components, for the different consumptions, and subsequently, according to the operating mode of the different parts, the sum of the involved powers was made through the “SOC” function.

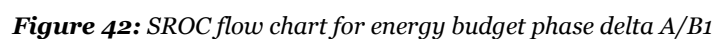
3.6.4 Energy budget phases delta A/B1

The energy budget, as said before, was realized through a connection between Valispace and Matlab considering, not only an average solar incidence angle on the panels, as hypothesized in the preliminary study by the Tyvak designers, but different inclinations (different cosine losses) as the orbit varies thanks to the matrices imported from STK. The latter, called Sun Vectors matrices, in detail, contained the cosines of the sun angle incidences θ_i with respect to the satellite reference axes changing along the orbits.

Valispace was used as a “warehouse” from which to draw and deposit data, the mathematical operations, on the other hand, were carried out on Matlab. This was possible thanks to a Matlab Valispace add-on that allows logging into the repository, to select, through an associated identification code, one or more component Valis, in the form of value, dataset, or matrix, to import them and in the same way to export the desired values.

The complete procedure for carrying out the computations is reported in the next chapter in the ModelSat energy budget paragraph (4.7) in which it is generalized. Unlike that version, however, in the case of SROC, the panels’ disposition was already chosen for the faces so it cannot be changed and only the characteristics of the panels or their number can be modified.

This evaluation has generally made it possible to study various parameters including the power generated by the panels or the depth of discharge during critical operating modes and to verify the feasibility of the desired tasks over time. An example version of the code can be viewed in the annexes while below it can be possible to observe a flow chart that summarizes the operations carried out for the specific case (Figure 42).



3.6.5 Link budget phase delta A/B1

In order to move in a model-based orientation the link budget, in the new work phases, has been performed inside the “components” section of Valispace. The areas of the mission architecture involved for its realization were: the ground station where the characteristics of the transmitter and the antenna for the bands S and the UHF have been inserted, the communication subsystem of the spacecraft, to place the data relating to the transceivers for the two bands, and the “operation” area in which two sub-areas have been created, one linked to the constants, distances and communication losses and one to carry out the calculations of the link budget.

Compared to phase O/A in order to comply with the manual “Satellite communications systems engineering atmospheric effects, satellite link design and system performance” [55] and with the “Space Systems” lectures slides [53] and therefore facilitate the understanding, equations similar to the previous ones were used for the estimates but with slight differences in terms of nomenclature and parameters.

These differences are:

- the use of the “Total Space Losses” L_{TLS} terminology instead of “Propagation Loss” within which losses related to rain L_r are also considered, hence $L_{TLS} = \text{Free Space Path Loss} + L_a + L_p + L_m + L_r$;
- the introduction of the parameter “System Noise Temperature” linked to the noise generated by the components $T_s = T_{r_a} + T_0 \left(\frac{1-L_r}{L_r} \right) + T_0 \left(\frac{F-1}{L_r} \right)$, inside which it can be observed the presence of the term F “noise figure”, that indicates the noise growth starting from a reference temperature $T_0 = 290K$, the temperature of the antenna T_{r_a} and the losses of the receiver cables;

The System Noise Temperature was used in the equations to replace the “Equivalent Temperature” thanks to the relation $10 \log_{10} T_e = 10 \log_{10} T_s - L_r$, that leads for example to write the gain over temperature ratio as *Receiver antenna* $G/T = G_r - (10 \log_{10} T_s - L_r)$.

These changes, thus, allow writing the E_b/N_0 as

$$\frac{E_b}{N_0} = EIRP_t - L_{TLS} + G_r - k - 10 \log_{10} T_s - 10 \log_{10} \text{Data Rate}_t.$$

4 ModelSat

After having described the work for the SROC satellite, it is now possible to generalize what has been done through reverse engineering for similar mission models.

This chapter deals with the concept of ModelSat, which consists in the creation of an MBSE procedure to which the systems engineer can refer in the definition of the first phases of the project, consisting of advice, modelling proposals and calculation methods for the determination of budgets on Capella and Valispace, which can be also reproduced on different tools. ModelSat also includes a dynamic database modelled on Valispace, as will be shown later in the physical architecture, in which different alternatives products have been collected for the same component that the user can choose for the design of CubeSat.

In the following paragraphs, the previous chapter will be retraced, proposing solutions for determining stakeholders and their needs, functional analysis, requirements, logical and physical architecture, technical budgets.

4.1 Advice for MBSE stakeholders and needs identification

The project phase of the management and identification of the stakeholders and their needs can be modelled, as mentioned in the previous chapter, using the “Operational Analysis” section of Capella.

The first action of the analysis consists precisely in determining the stakeholders through brainstorming sessions. In this phase it is essential to distinguish the different figures interested in the project and to consider any subgroups with different interests and needs among them, moreover, it is appropriate to classify them based on their power and interest in the mission. The use of MBSE tools such as Capella, defining the stakeholders as “Operational Entities”, allows representing them in a single framework using OEB graphic.

To optimize the organization of work, after having conducted interviews to define all interested parties’ needs, it would be appropriate to include them within “high-level topics” in which the various entities are interested, such as safety, costs, the development of innovative missions, the increase of knowledge, dissemination of data. This could be useful for the systems engineer to create a hierarchical order of implementation before even studying the requests in detail.

The “high-level topics” can be defined as “Operational Capabilities”, the real needs, instead, as “Operational Activities” to be respected in order to satisfy the parties involved.

Capella's OBC graphs consent to observe which stakeholders are interested in the identified operational capabilities and at the same time to create, following the MBSE logic, collections of operational activities associated with them (OAIB and ES graphs).

Within the OAIB and ES graphs, which allow evaluating the relationships with stakeholders, when possible it is advisable to arrange the needs in a logical sequence to facilitate, in a later stage, how to satisfy them and create input/output exchanges. At this level, the exchanges can be conceived as “values” that represent what the interested parties want to achieve with their requests.

The example described by Pascal Roques in the manual “Systems Architecture Modeling with the Arcadia Method” identifies them as the product of activities.

In this case, there are no differences with respect to his proposal except in the design level of detail.

Finally, to make the most of the model-based and tool potential, it is worthwhile to create a graph capable of collecting all the models created (in this case OAB) and providing an overview of the needs, the stakeholders, the interactions identified and highlighting the different “high-level topics” to which they belong through chains of activities. This chart allows having an immediate, dynamic, complete and rich view of the general picture unlike the simple lists produced on Excel.

A flowchart designed to summarize the recommended procedure is shown below (Figure 43).

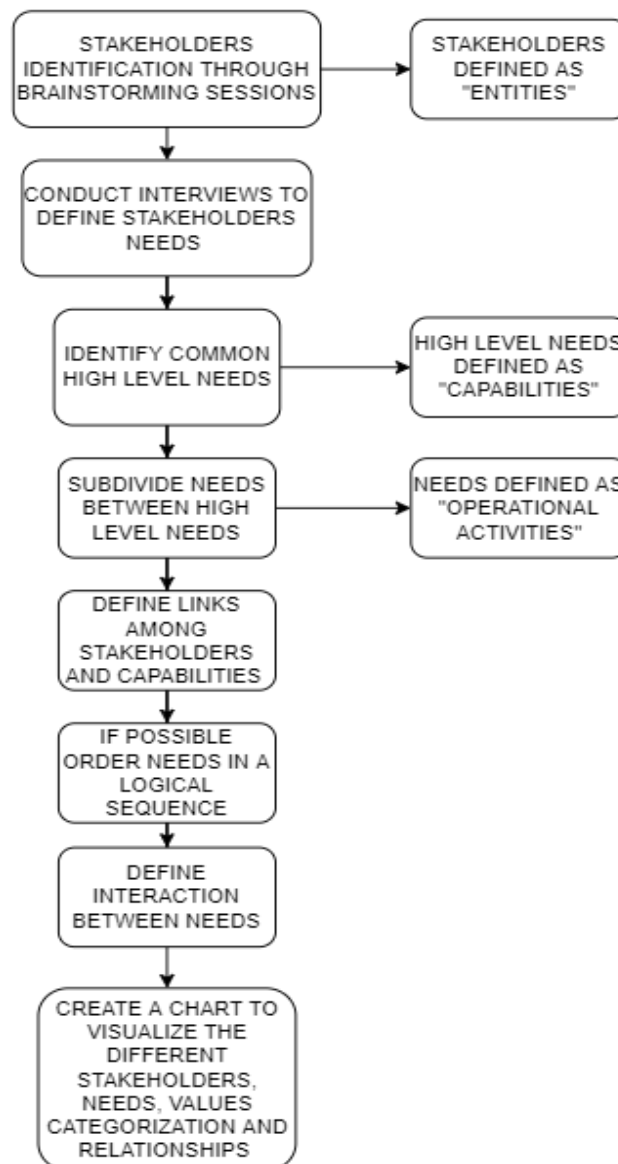


Figure 43: Flow chart general procedure stakeholders and needs analysis

4.2 Advice for MBSE Concept of Operations

The system analysis in the early stages of the project involves the creation of diverse types of graphics. A document-centric approach could lead to referring to a large number of files located in different directories, to be updated individually every time a change is made to the identified concept of operations, mission architecture or functional analysis. Model-based systems engineering, on the other hand, make available to manage them all in one place, as already highlighted, and always staying up to date on any changes, producing model results not obtainable by the graphs alone. The value added by MBSE, beyond that contributed independently by the graphs, is, indeed, primarily created by the relationships and interconnections among the graphs or graphs' elements. This behaviour could retrace the role of the system in SE. Indeed, for the system it is possible to write:

$$\begin{aligned} \text{System value} &\gg \Sigma \text{Part value} \\ \text{System value} &= \Sigma \text{Part value} + \text{relationships value.} \end{aligned}$$

And similarly for the model-based:

$$\begin{aligned} \text{MBSE models} &\gg \Sigma \text{graphs} \\ \text{MBSE models} &= \Sigma \text{graphs} + \text{interconnections.} \end{aligned}$$

To perform a complete analysis, the systems engineer must first identify the main system that will carry out the missions and the actors with whom it will interact. This distinction lets high-level functions be correctly associated with the various involved entities. Using the Capella “System Analysis” section these considerations can be made by creating a CSA chart.

Once the system has been identified, the next step is to define the objectives, “Missions” in Capella, that the satellite must fulfil based on the requests made by the stakeholders. Since each mission has a subject, it would be advisable to build a specific graph to highlight this, the chosen tool permits the creation of a graph (MB diagram) in which arrows indicate the connection between the mission carried out by the system and the actor subject. Within the same graph, the designer can also organize missions in order of importance, taking advantage of the ability to enter comments, identifying, for example, primary and secondary missions.

Similarly to the operational analysis, it is possible to define “Capabilities” for the missions, which in this case can be used to describe the different mission phases, that the satellite must carry out to complete the missions. An MCB chart could help the systems engineer, thanks to model-based logic, to better orientate within the project as it associates capabilities with missions and creates additional graphics for each capability (SDFB) collecting groups of functions useful for their performance. The difference with respect to the previous phase lies precisely in the level of the project referred to, which now concerns greater detail and an approach to carrying out the mission.

The creation of different MCB charts can be useful for the designer to represent different concepts of operations considering for example nominal or off-nominal conditions. Furthermore, the user can enter the initial and final conditions of the phase under analysis.

As previously mentioned, each capability (mission phase) through SDFB graphs can be associated with functions and sub-functions, which can be understood as tasks that the system and the actors must perform within phases in order to be declared completed. The MBSE approach in this case facilitates the understanding and assignment of functions as it allows to distinguish those performed by the system and those performed by the actors. Within the SDFB it is possible to define functional exchanges, which

indicate the condition or an event to switch from one function to another, to arrange them in order of execution, similarly to an FFBD chart, and create functional chains to identify these sets of functions in other types of graphs.

The mission architecture can be created by inserting the actors and the system in a graph, associating each of them with the functions performed, with the appropriate exchanges. A complete view, also including the mission phases to which the functions belong, using the functional chains created for the SDFBs, can be realized thanks to the SAB chart of Capella. This also allows defining interfaces through component or physical connections between actors and the system, that, at this level, can be useful for the engineer to facilitate the definition of requirements, to have a preliminary view with respect to the physical or logical architecture and to determine, according to ECSS-E-ST-10-24C, who will be in control of the interfaces, distinguishing between internal (“under the control of a given actor”) or external (“outside the control of a given actor”) [56].

4.3 Advice for MBSE requirements management

Based on what has been shown in the previous chapter, it is now possible to provide some advice regarding the modelling phase of the requirements that should be applied, as already done for SROC, for similar missions.

In general, to start drawing up a list of requirements, it is necessary to take into account inputs from mission objectives, system analysis, scientific and technological objectives and any constraints imposed by stakeholders.

At first, the identification of the requirements must be carried out in such a way as to remain at a high level of the project, identifying all the “subdivision branches”, and subsequently after several iterations, it is possible to go down a level by finely detailing and expanding them, determining “requirements children”.

To better manage the requirements, it is advisable to use platforms that take advantage of model-based systems engineering, such as Valispace, as it is possible to keep track of changes or updates over time and to be able to exploit clear and well-organized graphics and tabular representations, to establish their uniqueness and unambiguity, to verify them. The traceability of the requirements is one of the peculiarities of the MBSE tools since given the temporal extension of the projects it is very often necessary to make changes following the onset of new requests and create discussions between the different users. Precisely, for this reason, it is necessary over time to know when a requirement was changed and the reasons that led to the modification.

Valispace lets to organize the requirements according to different classifications and standard formats, such as the ECSS one or following the product tree or through a hybrid classification. And it enables to insert “Valis”, already described above, which will help the manager to verify the requirements even before the actual verification.

In the phase of writing it is always necessary to already think about the final product and the verification phase, for this purpose it can be essential to create a link between the requirement and the component, as Valispace allows to do, as this consents to have many more information on how to carry out the verifications.

A fundamental rule, however, is that requirements must be written positively using a complete sentence containing a “shall” statement.

The requirements management platform must also interface with the project modelling tool, in our case Capella.

When classifying and organizing the requirements, the designer should take into account various information: who is in charge of product development and its verification, the component or subsystem to which the requirement refers, the level within a requirements tree, the classification according to the ECSS regulation, the area of competence of the requirement. The latter can be mechanical, thermal, electrical, radiation, AIV, operational, performance, safety. Depending on the mission, the designer can add other categories.

The organization of the requirements can take place in different ways depending on the needs of the engineer. Three organizational proposals produced on Valispace are described below.

The first proposal organizes the requirements by dividing them into “folders” following the ECSS classification, in which some nomenclature changes are made when necessary. Within them, classification takes place using “specifications”, dynamic folders to which details can be added to characterize them, which represent the different subsystems or some parts of the satellite mission architecture. Finally, the requirements are associated with “sections” that are created inside the various “specifications” to indicate sub-categories of common topics. The requirements can then be linked, thanks to the Valispace functionalities, to the individual components. This proposal can be considered as a hybrid between classification by ECSS and product tree (Figure 44).

The second proposal involves the creation of a structure in “folders” that include two areas, following the ECSS-E-ST-10 24 C, for internal requirements, to identify requests of competence, and for external requirements, to consider interfaces of external competence with respect to who will develop the other areas of interest, so that they can be easily transmitted to the entities involved. This has been designed in order to visualize subdivisions that occurred at a contractual level and to apply a point of view oriented towards who will have to carry out the verifications. For both folders, “specifications” which, in this case, represent elements of the mission architecture or the product tree (e.g., spacecraft, mission, propulsion, etc.), and subsequently “sections”, to identify topics useful to classify the requirements (i.e., thermal, radiation, proximity operations, mechanical), can be created. The “types” have been used to indicate the classification of the requirement according to the ECSS regulation. Whenever possible, the requirements can be subsequently linked, as mentioned above, to the individual components. The second proposal, however, causes the designer to lose the information relating to the level of the mission architecture to which reference is being made (Figure 44).

The third classification proposes a distribution of the requirements that slavishly follows the mission architecture. In this case, the “specifications” can be used to recreate the first level branches of the architecture and, within them, the lower branches of the architecture can be created through the “sections” and “subsections”. However, this classification presents problems concerning the location of the interface requirements and the possibility of being able to easily share them with other entities and redundancy in the creation of the “sections” linked to the components, as the assignment of the requirement to the component can be carried out through the functionalities already present in Valispace. The type in this proposal can be used to indicate the kind of requirement (e.g., mechanical, thermal, operational, performance) in order to present a

totally product-oriented vision. Furthermore, this classification does not allow to have a visualization of the requisites according to ECSS (Figure 44).

The requirements can be identified by a code: XXX-YYY-ZZZ-NNN.

For the first two proposals, the three letters XXX represent an abbreviation of the project name, YYY indicates the name of the “folder”, ZZZ the “specification” name and NNN a number to put them in order. For the third, XXX represent the name of the project, YYY the “specification”, ZZZ an abbreviation for the section name and NNN the number of the requirement.

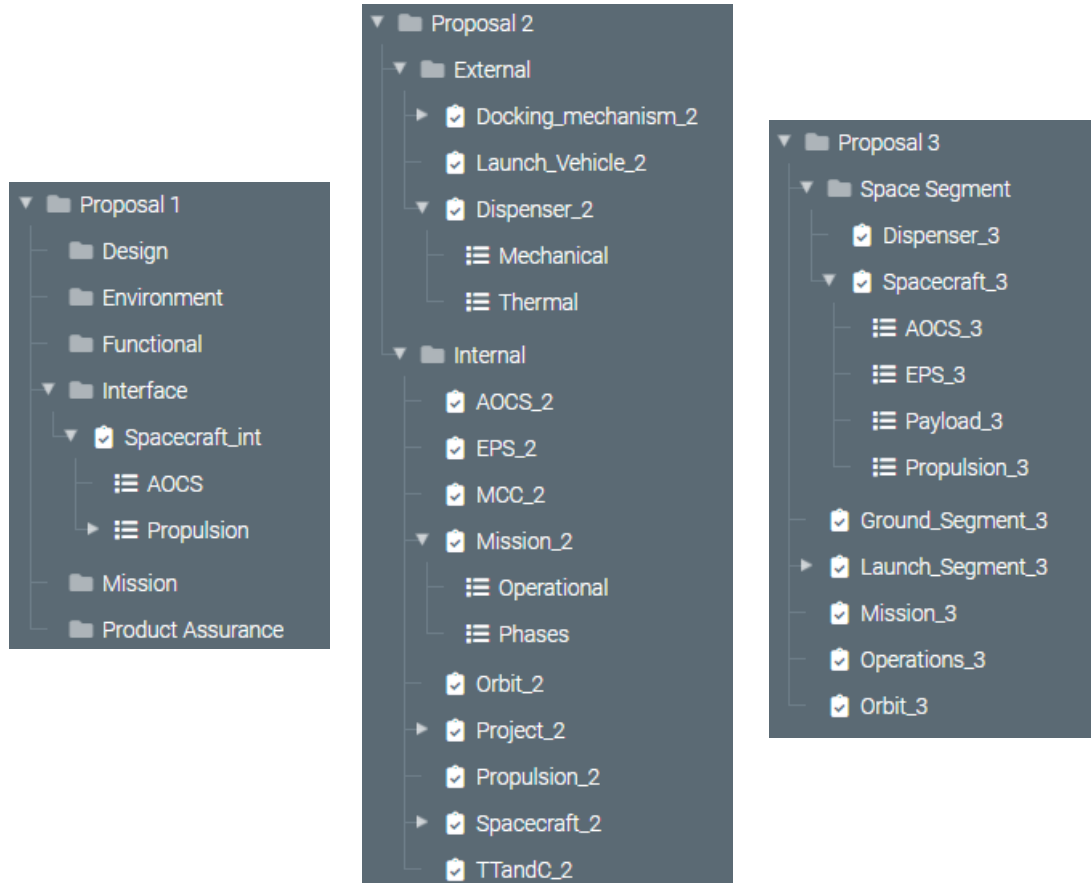


Figure 44: From the left Valispace organization of proposal 1, proposal 2 and proposal 3

4.4 Advice for MBSE logical analysis

What can be abstracted from the construction of the logical architecture for the SROC mission concerns the need for designers to remain free from physical products, so it is necessary to stay on the level of ideas.

In general, the construction of the architecture by the system engineer can take place by creating, using the chosen tool, only a single graph in which all the different components and interactions can be represented. This is a benefit of the MBSE that dynamically represents what happens to the system, maintaining a link with previous analyses.

The graph in question is the LAB, where the logical areas that allow to carry out the mission can be placed, usually the payload and the support platform, which includes areas related to data and command handling, communication, thrust generation,

determination and control of position and attitude, structural support, generation and supply of energy, maintenance of a thermal environment. Within these, to be able to identify the physical components in the next design phase, it is advisable to insert the logic functions and sub-functions performed, drawing whenever possible on those obtained in the functional analysis.

Through these organizational pieces of advice, it will be easy for the designer to identify the exchanges between functions and components that will make available defining the interfaces. Mainly the logical interfaces between the components that can be identified are of three types: data, energy, mechanical.

4.5 Advice for MBSE physical analysis

The physical architecture must be defined after the logical one. The use of Capella allows these analyses to be created in the same environment, ensuring that the designer is always updated on any changes affecting the functions that the components must perform. The architecture can be defined using a PAB diagram reproducing a container structure, in which it is important to distinguish all the different components and parts that make up the subsystems, connecting them via physical links (i.e., cables).

In parallel to the physical architecture built on Capella, it is advisable to implement one on a tool in which the components can be characterized in a simple way.

In this case, from a ModelSat perspective, it was decided to create a model on Valispace that considered not only the physical architecture but the entire mission architecture, including budgets, which can be reused by engineers for the future CubeSat projects through small modifications and adaptations to their missions.

The mission architecture was built considering five areas:

- Ground segment: to detail and describe the characteristics of the stations and their communication devices.
- Launch segment: to consider the vector capable of bringing the satellite into orbit.
- Operations: an area dedicated to the calculation of the link budget and to the collection of constants or parameters.
- Orbit: section on which to import the characteristics of the orbits performed by the satellite.
- Space segment: to characterize the payload and the satellite support platform by building a product tree.

In addition to the link budget, power, energy, mass and volume budgets have been implemented. The first three will be explained in detail in the following paragraphs, while the last two have been easily created on Valispace using the “SOC” function previously explained.

In general, in the initial phases of the project, it is necessary to make tradeoffs to choose the best component, considering the needs of the designer, that best suits the mission. For this reason, for the ground segment, the launch segment and the components of the subsystems of the space segment, alternatives were considered to be chosen, through the “alternative container” function, creating a database of components available on the market.

Taking into consideration the space segment, the following have been implemented hierarchically:

- the payload with different types of solutions;
- the AODCS subsystem containing actuators, sensors and thrusters, in which control moment gyroscopes, magnetic torques, momentum wheels, reaction wheels were considered for the actuators, earth sensor, gyroscope, IMU, star tracker and Sun sensors were included for the sensors, finally, the thrusters have been divided into electrics and cold gasses;
- the onboard computer for command and data handling;
- the communication system contains two-way radios of different bands;
- the subsystem of the electric power divided into batteries and solar panels;
- the subsystem dedicated to the structure;
- the thermal control subsystem.

The user for his project can then choose whether to keep the inserted components and select the desired alternative from the present ones or delete the component he will not use. The possibility of choosing among the alternatives allows, once the selection has been made, to automatically update all the calculations to which the characteristics of the component are linked. In fact, through the functions of Valispace, it is possible to connect to the calculations not only the values of the single component but also those of the element selected at a given moment among the proposals present.

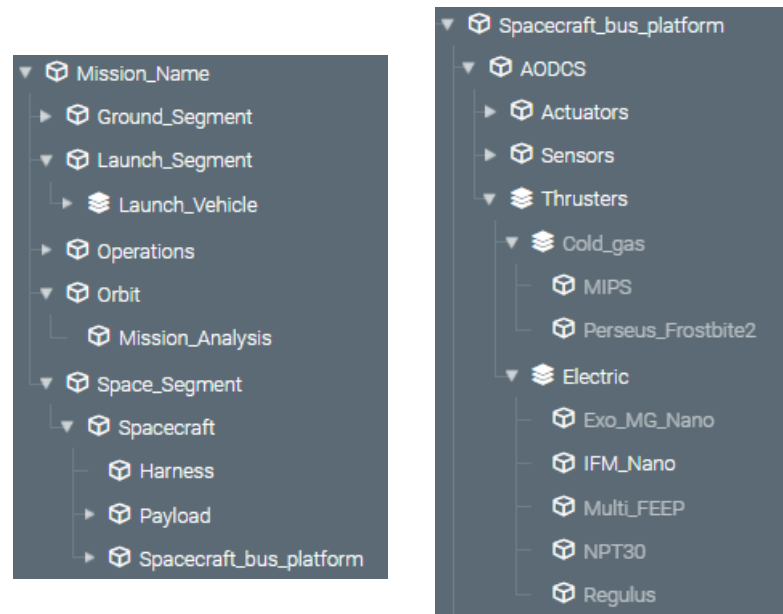


Figure 45: Valispace mission architecture (left) and thruster alternative container (right)

This lets to use a biunivocal approach, such as for thrusters, for which the designer can implement a top-down approach by first choosing the type of propulsion (between cold gas and electric) and then the product on the market through comparisons or a bottom-up approach by selecting initially the component that best suits its needs, between the two classes of engines, and then making a trade-off among the best of the two groups.

Thanks to its model-based approach, Valispace has also been granted the ability to create document folders, reproducing the product tree, to store the datasheets relating to the various components or systems implemented.


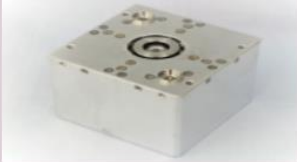

	Exo_MG_Nano	IFM_Nano	Multi_FEFP
			
Nominal_Power	60.00 W	40.00 W	20.00 W
DeltaV	208.33 m/s	208.33 m/s	541.67 m/s
CubeSat_Unit	2.00 U	1.00 U	0.50 U
Mass	1.50 kg	1.50 kg	0.50 kg
Max_Thrust	1.80 mN	0.35 mN	0.14 mN
Performance_Density	2500.00 N*s/U	5000.00 N*s/U	26000.00 N*s/U
Total_Impulse	5000.00 N*s	5000.00 N*s	13000.00 N*s
Volume	2.00e+6 mm ³	1.00e+6 mm ³	182250.00 mm ³
Height	100.00 mm	100.00 mm	45.00 mm
Length	200.00 mm	100.00 mm	90.00 mm
Width	100.00 mm	100.00 mm	45.00 mm
Power_Consumption_nominal	0.00 W	0.00 W	0.00 W
Power_Consumption	$\begin{bmatrix} 0.00 \text{ W} \\ 0.00 \text{ W} \\ 0.00 \text{ W} \end{bmatrix}$	$\begin{bmatrix} 0.00 \text{ W} \\ 0.00 \text{ W} \\ 0.00 \text{ W} \end{bmatrix}$	$\begin{bmatrix} 0.00 \text{ W} \\ 0.00 \text{ W} \\ 0.00 \text{ W} \end{bmatrix}$
Technology	Hall effect	Indium FEFP	Gallium FEFP

Figure 46: Electric cold gas thrusters alternatives

	Cold_gas	Electric
CubeSat_Unit	0.21 U	1.00 U
DeltaV	10.42 m/s	208.33 m/s
Length	30.00 mm	0.00 mm
Mass	0.54 kg	1.50 kg
Nominal_Power	8.50 W	40.00 W
Performance_Density	1205.96 W	5000.00 N*s/U
Height	100.00 mm	0.00 mm
Max_Thrust	10.00 mN	0.35 mN
Total_Impulse	250.00 N*s	5000.00 N*s
Volume	300000.00 mm^3	0.00 mm^3
Width	100.00 mm	0.00 mm
Power_Consumption_nominal	8.00 W	0.00 W
Power_Consumption	$\begin{bmatrix} 10.00 \text{ W} \\ 8.00 \text{ W} \\ 0.25 \text{ W} \end{bmatrix}$	$\begin{bmatrix} 0.00 \text{ W} \\ 0.00 \text{ W} \\ 0.00 \text{ W} \end{bmatrix}$
Thrust_Vectoring	yes	yes
Company	Tyvak/T4i (Italy)	Enpulsion (Austria)
Technology	ColdGas (R134a)	Indium FEFP

Figure 47: Cold gas and Electric thrusters comparison

4.6 Power budget

The Power Budget has been fully implemented on Valispace.

The platform owns the potentiality to create “operating modes”, for each subsystem component, that a user can associate to a certain property whose value varies according to the operating states of the mission. This is the case of “power consumption”.

In the ModelSat, indeed, three different operating modes have been created for the components’ power consumption: a maximum condition, when the consumption is at the peak, an average condition, to consider when the component operates nominally, and a minimum one, to select when the device is off or in a non-operating mode but with an operative integrated thermal control system.

Whenever the user wants to create a power consumption property, he can select the associated mode and a vector will be automatically generated, with the related number

of rows, in this case, three as the operating modes, and the linked different fields names (i.e. max, avg, min).



Figure 48: Mode example for components (left) and mission phases (right)

Similarly, a power consumption vector, corresponding to the one required by subsystems during the various hypothesized mission phases, was created for the satellite as a whole. The implemented phases were: deployment, manoeuvres, observation, safe, Sun pointing, downlink and docking, for which modes have been created and powers calculated.

In this way, the user will obtain the power budget for the different mission phases by choosing the desired components mode (Figure 49). The platform will make a sum of the powers involved for the specific phase and return the requested value.

	Deployment	Manoeuvres	Observation
<u>Momentum_wheel.Momentum_wheel_mode</u>	Max	Max	Max
<u>Reaction_wheel.Reaction_wheel</u>	Max	Max	Max
<u>Earth_Sensor.Earth_sensor</u>	Max	Max	Max
<u>Gyroscope.Gyroscope_mode</u>	Max	Max	Max
<u>IMU.IMU_mode</u>	Max	Max	Max

Figure 49: Mode selection matrix

A preliminary check was also implemented to verify that the power produced by solar panels during orbit was able to meet the demands of the subsystems.

This check took place through the following steps:

- creation of a duty cycle vector, associated with mission phases modes, for which the user must enter their percentage duration;
- creation of the “average required power” vector through a sum-product relationship between the duty cycle vector and the satellite power consumption vector;
- obtaining of the power generated in Sun Pointing P_{EOL} , the sum of those created by the panels arranged on the different faces, whose calculation steps are explained in the paragraph for the determination of the energy budget (4.7);
- creation of the “orbital average power” OAP property by multiplying the rule of thumb, parameter estimated from experience, and the P_{EOL} ;

- calculation of the “average generated power” through the equation $P_{EOL} \cdot dutycycle_{Sun\ pointing} + OAP (1 - dutycycle_{Sun\ pointing})$;
- comparison between the “average generated power” and the power consumption in the different phases.

Based on the results, the user can evaluate the need to vary some EPS parameters.

The designer can implement the duty cycles vector in several ways. He can create a single vector, indicating in the description the reference orbit, and update the values and description when the orbit under study changes, create as many duty cycle vectors as there are phases to be analysed (e.g., duty cycle downlink, duty cycle observation), create a matrix in which different duty cycles are implemented for each column (useful when the mission phases are definitively specified, given the impossibility of changing the dimensions of the matrices in Valispace) or create a fictitious component containing alternatives to be select for duty cycles.

4.7 Energy budget

This paragraph describes the process for evaluating the spacecraft energy budget, which was partly used for the SROC mission. As seen previously, it is implemented using Valispace to import and export the data, Matlab to perform the calculations and graphs, and STK to import the Sun Vector matrices containing the cosines of the Sun angles incidences for the three Cartesian axes.

The creation of the Matlab code took place through separation into sections, where various operations to be executed were realized. The user will have to launch them, from time to time, to arrive at the final result. The equations used are taken from energy budgets made by Tyvak International and from the Wertz manual. The basic steps of the code (complete version in the annexes) and the user's tasks will be explained below.

The first actions that the user must carry out consist in importing one “Sun Vector” matrix on Matlab containing all different mission phases or different “Sun Vector” matrices for the different phases, consisting of three columns, one for each axis of the reference system (x, y, z), and a number of lines equal to the diverse instants of time for which the values are estimated, and one or more matrices containing the dates of the measurements in the form of “day, month, year, hours, minutes, seconds” (Figure 50). In the event that the user imports a single matrix for the Sun Vector, it will be separated for the various mission phases.

```
%% IMPORT DATA FROM STK

load('Date_sun_vector_time')% Dates matrix
Total_Date_sun_vector_time=Date_sun_vector_time;
load('Sun_vector')% Sun Vector matrix
Total_Sun_vector=Sun_vector;
load('Sun_vector_ff_1')% mission phase zero
load('Sun_vector_ob_1')% mission phase one
load('Sun_vector_ff_2')% mission phase two
```

Figure 50: Import data from STK (Matlab script)

The user must assign a numerical identification code to each of these matrices (Figure 51), necessary, by entering from the “command window”, to select the phase to be analysed. If he enters a wrong code, an error sentence will be returned. These operations were implemented through vector comparisons made with “if” and “elseif” statements (Figure 52).

```
%% Vector selection (launch this section to study separately a phase)
% matrix      code
% ff1         0
% ob1         1
% ff2         2
```

Figure 51: Example codes for vector selection (Matlab script)

```
n_check=input('Write the phase number : ');
if n_check==0
    Sun_vector=Sun_vector_ff_1;
elseif n_check==1
    Sun_vector=Sun_vector_ob_1;
elseif n_check==2
    Sun_vector=Sun_vector_ff_2;
```

Figure 52: Comparisons for vector selection (Matlab script)

Once these steps have been carried out, if a single matrix, containing the dates, is initially entered, the code lets to create from it, through comparisons and “while” loops, a matrix containing only the dates of the case study. It changes its format for programming purposes and produces graphs from which to observe the change over time of the Sun Vector for the three axes.

The script also offers the possibility, through an implementation similar to the previous ones, to reduce the length of the matrices under study (time and Sun Vector), by entering from the command window the deadline or the number of rows of the matrix for which the user wants to perform the analyses. It also allows performing calculations by considering all the faces on which solar panels can be arranged (+x, +y, +z, -x, -y, -z).

The connection to Valispace takes place in a special section, where the command for the connection, “ValispaceInit(“https://demo.valispace.com”, “username”, “password””, is found (Figure 53). To create the link, the user must first launch the appropriate add-on from the “Matlab app” tab and then “Run” it.

```
%% VALISPACE CONNECTION
%Open Valispace addon in APPS and then run this section

ValispaceInit("https://demo.valispace.com", "usurname", "password")
```

Figure 53: Valispace connection

In this way, the data can be imported using the commands “ValispaceGetValue(“”, value name)”. In order to do the energy budget, the script imports:

- for solar panels the area of the solar cells A_{sc} in m^2 , the number of solar cells for the different faces N_{sc_i} , in which the index “i” is inserted to consider the

- i-th face, the Solar Irradiance P_{Sun} in W/m^2 , the efficiency of the solar cells at the beginning of life η , the efficiency at Maximum Power Point Tracker (MPPT) I_d , the degradation per year;
- for batteries the time step data in *seconds*, the initial battery capacity IBC in Wh , the initial battery energy IBE in J , the battery charge efficiency BCE , the battery discharge efficiency BDE , the minimum imposed value that the state of charge of the battery must have SOC_{limit} as a percentage;
- for the satellite the “Power Consumption” in W and the satellite life in *years*.

The code using these values determines:

- the area of each solar array for the different faces $A_{sa_i} = A_{sc} N_{sc}$;
- the power generated by the solar arrays for each face at the beginning of life $P_{sc_i} = A_{sc} N_{sc_i} P_{Sun} \eta I_d$;
- the life degradation $L_d = (1 - degradation/year)^{satellite\ life}$;
- the power generated at the end of life by the solar arrays for each face $P_{EOL_i} = P_{sc_i} L_d$.

These variables can also be exported directly from Valispace but it has been chosen to carry out the operations on the platform to give the user a clear and complete view of the various values and mathematical relations.

Thanks to these parameters, in a specific section, it is possible to identify the attitude of the vehicle, in “Sun Pointing”, for which the greatest overall input power is obtained, or to load the Sun Vector matrix for the “Sun Pointing” phase. In detail for the evaluation, two satellite faces, selected from time to time and perpendicular to each other, are made to rotate around the inertia axis that does not intersect them, starting from the condition in which a face is perpendicular to the sun's rays ($\theta = 0^\circ$ and cosine loss $\cos\theta = 1$) and the other parallel ($\theta = 90^\circ$ and $\cos\theta = 0$), until gradually reaching the opposite condition (Figure 54). In this way, for each angle of rotation, the sum of the powers absorbed by the single faces under study is evaluated

$$P_{SUNIN_{sp_i}} = P_{EOL_i} \cos\theta.$$

to identify the maximum total input power and the associated attitude (Figure 55). By evaluating this value, the user will be able to verify whether the satellite can produce the necessary power, after a comparison with imported power consumption, to carry out the operations required in that given set-up and therefore decide to maintain or modify the panels' disposition, the number of solar cells or their characteristics.

```
Cosine_loss_matrix(i,2)=t(i);
Cosine_loss_matrix(i,4)=1-t(i);
P_Sun_IN_sp_matrix(i,1)=0;
P_Sun_IN_sp_matrix(i,2)=P_EOL_y*Cosine_loss_matrix(i,2);
```

Figure 54: Cosine loss variation and Sunpointing input power computation (Matlab script)

```

P_SUN_IN_sp_matrix_tot(i,1)= P_Sun_IN_sp_matrix(i,1)...
    +P_Sun_IN_sp_matrix(i,2)+P_Sun_IN_sp_matrix(i,4);
end

[P_SUN_IN_sp_max,row_index_max]=max(P_SUN_IN_sp_matrix_tot);

```

Figure 55: Maximum Sunpointing input power computation (Matlab script)

The code focuses also on “Nadir Pointing” layouts. To the latter and the Sun Pointing setup, the same equations are implemented. For each face equipped with panels, a vector of the power produced is created, with “N” number of rows (Figure 56).

```

% x y z -x -y -z creation of P SUN IN for nadir pointing
P_Sun_IN=zeros(size(Sun_vector_rid,1),6);
for j=1:N_rows
%Nadir pointing
%+x
if Sun_vector_rid(j,1)>0
    P_Sun_IN(j,1)=P_EOL_x*Sun_vector_rid(j,1);
else
    P_Sun_IN(j,1)=0;
end

```

Figure 56: Nadir pointing input power computation (Matlab script)

The vectors of the faces with a positive sign (+x, +y, +z) take on values other than zero only when the values of the column of the Sun Vector concerned are greater than zero, while the vectors with a negative sign (−x, −y, −z) when the values of the Sun Vector column are less than zero. For rows with non-zero values, the estimate is made using the relationship:

$$P_{SUNIN_i} = P_{EOL_i} Sun Vector_{xyz}$$

that consists, as done before, in the product between the power produced by the satellite at end of life and the Sun Vector, that is the cosine loss. Furthermore, comparison graphs are generated between the power produced in nadir pointing and in Sun pointing, to provide the possibility of analysing the trend and values over time, and the *OAP* is calculated by means of an average of the powers produced along the orbit considering differing incidences. The latter represents a detailed version of the *OAP* determined as $P_{EOL} \cdot rule\ of\ thumb$, used to estimate the power produced by the satellite during orbit and make first verifications, in which the rule of thumb consists of a constant parameter based on previous experiences.

Within the same section, the calculations of parameters vectors useful for the realization of the energy budget are carried out. For the faces under analysis, they are:

- the energy produced for the different times in Ws , $Energy_{in_i}(N) = P_{Sun_{in_i}}(N) \text{ time step}$;
- the total accumulated energy, estimated considering the energy value at the given instant plus the sum of the previous values, $EnergySum_{x_i}(N) = Energy_{in_i}(N) + EnergySum_{x_i}(N - 1)$;

- the total energy in *Wh* determined by the relationship $Energy_{Wh_i}(N) = \frac{Energy_{Sum_{x_i}(N)}{3600}$;

For the satellite:

- the subtraction between the summation of the power generated by the faces and the power consumption of the spacecraft;
- the variation of the average energy obtained multiplying the previous result for the time step, $\Delta Energy_{tot}(N) = (\Sigma P_{Sun_{in_i}}(N) + -Power_{Consumption}) \text{ time step}$;
- the initial energy supplied by the batteries in *Ws*, calculated for the first row as $E_{tot} = (\Delta Energy_{tot} + IBE)$ when $(\Delta Energy_{tot} + IBE) < IBE$ or $E_{tot} = IBE$ when $(\Delta Energy_{tot} + IBE) \geq 0$ and for the subsequent rows as $E_{tot}(N) = \Delta E_{tot}(N) + E_{tot}(N - 1)$ when $\Delta E_{tot}(N) + E_{tot}(N - 1) < IBE$ or $E_{tot}(N) = IBE$ if $\Delta E_{tot}(N) + E_{tot}(N - 1) \geq IBE$;
- the batteries state of charge in *Wh* as $SOC_{tot} = E_{tot}/3600$;
- the batteries state of charge percentage $SOC_{perc_{tot}} = 100 SOC_{tot}/IBC$;
- an average value across the state of charge vector;
- the depth of discharge $DOD_{tot} = 100 (1 - SOC_{tot}/IBC)$;
- the maximum value of the depth of discharge vector.

Thanks to these results it is possible to generate graphs (Figure 57) that allow the user to see the trend of the state of charge as time changes and to identify, also via a message from the command window, the moment in which the state of charge falls below the limit imposed. The user can thus choose whether to repeat the calculations by modifying some parameters or whether to export the data.

```
plot(time_pos_v_tot,SOC_perc_tot_pos_v)
xlabel('time')
ylabel('Total SOC percentage')
title('Total SOC till 75%')
hold on
```

Figure 57: Total state of charge plot (Matlab script)

The script exports the desired data to Valispace using the “ValispacePull ()” commands to select the parameters and “ValispacePushValue (“”, value name)” to make the transfer (Figure 58).

```
%% EXPORT DATA IN VALISPACE

ValispacePull()
ValispacePushValue("",MAX_DOD_tot)
ValispacePushValue("",SOC_perc_tot_mean)
ValispacePushValue("",OAP)
```

Figure 58: Valispace parameters export (Matlab script)

Key data exported are the maximum depth of discharge vector value, the average state of charge and the OAP.

Figure 59 shows a flowchart that summarizes the process.

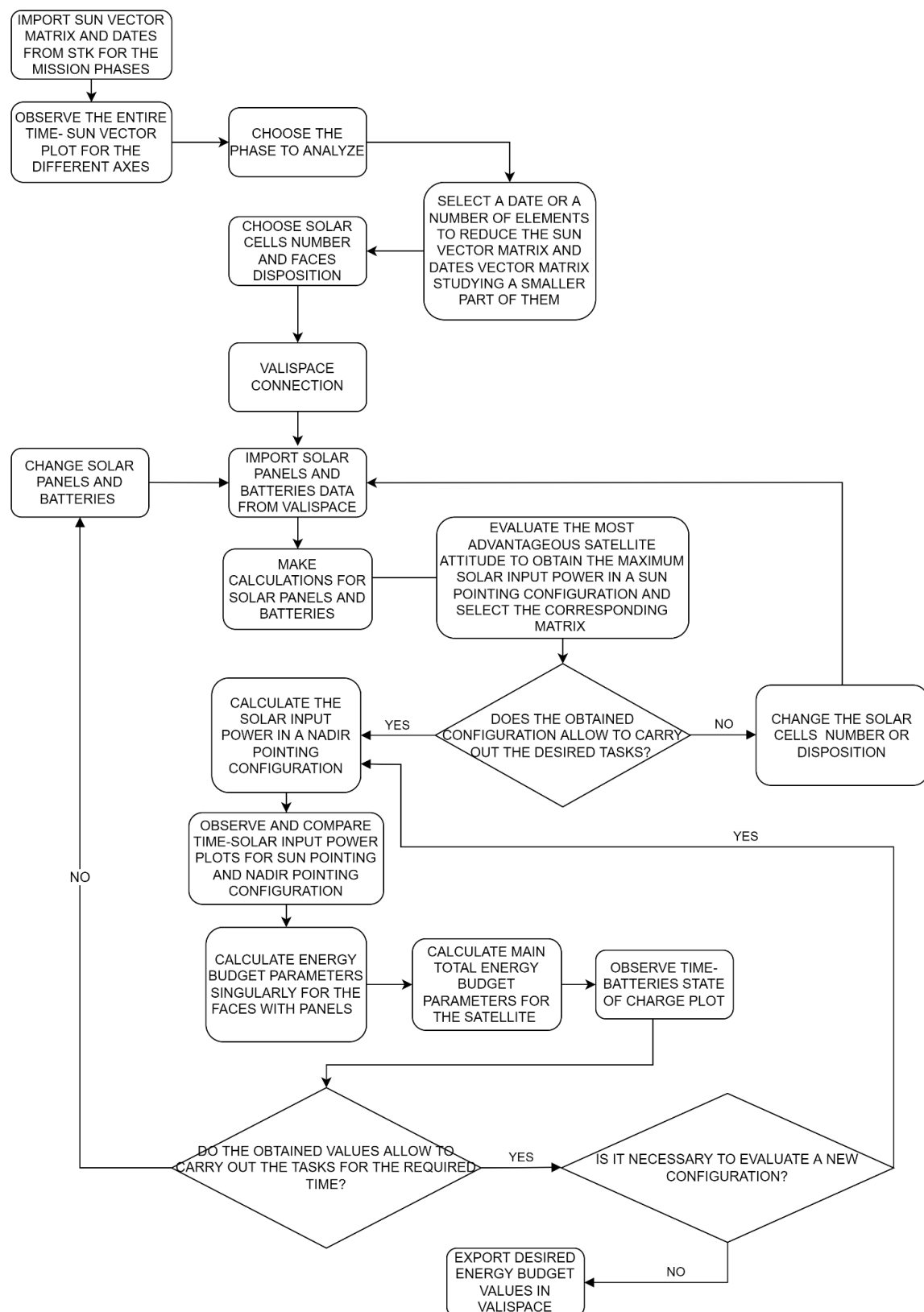


Figure 59: Flow chart for energy budget determination

4.8 Link budget

The link budget was implemented in the ModelSat using exclusively the “components” section of Valispace, offering to the end-user the possibility of inserting one or more antennas for communication both on the ground and on the spacecraft. In particular, the following elements of the mission architecture were involved: Ground Segment, Operations and Space Segment.

The section relating to the ground station was created considering two macro-groups separately: the antennas (including receivers) and the transmitters. For each of these, in order to speed up the link budget calculations, various ground stations with different types of antennas and transmitters and their corresponding properties have been inserted hierarchically, using the “alternative container” function. This function allows to compare the properties of the different solutions and to choose the favourites that will be used for the calculations. It was implemented, for this section, first to select the station and then to select the antenna (e.g. UHF-band antenna, S-band antenna). The user could subsequently insert alternatives for the single antenna considering, for example, different models or manufacturers.

The space segment data have been placed in the spacecraft Comsys section where different types of antennas can be selected for bandwidth and characteristics.



Figure 60: Example of Valispace architecture for the ground segment (left), space segment (centre), operations (right)

The operations compartment, on the other hand, was built considering an area dedicated to collect values, for example, losses, and constants to use in the various calculations, and an area to execute the link budget determination, both for the downlink and the uplink case.

The separation between transmitter and receiver in the ground segment has been carried out as the user can choose whether to select the uplink or downlink mode. In this way, he will have to select hierarchically, among the various alternatives, the ground station to which that receiver or transmitter belongs, the antenna band on which to operate and then the component among the various offers with the preferred characteristics.

Similarly, he will have to act for the spacecraft in which transceivers of different bands have been implemented, so he will have to choose the component with the desired characteristics among the different alternatives for the transceiver band.

By varying the selected elements and implementing the disturbances related to the links, the user can obtain different link budgets for the transmission bands.

Mixed equations and nomenclatures were chosen to execute the budgets, taking into consideration the slides of Professor Sabrina Corpino's "Sistemi aerospaziali" lectures and the link budgets made for the SROC satellite by Tyvak International in phase O/A of the project.

As described in the previous chapter calculations were performed using the E_b/N_0 method, considering the same equations for the downlink and the uplink. In the equations, the subscript "t" will be used to indicate the transmitter and the subscript "r" for the receiver.

As for the UHF band in phase O/A, the following data have been used.

For the transmitter: the bandwidth in kHz , the gain G_t in dB , the transmission power P_t in W , the signal frequency f in MHz , the data rate in bps , the losses along the transmission line L_t in dB , the noise bandwidth estimated as $10 \log_{10} Bandwidth_t$ and the $EIRP_t$ equal to $10 \log_{10} P_t - L_t + G_t$.

For the receiver: the reference temperature T_0 in K , the actual temperature T_{ra} in K , the gain G_r in dB , the line loss L_r in dB , the implementation losses in dB , the noise figure F in dB , the required ratio E_b/N_0 .

For the link the atmospheric losses L_a , the polarization losses L_p , the losses due to misalignment L_m , the rain losses L_{rain} , all in dB , the elevation angle δ in rad , the radius of the earth R_{Earth} in km , the average distance of the satellite from the ground station during communications h in km , the Boltzmann constant $k = -228.6 \text{ dBW/HzK}$, the link distance or slant range equal to $R_{Earth} \left(\sqrt{\left(\frac{(R_{Earth}+h)^2}{R_{Earth}^2} \right) - (\cos\delta)^2} - \sin\delta \right)$.

The calculation was carried out starting from the determination of the total losses by means of the relationship:

$$L_{TLS} = \text{Free Space Path Loss} + L_a + L_p + L_m + L_{rain},$$

whit the Free Space Path Loss in dB was:

$$\text{Free Space Path Loss} = 20 \log_{10} \text{Link Distance} + 20 \log_{10} \text{frequency}_t + 32,4.$$

Subsequently, the temperature disturbance in communication called "System noise temperature" was computed:

$$T_s = T_{ra} + T_0 \left(\frac{1-L_r}{L_r} \right) + T_0 \left(\frac{F-1}{L_r} \right).$$

And the ratio E_b/N_0 in dB as the sum of the previously determined or collected values:

$$\frac{E_b}{N_0} = EIRP_t - L_{TLS} + G_r - k - 10 \log_{10} T_s - 10 \log_{10} \text{Data Rate}_t$$

with E_b , energy signal associated with each user data bit, equal to $EIRP_t - L_{TLS} + G_r$ and N_0 , noise power spectral density corresponding to the noise power in a 1 Hz bandwidth, equal to $-k - 10 \log_{10} T_s - 10 \log_{10} Data Rate_t$.

Thanks to this last equation, the link margin is obtained considering also the required E_b/N_0 and the receiver additional losses in order to determine if the link is absent (when the threshold is less than zero), marginal (when between zero and six) or closed (when greater than six):

$$Link\ Margin = \frac{E_b}{N_0} - \left(\frac{E_b}{N_0} \right)_{required} - Implementation\ Loss_r.$$

In addition, intermediate values have been determined that can be useful for evaluating the goodness of communication, these are:

- the isotropic receive level $IRL_{r_a} = EIRP_{t_a} - L_{TLS}$;
- the signal level of the receiver $Rx\ signal\ level = IRL_{r_a} + G_r - L_r$;
- the Receiver antenna $GT = G_r - (10 \log_{10} T_s - L_r)$ in dB with $T_e = 10 \log_{10} T_s + -L_r$;
- the carrier to noise density ratio $\frac{C}{N_0} = EIRP_t - L_{TLS} + G_r - k - 10 \log_{10} T_s + L_r$;
- the Carrier to Noise ratio $\frac{C}{N} = \frac{C}{N_0} - Noise\ bandwidth$.

5 Conclusions

The general objective of the thesis was to develop, through the use of MBSE tools, a procedure, useful for systems engineers, to be applied in the early stages of a small satellites project.

Within the thesis, after introducing the context in which it was carried out, a careful analysis of the literature was made to identify the state of the art of the main methodologies, tools and languages to highlight their characteristics.

One of the main problems still present for these topics is the lack of uniformity in the terminology between the different authors, probably linked to the relatively young age of the model-based world, for which it was necessary to uniquely define the terms: methodology, tool, procedure and environment. Despite this, there are several benefits that it has been possible to draw from the bibliographic searches: the presence in the market of a great variety of software and platforms capable of satisfying the different requests of project teams or companies, the optimization of communication thanks to a better data sharing and a better exchange of information, an improvement in the management of complexity, an higher product quality, the possibility of reusing previously created models reducing time and costs, increasing productivity, the opportunity to manage, define and unambiguously trace the requirements, the ease in learning the different concepts thanks to the use of standardized models.

The set goal was achieved through reverse engineering of ESA's SROC rendezvous and docking mission. For it, the project was modelled in an MBSE environment, choosing the tools based on the needs of the moment: being able to use methodologies and tools with a license already available to the project team or open-source, being able to implement large systems considering multiple levels of design, the need to learn how to use them in the shortest time possible, also taking advantage of user friendly and easy to understand interfaces, the presence of documentation to learn how to use the software. The choice fell on Capella with the Arcadia methodology and Valispace, which have been explored in the course of the thesis.

Capella has been used to define the SROC stakeholders and their needs, the concept of operations by implementing different scenarios, the logical and physical architecture for which different kinds of graphs were created to carry out the analysis in detail. Valispace has been taken into consideration to keep track of the requirements and the documents used, to create a mission architecture and budgets.

In parallel to the modelling that took place for the delta A and B1 phases of the project, the work carried out in the previous phases was also described, which in most cases presented a document-based approach. Finally, through a work of abstraction, a generalization was made for the different levels of design and the computation of budgets, to be able to create a high-level guide, through a series of tips and codes to be modified, to design a CubeSat mission. In this second phase, a database of components has been also created on Valispace, to guide the designer in their selection by offering different solutions on the market, and a list of three proposals for organizing the requirements.

From the parallelism between the work carried out in phase O/A and that in delta A/B1, which consists in the transition from a document-based to a model-based approach, it was possible to observe how this variation does not only constitute a transition from the

use of tables to the use of graphics, able to communicate the various information more effectively, but also to manage them all in one place, therefore a greater mental order, and always staying up to date on any changes, producing model results not obtainable by the graphs alone.

Thanks to this functionality, the project team, even during this period in which work in presence and smart-working were alternated, was able to complete the various pre-set tasks without slowing down due to updating errors.

It was also noted that the real power of MBSE consists in the relationships and interconnections among the graphs or graphs' elements to ensure the conduct of the various analysis by addressing them from different points of view, highlighting the different facets and increasing the designers' visions. All this perfectly reproduces the role of the system and its value which is greater than the single parts that compose it thanks to the different relationships.

$$\begin{aligned} MBSE\ models &>> \Sigma\ graphs \\ MBSE\ models &= \Sigma\ graphs + interconnections \end{aligned}$$

One of the examples that can be made to corroborate the thesis of the different points of view is, for example, that of the organization of the requirements that despite they are ordered following a proposal can be viewed according to different criteria, based on: the folder, the specification, the section, the identification code, the associated components, the type, etc.

The thesis in addition to the modelling of the analysis also considered the calculation of the budgets because they are part of the data that the systems engineer must pay attention to during the design phase, therefore it was decided to create budget models that the future designers can use by making small changes. Confirmation of this is also given by the possibility of the Valispace tool to compute them internally or thanks to the Excel and Matlab add-ons and to be able to store the different results in one place. If in the early stages it is possible to carry out high-level evaluations by moving forward in the project, it is necessary to consider the greatest number of data by increasing the calculation precision.

The work gave the opportunity to understand that to learn how to use a tool and master it, it is necessary to invest a lot of time, it is, therefore, necessary to choose the tool wisely to tackle the project, the less intuitive it will be, the greater will be the learning time. This from a business perspective could lead to an increase in costs related to staff training and an extension of project times.

What can be said in the long term is that the use of MBSE, and in this case the combination of Capella and Valispace cover every aspect of project management and design, facilitates the work and makes it more understandable.

In a future perspective, the work on Capella could be expanded by defining in the subsequent phases of the project what the functional exchanges between components represent, identifying interface values between the parts. The logical and physical architecture will also have to be expanded by considering the subsystems, components and final connections to be used to complete the mission.

Additional graphs can be created to define the operating modes of the components.

On Valispace, on the other hand, the ModelSat database can be expanded by adding components on the market and other proposals for the classification of requirements.

All this could then be further increased by connecting as many instruments as possible, such as STK to have information on the orbits and always enrich any numerical evaluations, providing a 360° view.

Considering the various benefits of MBSE listed, in the future, it should always be used by exploiting its potential and overcoming the document-based approach that until now preceded it within project teams and above all to manage large missions size.

Annex A: Capella diagrams

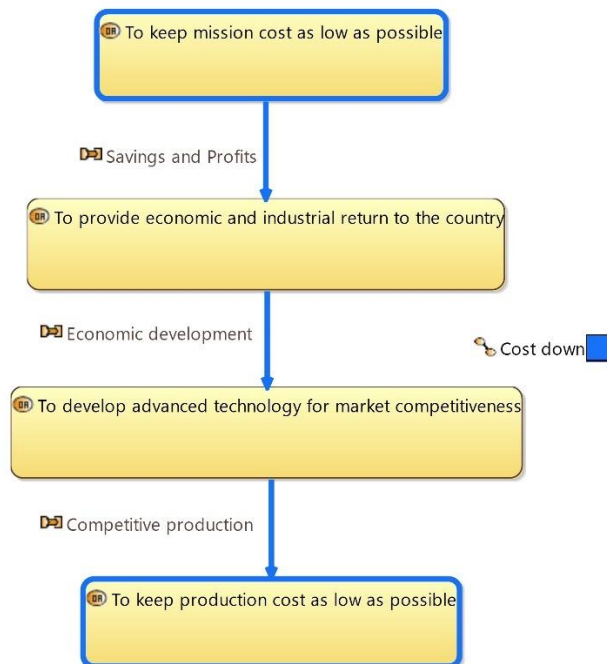


Figure 61: Cost down OAIB

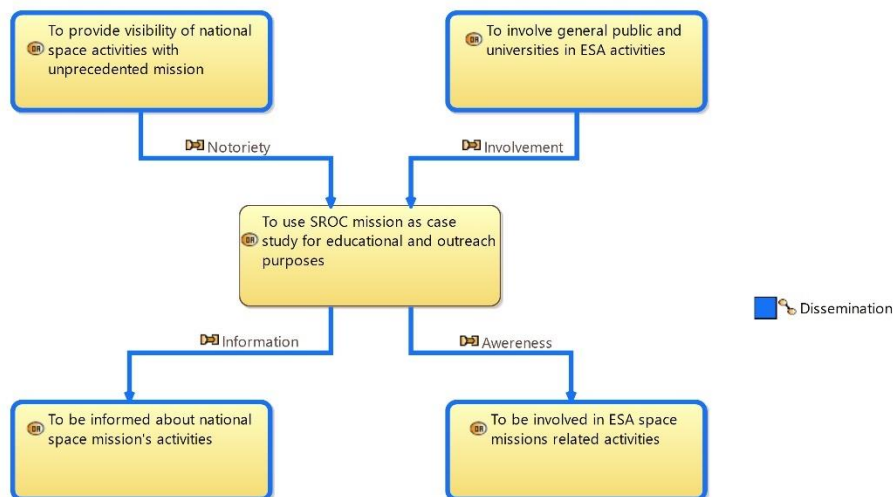


Figure 62: Dissemination OAIB

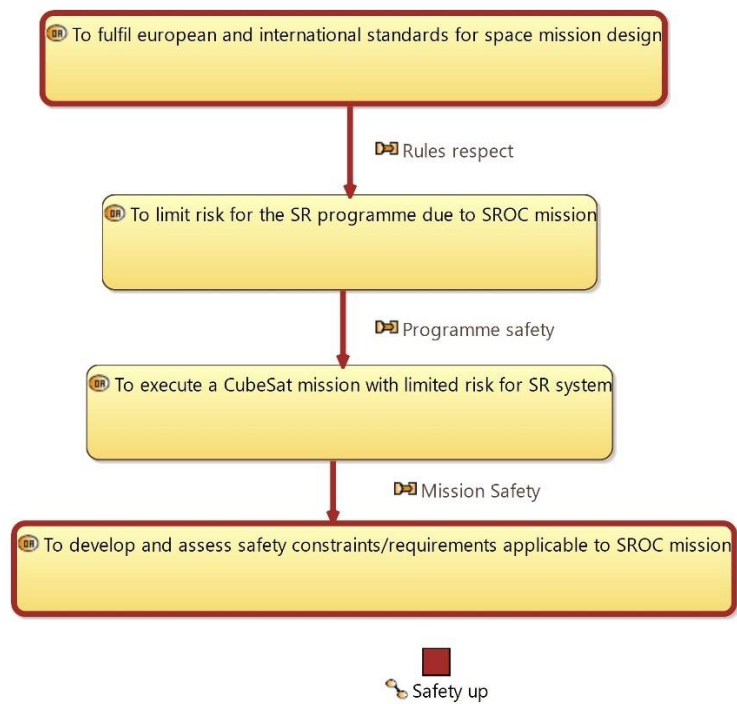


Figure 63: Safety up OAIB

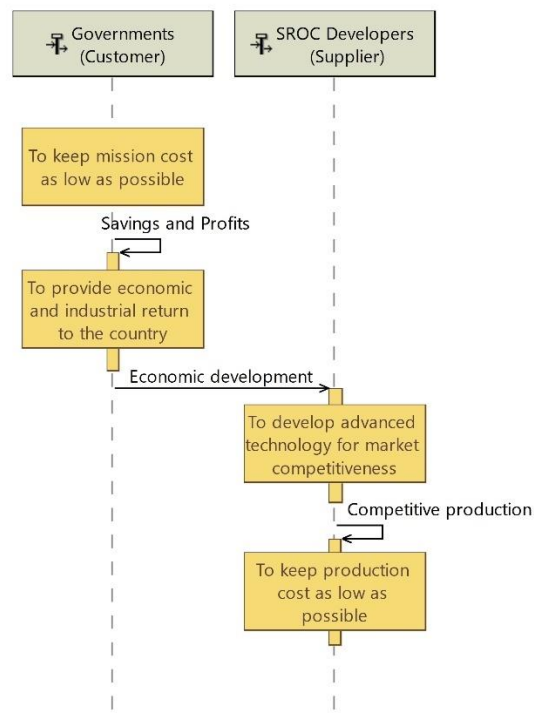


Figure 64: Cost down OES

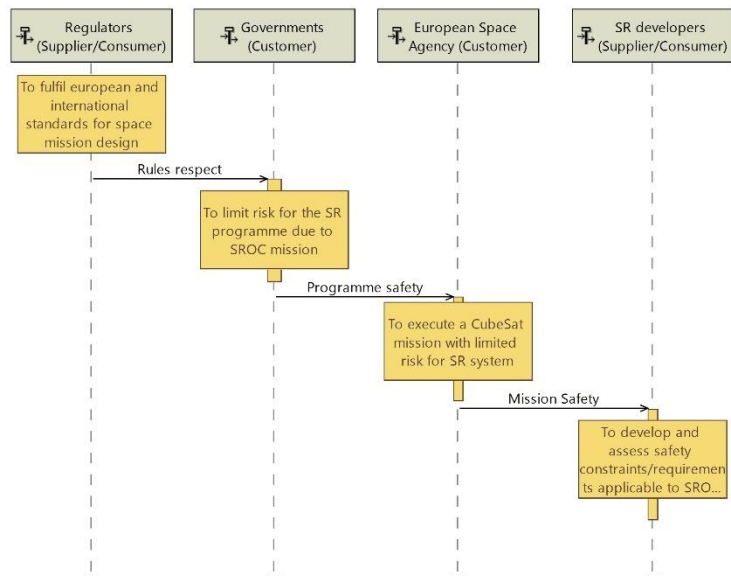


Figure 65: Safety up OES

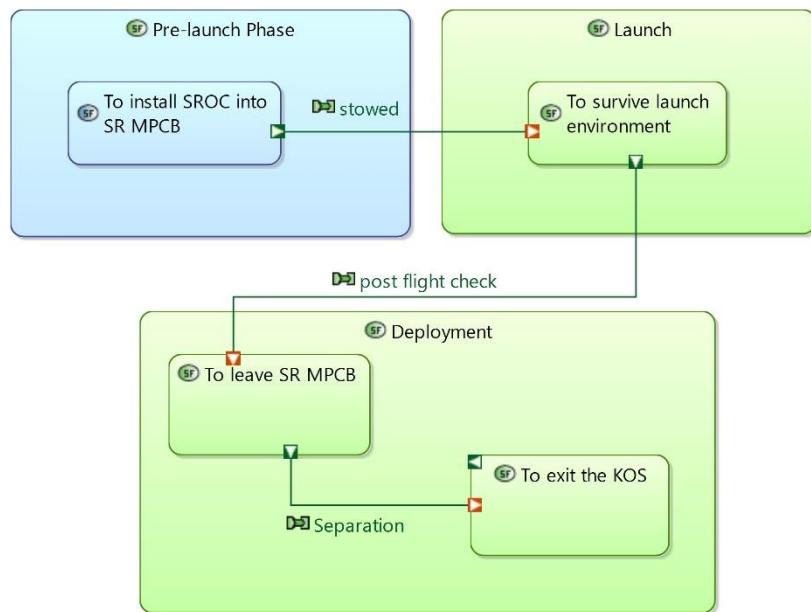


Figure 66: Launch and Early Operations Phase (LEOP) SDFB

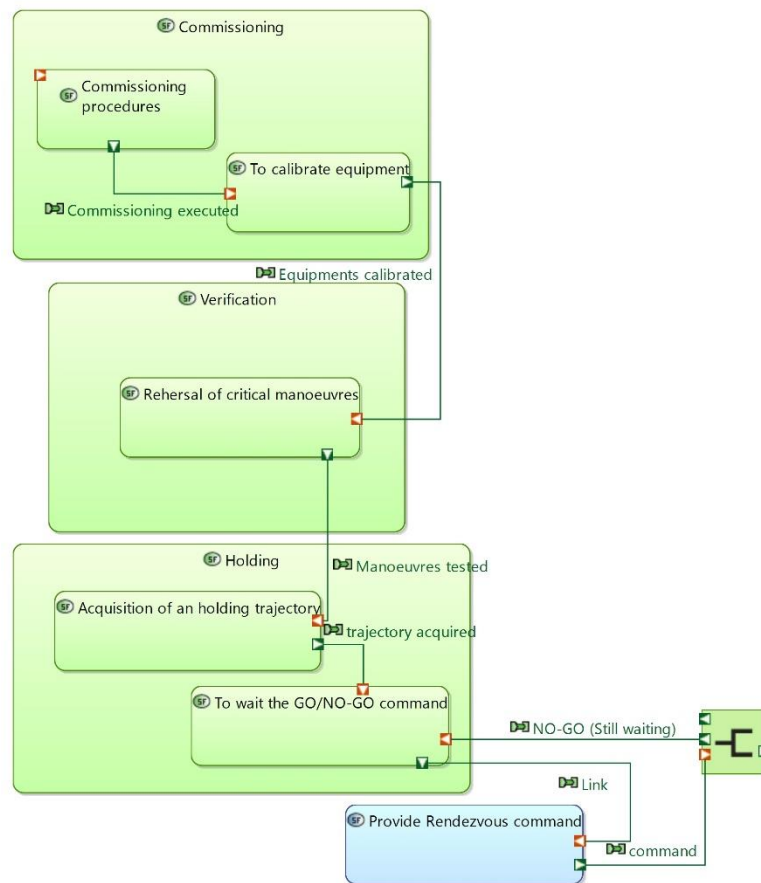


Figure 67: Commissioning and Performance Verification Phase (CPVP) SDFB

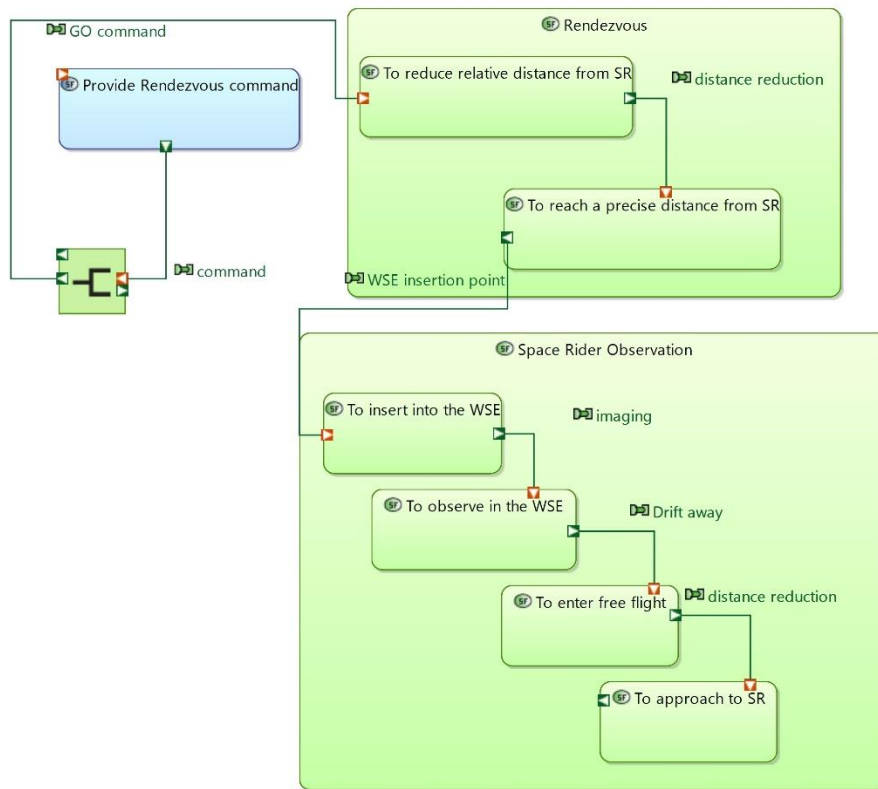


Figure 68: Proximity Operations Phase (POP) SDFB

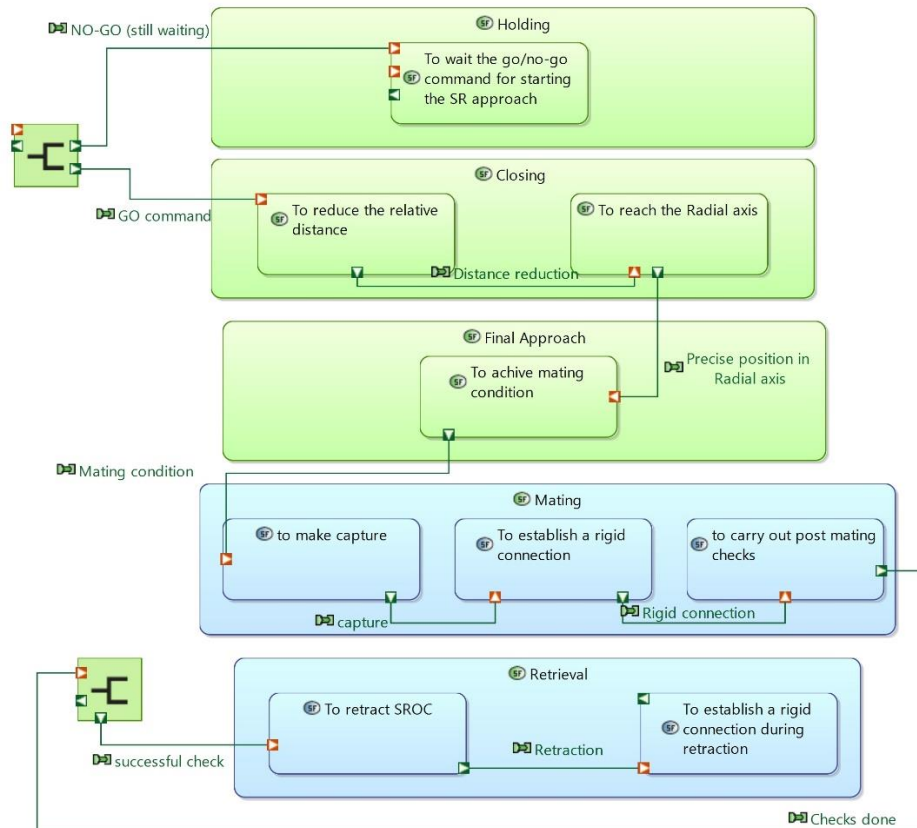


Figure 69: Docking and Retrieval Phase (DRP) SDFB

Annex B: Energy budget script

```
clc
clear all
%% IMPORT DATA FROM STK
load('Date_sun_vector_time')% Dates matrix
Total_Date_sun_vector_time=Date_sun_vector_time;
load('Sun_vector')% Sun Vector matrix
Total_Sun_vector=Sun_vector;
load('Sun_vector_ff_1')% mission phase zero
load('Sun_vector_ob_1')% mission phase one
load('Sun_vector_ff_2')% mission phase two
load('Sun_vector_ob_2')
load('Sun_vector_ff_3')
load('Sun_vector_ob_3')
load('Sun_vector_ff_4')
load('Sun_vector_ob_4')
load('Sun_vector_ff_5')
load('Sun_vector_ob_5')
load('Sun_vector_ff_6')
load('Sun_vector_ob_6')
load('Sun_vector_ff_7')
%% Vector selection (launch this section to study separately a
phase)
% matrix      code
% ff1         0
% ob1         1
% ff2         2
% ob2         3
% ff3         4
% ob3         5
% ff4         6
% ob4         7
% ff5         8
% ob5         9
% ff6        10
% ob6        11
% ff7        12

n_check=input('Write the phase number : ');
if n_check==0
    Sun_vector=Sun_vector_ff_1;
elseif n_check==1
    Sun_vector=Sun_vector_ob_1;
elseif n_check==2
    Sun_vector=Sun_vector_ff_2;
elseif n_check==3
    Sun_vector=Sun_vector_ob_2;
elseif n_check==4
    Sun_vector=Sun_vector_ff_3;
elseif n_check==5
    Sun_vector=Sun_vector_ob_3;
elseif n_check==6
    Sun_vector=Sun_vector_ff_4;
```

```

        elseif n_check==7
            Sun_vector=Sun_vector_ob_4;
        elseif n_check==8
            Sun_vector=Sun_vector_ff_5;
        elseif n_check==9
            Sun_vector=Sun_vector_ob_5;
        elseif n_check==10
            Sun_vector=Sun_vector_ff_6;
        elseif n_check==11
            Sun_vector=Sun_vector_ob_6;
        elseif n_check==12
            Sun_vector=Sun_vector_ff_7;
    else 'error'
    end
    %% Date - subdivision of date vector in subvectors for the
    different phases

    i=1;
    j=1;
    Date_sun_vector_time=[];
    while i<=size(Sun_vector,1) && j<=size(Total_Sun_vector,1)
        if Total_Sun_vector(j,:)==Sun_vector(i,:)

            Date_sun_vector_time(i,:)=Total_Date_sun_vector_time(j,:);
            i=i+1;
        end
        j=j+1;
    end
    %% Format change

    Date_v=Date_sun_vector_time;
    Date_v(:,1)=Date_v(:,3);
    Date_v(:,3)=Date_sun_vector_time(:,1);
    time=[1:size(Date_v,1)]; % vector of seconds based on dates
    %% x y z plot all phases

    figure(1)
    plot(time,Sun_vector(:,1),time,Sun_vector(:,2))
    xlabel('time [s]')
    ylabel('Sun vector')
    legend('x','y')
    %% DATE SELECTION

    yyyy=input('write the year yyyy of the data for which you want
to end the rows of the coloums (else write 0): ');
    mm=input('write the month mm of the data for which you want to
end the rows of the coloums (else write 0): ');
    dd=input('write the day dd of the data for which you want to end
the rows of the coloums (else write 0): ');
    HH=input('write the hour HH of the data for which you want to
end the rows of the coloums (else write 0): ');
    MM=input('write the minutes MM of the data for which you want to
end the rows of the coloums (else write 0): ');

```

```

SS=input('write the seconds SS of the data for which you want to
end the rows of the coloums (else write 0): ');
Stop_date=[yyyy,mm,dd,HH,MM,SS];
if Stop_date==[0,0,0,0,0,0]
    N_rows=input('write the numbers of rows you want consider:
');
else
    N_rows=0;
end
i=1;
while N_rows ==0
    if Date_v(i,:)== Stop_date
        N_rows=i;
    else i=i+1;
    end
end
% date vector reduction
for i=1:N_rows
    Date_v_rid(i,:)=Date_v(i,:);
end
%Sun vector nadir pointing reduction
Sun_vector_rid=[];
for i=1:N_rows
    Sun_vector_rid(i,:)=Sun_vector(i,:);
end
time_rid=[1:N_rows];
%% VALISPACE CONNECTION
%Open Valispace addon in APPS and then run this section
ValispaceInit("https://demo.valispace.com","username","password")
%% =====> VALISPACE ADDON
<=====
% Thank you for using Valispace: the best place for your
engineering data.
%
% Example Usage:
%
% 1)
ValispaceInit("https://demo.valispace.com","username","password"
) % Valispace Login
% 2) ValispacePull()
% optional: pull all Valis for faster access or access via name
% 3a) ValispaceGetVali("MySat.Mass")
% get Vali as a struct
% 3b) ValispaceGetValue("MySat.Mass")
% get Value
% 4) ValispacePushValue("MySat.Mass",value)
% push Value to Valispace
% 4b) ValispacePushDataset("MySat.Mass",dataset)
% push Dataset to Valispace
% 4b) Note: The first row is x value and the second row is y
value
% 5) ValispaceGetMatrix(217)
% get Matrix Values

```

```

% 6) ValispacePushMatrix(217,[2,3;4,5])
% push Matrix Values
%
% Please note: Until you run "clear all" the Values be used from
your last ValispacePull() call.
%
% ValispaceGetVali() / ValispaceGetValue() /
ValispacePushValue() work with the argument as a string (name)
or integer (id)
% i.e. ValispaceGetValue("MySat.Mass") and
ValispaceGetValue(217)
% When using these functions with an integer id, step 2) can be
skipped.
% In this case the WebInterface will be accessed with every
individual call
%
% Both ValispacePushValue() and ValispacePushMatrix() can also
push formulas (e.g. $MySat.Mass*5) instead of values

%% IMPORT DATA FROM VALISPACE Solar panel sizing parameters

A_sc=ValispaceGetValue();%[m^2]area solar cell
N_sc_x=ValispaceGetValue();%number solar cell
N_sc_y=ValispaceGetValue();%number solar cell
N_sc_xneg=ValispaceGetValue();%number solar cell

P_Sun=ValispaceGetValue();%1367;%[W/m^2] Solar irradiance
eta=ValispaceGetValue();% Solar cell efficiency BOL
Id=ValispaceGetValue();%MPPT efficiency
Theta=ValispaceGetValue();%[deg] sun angle incidence
Sat_life=ValispaceGetValue();% year
degrad_year=ValispaceGetValue();
rule_of_thumb_k=ValispaceGetValue();%number in %/100

% Battery data
time_step=ValispaceGetValue();%[sec]
S_C_Power_consumption=ValispaceGetValue();%[W]
IBC=ValispaceGetValue();%[Wh] initial battery capacity
IBE=ValispaceGetValue();%[J] initial battery energy
BCE=ValispaceGetValue();% [%] battery charge efficiency
BDE=ValispaceGetValue();% [%] battery discharge efficiency
SOC_limit= ValispaceGetValue();% [%]
%% OPERATION 1

%Solar panel sizing parameters
P_sc_x=A_sc*N_sc_x*P_Sun*eta*Id;%[W] generated power BOL
P_sc_y=A_sc*N_sc_y*P_Sun*eta*Id;
P_sc_xneg=A_sc*N_sc_xneg*P_Sun*eta*Id;
L_d=(1-degrad_year)^Sat_life;%life degradation

P_EOL_x=P_sc_x*L_d;%[W] generated power EOL per square meter
P_EOL_y=P_sc_y*L_d;
P_EOL_xneg=P_sc_xneg*L_d;

```

```

OAP_SA_x=P_EOL_x*rule_of_thumb_k;%[W]OAP
OAP_SA_y=P_EOL_y*rule_of_thumb_k;
OAP_SA_xneg=P_EOL_xneg*rule_of_thumb_k;

A_sa_x=A_sc*N_sc_x;%[m^2] area solar array
A_sa_y=A_sc*N_sc_y;
A_sa_xneg=A_sc*N_sc_xneg;

degrad_year=0.015;
L_d=(1-degrad_year)^Sat_life;%life degradation
%% SUN POINTING ATTITUDE EVALUATION
% attitude identification in Sun Pointing
flag_sun_pointing=input('Would you like to load the Sun Vector
Matrix? If yes insert 1,If no insert 0: ');
if flag_sun_pointing==0

t=[0:0.01:1];%cosine loss cos(theta)
l=length(t);
Cosine_loss_matrix=zeros(l,6);%coloumn: x y z -x -y -z
P_SUN_IN_sp_matrix=zeros(l,6);
P_SUN_IN_sp_matrix_tot=[];
for i=1:(l)
    Cosine_loss_matrix(i,2)=t(i);
    Cosine_loss_matrix(i,4)=1-t(i);
    P_Sun_IN_sp_matrix(i,1)=0;
    P_Sun_IN_sp_matrix(i,2)=P_EOL_y*Cosine_loss_matrix(i,2);
    P_Sun_IN_sp_matrix(i,3)=0;
    P_Sun_IN_sp_matrix(i,4)=P_EOL_xneg*Cosine_loss_matrix(i,4);
    P_Sun_IN_sp_matrix(i,5)=0;
    P_Sun_IN_sp_matrix(i,6)=0;

    P_SUN_IN_sp_matrix_tot(i,1)= P_Sun_IN_sp_matrix(i,1)...
        +P_Sun_IN_sp_matrix(i,2)+P_Sun_IN_sp_matrix(i,4);
end

[P_SUN_IN_sp_max,row_index_max]=max(P_SUN_IN_sp_matrix_tot);

Sun_p_vec=ones(size(Date_v,1),6);
for i=1:6

Sun_p_vec(:,i)=Sun_p_vec(:,i)*Cosine_loss_matrix(row_index_max,i)
end
% Sun pointing vector reduction
Sun_p_vec_rid=[];
for i=1:N_rows
    Sun_p_vec_rid(i,:)=Sun_p_vec(i,:);
end
%creation of P SUN IN sun pointing
P_Sun_IN_sp=zeros(size(Sun_vector_rid,1),6);
for j=1:N_rows

```

```

%Sun pointing
%+x
if Sun_p_vec_rid(j,1)>0
    P_Sun_IN_sp(j,1)=P_EOL_x*Sun_p_vec_rid(j,1);
else
    P_Sun_IN_sp(j,1)=0;
end

%+y
if Sun_p_vec_rid(j,2)>0
    P_Sun_IN_sp(j,2)=P_EOL_y*Sun_p_vec_rid(j,2);
else
    P_Sun_IN_sp(j,2)=0;
end

%-x
if Sun_p_vec_rid(j,4)>0
    P_Sun_IN_sp(j,4)=P_EOL_xneg*Sun_p_vec_rid(j,4);
else
    P_Sun_IN_sp(j,4)=0;
end

end

else Sun_p_matrix=load('Sun_Vector_Sun_pointing') %values from
STK N_coloumns=3
    % Sun pointing vector reduction
Sun_p_vec_rid=[];
for i=1:N_rows
    Sun_p_vec_rid(i,:)=Sun_p_matrix(i,:);
end

%creation of P SUN IN sun pointing
P_Sun_IN_sp=zeros(size(Sun_vector_rid,1),6);
for j=1:N_rows

%Sun pointing
%+x
if Sun_p_vec_rid(j,1)>0
    P_Sun_IN_sp(j,1)=P_EOL_x*Sun_p_vec_rid(j,1);
else
    P_Sun_IN_sp(j,1)=0;
end

%+y
if Sun_p_vec_rid(j,2)>0
    P_Sun_IN_sp(j,2)=P_EOL_y*Sun_p_vec_rid(j,2);
else
    P_Sun_IN_sp(j,2)=0;
end

%-x
if (-1*Sun_p_vec_rid(j,1))>0

```



```

        P_Sun_IN_sp(j,4)=P_EOL_xneg*(-1*Sun_p_vec_rid(j,1));
else
    P_Sun_IN_sp(j,4)=0;
end
end
end
P_sun_IN_sp_TOT=mean(P_Sun_IN_sp(:,1)+P_Sun_IN_sp(:,2)+P_Sun_IN_
sp(:,4));
% OPERATION 2 Sun pointing
% operation to calculate OAP using an average value
P_Sun_sp_IN_reduced=[];
for i=1:6
    for j=1:N_rows
        P_Sun_sp_IN_reduced(j,i)=P_Sun_IN_sp(j,i);
    end
end
OAP=mean(P_Sun_sp_IN_reduced);

%nadir pointing
%+x
Energy_IN_sp_x=[];
for i=1:N_rows
    Energy_IN_sp_x(i,1)=P_Sun_IN_sp(i,1)*time_step;
end

Energy_SUM_sp_x=[];
Energy_SUM_sp_x(1,1)=Energy_IN_sp_x(1,1);
for i=2:N_rows

Energy_SUM_sp_x(i,1)=Energy_IN_sp_x(i,1)+Energy_SUM_sp_x(i-1,1);
end

Wh_sp_x=[];
for i=1:N_rows
    Wh_sp_x(i,1)=Energy_SUM_sp_x(i,1)/3600;
end

%y
Energy_IN_sp_y=[];
for i=1:N_rows
    Energy_IN_sp_y(i,1)=P_Sun_IN_sp(i,2)*time_step;
end

Energy_SUM_sp_y=[];
Energy_SUM_sp_y(1,1)=Energy_IN_sp_y(1,1);
for i=2:N_rows

Energy_SUM_sp_y(i,1)=Energy_IN_sp_y(i,1)+Energy_SUM_sp_y(i-1,1);
end

Wh_sp_y=[];

```

```

for i=1:N_rows
    Wh_sp_y(i,1)=Energy_SUM_sp_y(i,1)/3600;
end

%-x
Energy_IN_sp_x_neg=[];
for i=1:N_rows
    Energy_IN_sp_x_neg(i,1)=P_Sun_IN_sp(i,2)*time_step;
end

Energy_SUM_sp_x_neg=[];
Energy_SUM_sp_x_neg(1,1)=Energy_IN_sp_x_neg(1,1);
for i=2:N_rows

    Energy_SUM_sp_x_neg(i,1)=Energy_IN_sp_x_neg(i,1)+Energy_SUM_sp_x_
_neg(i-1,1);
end

Wh_sp_x_neg=[];
for i=1:N_rows
    Wh_sp_x_neg(i,1)=Energy_SUM_sp_x_neg(i,1)/3600;
end

%tot

Pin_Pmedium_sp_tot=[];% power subtraction
for i=1:N_rows

    Pin_Pmedium_sp_tot(i)=(P_Sun_IN_sp(i,1)+P_Sun_IN_sp(i,2)+P_Sun_I
N_sp(i,4))-S_C_Power_consumption;
end

delta_Energy_sp_tot=[];
for i=1:N_rows
    delta_Energy_sp_tot(i)=Pin_Pmedium_sp_tot(i)*time_step;
end

E_sp_tot=[];
if (delta_Energy_sp_tot(1)+IBE)<IBE
    E_sp_tot(1)=(delta_Energy_sp_tot(1)+IBE);
else
    E_sp_tot(1)=IBE;
end
for i=2:N_rows
    if (delta_Energy_sp_tot(i)+E_sp_tot(i-1))<IBE
        E_sp_tot(i)=(delta_Energy_sp_tot(i)+E_sp_tot(i-1));
    else
        E_sp_tot(i)=IBE;
    end
end

SOC_Wh_sp_tot=[];
for i=1:N_rows

```

```

        SOC_Wh_sp_tot(i)=E_sp_tot(i)/3600;
end
SOC_perc_sp_tot=100*SOC_Wh_sp_tot/(IBC);
SOC_perc_sp_tot_mean=mean(SOC_perc_sp_tot);
DOD_sp_tot=100*(1-SOC_Wh_sp_tot/(IBC));
MAX_DOD_sp_tot=max(DOD_sp_tot)

%% OPERATION 3 nadir pointing

% x y z -x -y -z creation of P SUN IN for nadir pointing
P_Sun_IN=zeros(size(Sun_vector_rid,1),6);
for j=1:N_rows
    %Nadir pointing
    %+x
    if Sun_vector_rid(j,1)>0
        P_Sun_IN(j,1)=P_EOL_x*Sun_vector_rid(j,1);
    else
        P_Sun_IN(j,1)=0;
    end
    %+y
    if Sun_vector_rid(j,2)>0
        P_Sun_IN(j,2)=P_EOL_y*Sun_vector_rid(j,2);
    else
        P_Sun_IN(j,2)=0;
    end

    %-x
    if (-1*Sun_vector_rid(j,1))>0
        P_Sun_IN(j,4)=P_EOL_xneg*(-1*Sun_vector_rid(j,1));
    else
        P_Sun_IN(j,4)=0;
    end
end

end

% +x P_SUN_IN
figure (2)
plot(time_rid,P_Sun_IN(:,1),time_rid,P_Sun_IN_sp(:,1))
xlabel('time[s]')
ylabel('Sun Power In +x [W]')
title ('Power produced by solar panels')
legend ('Nadir pointing','Sun pointing')
hold on
% +y P_SUN_IN
figure (3)
plot(time_rid,P_Sun_IN(:,2),time_rid,P_Sun_IN_sp(:,2))
xlabel('time[s]')
ylabel('Sun Power In +y [W]')
title ('Power produced by solar panels')
legend ('Nadir pointing','Sun pointing')
hold on
% -x P_SUN_IN
figure (4)

```

```

plot(time_rid,P_Sun_IN(:,4),time_rid,P_Sun_IN_sp(:,4))
xlabel('time[s]')
ylabel('Sun Power In -x [W]')
title ('Power produced by solar panels')
legend ('Nadir pointing','Sun pointing')
hold on

% operation to calculate OAP using an average value
P_Sun_IN_reduced=[];
for i=1:6
    for j=1:N_rows
        P_Sun_IN_reduced(j,i)=P_Sun_IN(j,i);
    end
end
OAP=mean(P_Sun_IN_reduced);

%nadir pointing
%+x
Energy_IN_x=[];
for i=1:N_rows
    Energy_IN_x(i,1)=P_Sun_IN(i,1)*time_step;
end

Energy_SUM_x=[];
Energy_SUM_x(1,1)=Energy_IN_x(1,1);
for i=2:N_rows
    Energy_SUM_x(i,1)=Energy_IN_x(i,1)+Energy_SUM_x(i-1,1);
end

Wh_x=[];
for i=1:N_rows
    Wh_x(i,1)=Energy_SUM_x(i,1)/3600;
end

%y
Energy_IN_y=[];
for i=1:N_rows
    Energy_IN_y(i,1)=P_Sun_IN(i,2)*time_step;
end

Energy_SUM_y=[];
Energy_SUM_y(1,1)=Energy_IN_y(1,1);
for i=2:N_rows
    Energy_SUM_y(i,1)=Energy_IN_y(i,1)+Energy_SUM_y(i-1,1);
end

Wh_y=[];
for i=1:N_rows
    Wh_y(i,1)=Energy_SUM_y(i,1)/3600;
end

%-x
Energy_IN_x_neg=[];

```

```

for i=1:N_rows
    Energy_IN_x_neg(i,1)=P_Sun_IN(i,2)*time_step;
end

Energy_SUM_x_neg=[];
Energy_SUM_x_neg(1,1)=Energy_IN_x_neg(1,1);
for i=2:N_rows

Energy_SUM_x_neg(i,1)=Energy_IN_x_neg(i,1)+Energy_SUM_x_neg(i-
1,1);
end

Wh_x_neg=[];
for i=1:N_rows
    Wh_x_neg(i,1)=Energy_SUM_x_neg(i,1)/3600;
end

Pin_Pmedium_tot=[];% power subtraction
for i=1:N_rows

Pin_Pmedium_tot(i)=(P_Sun_IN(i,1)+P_Sun_IN(i,2)+P_Sun_IN(i,4))-
S_C_Power_consumption;
end

delta_Energy_tot=[];
for i=1:N_rows
    delta_Energy_tot(i)=Pin_Pmedium_tot(i)*time_step;
end

E_tot=[];
if (delta_Energy_tot(1)+IBE)<IBE
    E_tot(1)=(delta_Energy_tot(1)+IBE);
else
    E_tot(1)=IBE;
end
for i=2:N_rows
    if (delta_Energy_tot(i)+E_tot(i-1))<IBE
        E_tot(i)=(delta_Energy_tot(i)+E_tot(i-1));
    else
        E_tot(i)=IBE;
    end
end

SOC_Wh_tot=[];
for i=1:N_rows
    SOC_Wh_tot(i)=E_tot(i)/3600;
end
SOC_perc_tot=100*SOC_Wh_tot/(IBC);
SOC_perc_tot_mean=mean(SOC_perc_tot);
DOD_tot=100*(1-SOC_Wh_tot/(IBC));
MAX_DOD_tot=max(DOD_tot)

%% plot spacecraft

```

```

figure(5)
plot(time_rid,SOC_perc_tot)
xlabel('time')
ylabel('Total SOC percentage')
title('Total SOC')
hold on

i=1;
while SOC_perc_tot(i) > SOC_limit && i< N_rows
    SOC_perc_tot_pos_v(i)=SOC_perc_tot(i);
    time_pos_v_tot(i)=time_rid(i);
    i=i+1;
end
data0tot=Date_v(i,:);
'the solar panels do not generate power in date '
datestr(data0tot)

figure(6)
plot(time_pos_v_tot,SOC_perc_tot_pos_v)
xlabel('time')
ylabel('Total SOC percentage')
title('Total SOC till 75%')
hold on

i=1;
while SOC_perc_sp_tot(i) > SOC_limit && i< N_rows
    SOC_perc_tot_sp_pos_v(i)=SOC_perc_sp_tot(i);
    time_pos_v_sp_tot(i)=time_rid(i);
    i=i+1;
end
data0tot=Date_v(i,:);
'the solar panels do not generate power in date '
datestr(data0tot)

figure(7)
plot(time_pos_v_sp_tot,SOC_perc_tot_sp_pos_v)
xlabel('time')
ylabel('Sun Pointing Total SOC percentage')
title('Total SOC till 75% for Sun Pointing')
hold on

%% EXPORT DATA IN VALISPACE

ValispacePull()
ValispacePushValue("",MAX_DOD_tot)
ValispacePushValue("",SOC_perc_tot_mean)
ValispacePushValue("",OAP)

```

Bibliography

- [1] NASA, “CubeSats Overview”, <https://www.nasa.gov>.
- [2] S. T. Luther, “SYSML BASED CUBESAT MODEL DESIGN AND INTEGRATION WITH THE HORIZON SIMULATION FRAMEWORK”, 2016.
- [3] ESA, “Technology CubeSats”, <https://www.esa.int>.
- [4] E. Canbulut, V. Aydingül, and B. Yaglioglu, “APPLICATION OF MODEL-BASED SYSTEMS ENGINEERING WITH SYSML IN A SMALL SATELLITE PROJECT”, 2019, [Online], Available: <https://www.researchgate.net/publication/336130617>.
- [5] M. Duzzi, “SPACECRAFT RENDEZVOUS AND DOCKING USING ELECTROMAGNETIC INTERACTIONS”.
- [6] L. Yazhong, Z. Jin, and T. Guojin, “Survey of orbital dynamics and control of space rendezvous”, *Chinese Journal of Aeronautics*, 2013.
- [7] L. Patrioli and S. Corpino, “D1.1 - Assessment of the SROC mission and preliminary functional specification”, 2020.
- [8] eoPortal Directory, “Space Rider”, <https://directory.eoportal.org/web/eoportal/satellite-missions/s/space-rider>.
- [9] ASI, “SPACE RIDER”, <https://www.asi.it/trasporto-spaziale/space-rider/>.
- [10] ESA, “USER GUIDE for the SPACE RIDER Re-usable Free Flyer Platform integrated with VEGA C”, 2021.
- [11] ESA, “Space Rider Europe’s first reusable space transportation system”, https://www.esa.int/Enabling_Support/Space_Transportation/Space_Rider.
- [12] Dimeas group and S. Corpino, “DEVELOPMENT OF THE SPACE RIDER OBSERVER CUBE PHASE B1”, 2020.
- [13] S. Corpino, “*01NZLMT-PROGETTO DI MISSIONI E SISTEMI SPAZIALI*”, *lezioni*, 2021.
- [14] D. D. Walden, G. J. Roedler, K. J. Forsberg, R. D. Hamelin, and T. M. Shortell, “*Systems Engineering Handbook*”, 2015.
- [15] S. ECSS, *ECSS-M-ST-10C Rev. 1 “Space project management-Project planning and implementation-ECSS Secretariat ESA-ESTEC Requirements & Standards Division Noordwijk”*, The Netherlands, 2009.
- [16] S. P. Alai, “EVALUATING ARCADIA/CAPELLA VS. OOSEM/SYSML FOR SYSTEM ARCHITECTURE DEVELOPMENT”, 2019.
- [17] P. Pontillo, “Studio sullo sviluppo del Digital Twin partendo dal Model-Based System Engineering”, 2021.
- [18] NASA, “*NASA Systems Engineering Handbook*”, vol. Revision 2, 2016.
- [19] M. Chami, A. Aleksandraviciene, A. Morkevicius, and J.-M. Bruel, “Towards Solving MBSE Adoption Challenges: The D3 MBSE Adoption Toolbox”, *INCOSE*

International Symposium, vol. 28, no. 1, pp. 1463–1477, Jul. 2018, doi: 10.1002/j.2334-5837.2018.00561.x.

- [20] A. Butting, S. Konar, B. Rumpe, and A. Wortmann, “Teaching model-based systems engineering for industry 4.0: Student challenges and expectations”, in *21st ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings, MODELS-Companion 2018*, Oct. 2018, pp. 74–81, doi: 10.1145/3270112.3270122.
- [21] E. E. Bürger, “A CONCEPTUAL MBSE FRAMEWORK FOR SATELLITE AIT PLANNING”, 2019, [Online], Available: <http://urlib.net/8JMKD3MGP3W34R/3S3JPME>.
- [22] M. Ingham *et al.*, “A Model-Based Approach to Engineering Behavior of Complex Aerospace Systems”.
- [23] L. E. Hart Lockheed Martin and G. LauraEHart, “Introduction To Model-Based System Engineering (MBSE) and SysML”, 2015.
- [24] P. Roques, “*Systems Architecture Modeling with the Arcadia Method- A practical guide to Capella*”, 2018.
- [25] J. A. Estefan, “*Survey of Model-Based Systems Engineering (MBSE) Methodologies*”, 2008.
- [26] A. Alberts and C. Zingel, “Challenges of Model-Based Systems Engineering: A Study towards Unified Term Understanding and the State of Usage of SysML”, 2013.
- [27] A. Morkevicius, A. Aleksandraviciene, D. Mazeika, L. Bisikirskiene, and Z. Strolia, “MBSE Grid: A Simplified SysML-Based Approach for Modeling Complex Systems”, 2017.
- [28] H.-P. Hoffmann, “Systems Engineering Best Practices with the Rational Solution for Systems and Software Engineering-Model-Based Systems Engineering with Rational Rhapsody and Rational Harmony for Systems Engineering”, 2011.
- [29] IBM, “IBM Rhapsody”, <https://www.ibm.com/it-it/products/systems-design-rhapsody/details>.
- [30] T. Peterson and W. D. Schindel, “Model-Based System Patterns for Automated Ground Vehicle Platforms”, *INCOSE International Symposium*, vol. 25, no. 1, pp. 388–403, Oct. 2015, doi: 10.1002/j.2334-5837.2015.00070.x.
- [31] Vitechcorp, “STRATA: A Layered Approach to Project Success”, <https://www.vitechcorp.com/strata-methodology>.
- [32] Vitechcorp, “Core Suite”, <https://www.vitechcorp.com/>.
- [33] Vitechcorp, “Intro to MBSE with Genesys”, <https://www.vitechcorp.com/intro-to-mbse-with-genesys/>.
- [34] M. D. Ingham, R. D. Rasmussen, M. B. Bennett, and A. C. Moncada, “Engineering Complex Embedded Systems with State Analysis and the Mission Data System”.

- [35] Safeware Engineer Corporation, “SpecTRM”, <http://www.safeware-eng.com>.
- [36] T. Weilkiens, J. G. Lamm, S. Roth, and M. Walker, “*MODEL-BASED SYSTEM ARCHITECTURE*”, 2016.
- [37] Y. Grobshtein, V. Perelman, E. Safra, and D. Dori, “Systems Modeling Languages: OPM Versus SysML”.
- [38] D. Dori, “Model-Based Systems Engineering with OPM and SysML”, 2016.
- [39] D. Dori, “Chapter 1 OPCAT-An Object-Process CASE Tool for OPM-Based Conceptual Modelling Digital radiograph interpretation View project”, 2010, [Online], Available: <https://www.researchgate.net/publication/268366929>.
- [40] S. Friedenthal, A. Moore, and R. Steiner, “Residential Security System Example Using the Object-Oriented Systems Engineering Method”, in *A Practical Guide to SysML*, Elsevier, 2015, pp. 417–504, doi: 10.1016/b978-0-12-800202-5.00017-5.
- [41] INCOSE, “Object-Oriented SE Method Mission & Objectives About The Method”, <https://www.incose.org>.
- [42] MathWorks, “System Composer”, <https://it.mathworks.com/products/system-composer>.
- [43] Dassault Systemes, “Cameo System Modeler”, <https://www.3ds.com/products-services/catia/products/no-magic/cameo-systems-modeler>.
- [44] Dassault Systemes, “MagicDraw”, <https://www.3ds.com/products-services/catia/products/no-magic/magicdraw>.
- [45] H. P. de Koning, R. Paris Lopez, T. Bieler, A. Braukhane, R. Richardson, and T. Stoitsev, “European Concurrent Engineering Model for System-of-Systems ArchitectureStudies”, 2014.
- [46] A. Pickering, S. Gerené, and H. de Koning, “Status Draft Document Type UM Distribution OCDT Community OCDT ConCORDE Software User Manual”, 2015, [Online], Available: www.esa.int.
- [47] I. Ferreira and H. P. de Koning, “Open Concurrent Design Tool: ESA Community Open Source Ready to Go!”, 2014.
- [48] Isis Papyrus, “Isis Papyrus”, <https://www.isis-papyrus.com/e15/pages/software/platform-concept.html>.
- [49] ESA, “Valispace: the smart collaboration platform for engineers”, <https://www.esa.int>.
- [50] Valispace, “Introduction to Valispace”, <https://valispace.zendesk.com>.
- [51] J. L. Voirin, S. Bonnet, and D. Exertier, “A Model-Based Engineering Method for System, Software and Hardware Architectural Design-A method best-supported by”, 2015.
- [52] G. Ammirante, F. Corradino, S. Corpino, and F. Nichele, “D3.1 - Feasibility and Preliminary Specification”, 2020.

- [53] F. Stesina, "COMMUNICATION SYSTEM SISTEMI AEROSPAZIALI-03GKZMT".
- [54] Fishercome, "Isotropic Receive Level", <https://www.fishercom.xyz/network-management/info-jvy.html>.
- [55] L. J. Ippolito, "Satellite Communications Systems Engineering - Atmospheric Effects, Satellite Link Design and System Performance", 2009.
- [56] ECSS, "ECSS-E-ST-10-24C Space engineering Interface management ECSS Secretariat ESA-ESTEC Requirements & Standards Division Noordwijk, The Netherlands", 2015.